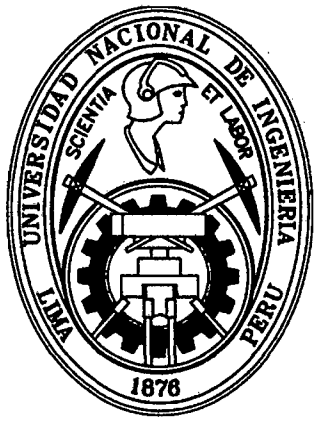
UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA INDUSTRIAL Y DE SISTEMAS



"ALGORITMO GENÉTICO EN JAVA PARA LA OPTIMIZACIÓN DEL DISEÑO DE RESORTES HELICOIDALES DE COMPRESION"

TESIS

PARA OPTAR EL GRADO ACADÉMICO DE MAESTRO EN CIENCIAS CON MENCIÓN EN INGENIERÍA DE SISTEMAS

ELABORADO POR

JUAN GAMARRA MORENO

ASESOR

Mg. ALFREDO RAMOS MUÑOZ

LIMA – PERÚ

2012

Asesor:

Mg. Alfredo Ramos Muñoz

A mis padres Silvestre y Amanda, bajo su cobijo experimenté el amor, la honestidad, la perseverancia, el optimismo y la lucha por nobles ideales. A mi esposa Kenny e hijo Fernando, por toda su paciencia, amor y comprensión. A mis suegros Oswaldo y Estela por todo su apoyo.

ÍNDICE

CAPÍTULO II

MARCO TEÓRICO Y CONCEPTUAL

CAPÍTULO III

COMPONETE AGJAVA APLICADO AL ALGORITMO GENÉTICO PARA LA OPTIMIZACIÓN DEL DISEÑO DE RESORTES HELICOIDALES DE COMPRESIÓN

CAPÍTULO IV

PRESENTACIÓN Y ANÁLISIS DE RESULTADOS

ANEXOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En esta tesis se desarrolló AGJava (Algoritmo Genético en Java) que es un componente Java desarrollado con la Ingeniería de Software Orientada a Objetos para la aplicación de los Algoritmos Genéticos a la solución de problemas de optimización donde el espacio de búsqueda sea ilimitado o demasiado grande. Para demostrar la aplicabilidad de AGJava se resolvió un problema práctico de optimización relacionado al diseño de resortes helicoidales de compresión en la industria mecánica y se le comparó con un método de diseño tradicional.

ABSTRACT

This thesis developed AGJava (Genetic Algorithm in Java) which is a Java component developed with the Object Oriented Software Engineering for the application of genetic algorithms to solve optimization problems where the search space is unlimited or too large. To demonstrate the applicability of AGJava was solved a practical problem related to the optimization-of the Design of Helical Compression Springs in mechanical engineering and was compared with a traditional design method.

PALABRAS CLAVE

Componente para Algoritmos Genéticos, Optimización, Problemas con espacio de soluciones ilimitado, Componente AGJava, AGJava (Algoritmos Genéticos en Java), Cromosomas AGJava, Operadores Genéticos AGJava, Población AGJava, Tipos de variable AGJava, Diseño de resortes helicoidales de compresión.

INTRODUCCIÓN

Las aplicaciones que se le pueden dar a los algoritmos genéticos como un método de optimización crecen aceleradamente, esto originó que se desarrolle una vasta variedad de código para los algoritmos genéticos, sin embargo una de las dificultades con la que se encuentran investigadores y desarrolladores es la escasa (a veces ninguna) documentación de la implementación de estos paquetes y librerías, siendo esto una gran dificultad para su reutilización y extensibilidad, para su uso en un proyecto serio. Esta tesis trata acerca del desarrollo del Componente AGJava, componente cuyo propósito es el de servir de base para el desarrollo de algoritmos genéticos. Para probar la funcionalidad del componente se desarrolla un algoritmo genético para la optimización a problemas de diseño de resortes helicoidales de compresión, para comparar la mejora en la obtención de los resultados se compara con los obtenidos con un método de diseño tradicional. La tesis se divide contiene los siguientes capítulos:

En el Capítulo I, se tratan los aspectos generales de la tesis donde se define y formula el problema, se describen los objetivos y se plantean las hipótesis de investigación.

El Capítulo II, describe el Marco Teórico y Conceptual de la tesis donde se tratan aspectos relacionados al desarrollo basado en componentes, algoritmos genéticos y diseño de resortes helicoidales de compresión.

En el Capítulo III, se explica detalladamente el desarrollo del Componente AGJava, también se expone acerca del uso del componente para la elabora-

ción de un algoritmo genético para la optimización del diseño de resortes helicoidales de compresión.

En el Capítulo IV se presentan los resultados del cuasi experimento que prueba al algoritmo genético basado en el componente AGJava con diez casos de aplicación para contrastar las hipótesis planteadas.

En los anexos se expone el código Java del paquete util y ag del Componente agjava, de las clases del resorte, del algoritmo genético de optimización para el diseño de resortes helicoidales de compresión y de los programas para el diseño tradicionales. Además se muestra la ejecución del cuasi experimento diseñado para probar la hipótesis.

CAPÍTULO I

ASPECTOS GENERALES DE LA TESIS

1.1. DIAGNÓSTICO Y ENUNCIADO DEL PROBLEMA

Durante las dos últimas décadas ha habido un creciente interés en los algoritmos que están basados en el principio de la evolución (supervivencia del más capaz). Un término común, aceptado recientemente, se refiere a tales técnicas como algoritmos evolutivos (EA) (o métodos de computación evolutiva). Los más conocidos algoritmos en esta clase incluyen los algoritmos genéticos, programación evolutiva, estrategias evolutivas, y programación genética. Existen también varios sistemas híbridos que incorporan varias características de los paradigmas anteriormente mencionados, y por lo tanto es difícil clasificarlos; de todas formas, nos referimos a ellas como métodos de computación evolutiva (EC).

El campo de la computación evolutiva ha alcanzado un estado de alguna madurez. Existen varias conferencias bien establecidas que atraen cientos de participantes (GEM'12 - The 2012 International Conference on Genetic and Evolutionary Methods, 12th International Conference on Parallel Problem Solving From Nature - PPSN 2012, IEEE Congress on Evolutionary Computation - IEEE CEC 2013). También hay muchos talleres, sesiones especiales, y conferencias locales cada año, en todo el mundo. Las revistas, Evolutionary Computation (MIT Press) y IEEE Transactions on Evolutionary Computation entre otras, están dedicadas completamente a las técnicas de computación evolutiva. Otras revistas organizan temas especializados en

temas de computación evolutiva. Muchos libros, artículos tutoriales excelentes y reportes técnicos ofrecen más o menos bibliografías completas de este campo.

Existen problemas en que los métodos de optimización tradicionales, basados en algoritmos de búsqueda exhaustiva o en programación matemática (lineal, cuadrática, mixta, gradientes) no se pueden aplicar cuando existe gran cantidad de datos o la complejidad los haga inapropiados de resolver por tales algoritmos, este tipo de problemas da paso a métodos heurísticos y evolutivos entre los que destacan los algoritmos genéticos.

Los algoritmos genéticos son un recurso común en ámbitos tan dispares como la Ingeniería, la Medicina, la Astrofísica, las Telecomunicaciones o la Economía. Y, en general, en todas aquellas actividades en las que el data mining y la resolución de problemas complejos de tipo no lineal tengan alguna importancia. Un ejemplo temprano de este método de optimización lo encontramos en la búsqueda de conexiones óptimas en grandes redes. En general podemos definir los algoritmos evolutivos como métodos robustos de optimización diseñados para resolver problemas complejos con elevado número de elementos, restricciones y variables y en los que, por lo general, coexisten una o varias soluciones optimas que no pueden aproximarse por los métodos tradicionales propios de la programación lineal. El tipo de problemas característico de estos algoritmos es aquel en el que partiendo de una población de "n" individuos (propuestas de diseño, estrellas conocidas de la galaxia, historiales clínicos, incidencia de accidentes de tráfico, hábitos de consumo, históricos de cotizaciones, etc.) se busca un conjunto de parámetros que minimicen o maximicen una función de adaptación (capacidad) u objetivo. Como en los algoritmos clásicos, el proceso se realiza en sucesivos pasos o iteraciones en las que se van combinando y seleccionando aquellos juegos de parámetros que más se aproximen al criterio establecido. Siendo la principal diferencia el método meta-heurístico de búsqueda y selección de los mejores parámetros. En definitiva, estamos ante métodos no determinís-

ticos que permiten llegar a soluciones distintas recorriendo innumerables caminos y bifurcaciones en cada optimización.

Por otra parte los algoritmos genéticos consiguen eludir mejor el problema de la curva de ajuste que, a diferencia de otros métodos de optimización en los que iterando de manera lineal se avanza hacia una solución única (siempre la misma) determinada por el criterio objetivo elegido, en los algoritmos evolutivos se mantiene una población de posibles candidatos potenciales que cubre una amplia "región de valor" (o zona robusta). Esta diversidad de soluciones alternativas y viables, unida a la presencia de mecanismos para evitar caer en lo que se denomina "óptimo local", es un factor determinante para minimizar el riesgo de sobre la optimización.

Con el creciente interés sobre los algoritmos evolutivos y sus aplicaciones como un método de optimización se desarrollaron varios paquetes y bibliotecas de algoritmos genéticos, para diversos lenguajes de programación. Sin embargo una de las dificultades con la que se encuentran varios investigadores y desarrolladores es el hecho de que los códigos proporcionados tienen escasa (a veces ninguna) documentación de la implementación de estos paquetes y librerías, siendo esto una gran dificultad para su reutilización y extensibilidad, para su uso en un proyecto serio. El desarrollo de un componente de código abierto (bien desarrollado y documentado) para algoritmos genéticos que permita aplicarlos y extenderlos fácilmente permitirá el desarrollo de algoritmos genéticos para resolver problemas de optimización donde el espacio de soluciones es muy extenso o ilimitado.

1.2. DEFINICIÓN Y FORMULACIÓN DEL PROBLEMA DE INVESTIGACIÓN

Las aplicaciones que se le pueden dar a los algoritmos genéticos como un método de optimización crecen aceleradamente, así se desarrollaron varios programas, paquetes y bibliotecas de algoritmos genéticos con diversos lenguajes de programación, pero sin embargo una de las principales dificultades con la que se encuentran los investigadores y desarrolladores de softwa-

re es la escasa (o nula) documentación de la implementación de estos, representando esto una dificultad mayor para su reutilización y extensibilidad en un proyecto serio. Por consiguiente se desarrollará AGJava (Algoritmo Genético en Java) que será un componente Java desarrollado con la Ingeniería de Software Orientada a Objetos para la aplicación de los Algoritmos Genéticos en la solución de problemas de optimización donde el espacio de búsqueda sea ilimitado o demasiado grande. Para demostrar la aplicabilidad de AGJava se deberán resolver un problema práctico de optimización relacionado al diseño de resortes helicoidales de compresión en la industria mecánica y que será comparado a un método de diseño alternativo.

1.2.1. PROBLEMA GENERAL

¿En qué medida el componente AGJava mediante algoritmos genéticos mejora la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión?

1.2.2. PROBLEMAS ESPECÍFICOS

¿En qué medida los cromosomas del componente AGJava permiten representar las variables para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?

¿En qué medida la población del componente AGJava permite trabajar con más de una solución en una generación para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?

¿En qué medida los operadores genéticos del componente AGJava permiten crear nuevas soluciones para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?

¿En qué medida las vistas del componente AGJava permiten representar diferentes tipos de variables para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Desarrollar el Componente AGJava y aplicarlo a un algoritmo genético para la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión.

1.3.2. OBJETIVOS ESPECÍFICOS

- Desarrollar la funcionalidad de los cromosomas en el componente AGJava para representar las variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.
- Desarrollar la funcionalidad de la población del componente AGJava para trabajar con más de una solución en una generación en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.
- Desarrollar la funcionalidad de los operadores genéticos del componente AGJava para crear nuevas soluciones en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.
- Desarrollar la funcionalidad de las vistas del componente AGJava para representar diferentes tipos de variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

1.4. HIPÓTESIS DE INVESTIGACIÓN

1.4.1. HIPÓTESIS

1.4.1.1. Hipótesis General

El Algoritmo Genético basado en el componente AGJava mejora la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión.

1.4.1.2. Hipótesis Especificas

- La funcionalidad de los cromosomas en el componente AGJava permite representar las variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

- La funcionalidad de la población del componente AGJava permite trabajar con más de una solución en una generación en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

- La funcionalidad de los operadores genéticos del componente AGJava permite crear nuevas soluciones en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

- La funcionalidad de las vistas del componente AGJava permite representar diferentes tipos de variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

1.4.2. IDENTIFICACIÓN Y OPERACIONALIZACIÓN DE VARIABLES

$V_{independiente}$: Algoritmo Genético basado en el Componente AGJava.

$V_{dependiente}$: Solución óptima de problema de Diseño de Resortes Helicoidales de Compresión

Variables	Indicadores	Índices
Algoritmo Genético basado en el Componente AGJava.	Cromosoma AGJava	Nro. de variables que representa en un problema.
	Operador Genético AGJava	Nro. de operadores genéticos que se pueden usar en los cromosomas representados.
	Población AGJava	Nro. de cromosomas de la población
	Tipo de variable AGJava	Nro. de tipos de variables
Solución óptima de Problema de Diseño de Resortes Helicoidales de Compresión	Volumen óptimo del resorte helicoidal de compresión.	Ratio de Número de problemas de diseño de resortes helicoidales de compresión resueltos por algoritmos genéticos con relación al método tradicional.

1.4.2.1. Algoritmo Genético basado en el Componente AGJava

El Algoritmo Genético desarrollado utiliza el Componente AGJava para solucionar problemas de optimización en el Diseño de Resortes Helicoidales de Compresión. Tiene en su estructura elementos que representan el tamaño de genes, generaciones, número de cromosomas, además crea los cromosomas de forma aleatoria y los hace evolucionar a través de varias generaciones con el uso de operadores genéticos. Tiene entradas que se utilizan para el Diseño de Resortes Helicoidales de Compresión, genera cromosomas que representa al diámetro del alambre, diámetro medio y número de bobinas activas del resorte. Devuelve como resultado los valores para las otras variables del resorte, buscando minimizar el volumen en sus soluciones.

Estas variables son: Tipo de material, tipo de servicio, tipo de extremo, tipo de sujeción, módulo del alambre del resorte, longitud libre, diámetro exterior, diámetro interior, diámetro del alambre, diámetro medio, fuerza en longitud comprimido, longitud instalado, fuerza en longitud instalado, fuerza en longitud libre, razón de resorte, índice de resorte, número total de bobinas, número de bobinas activas, número máximo de bobinas activas, espaciamiento, ángulo de espaciamiento, diámetro exterior en longitud comprimido, margen de bobina, factor de Wahl, carga máxima normal, esfuerzo o tensión en carga de operación, deflexión en longitud de operación, longitud de operación, longitud comprimido, tensión o esfuerzo en longitud comprimido, tensión de diseño, tensión máxima permisible, diámetro del orificio de instalación, diámetro de la varilla de instalación, razón crítica y volumen.

El Componente AGJava básicamente le permite al algoritmo genético crear cromosomas para representar la solución de un problema, además permite aplicar operadores genéticos sobre estos cromosomas, también establece una población con los cromosomas que conjuntamente representan varias soluciones múltiples y asimismo a través de vistas maneja los tipos diferentes de variables que se pueden presentar en el diseño, pudiendo ser enteras, reales o discretas.

1.4.2.2. Solución Óptima en problemas de Diseño de Resortes Helicoidales de Compresión.

Se considera óptimo el Diseño de Resortes Helicoidales de Compresión cuando este tiene la menor cantidad posible de un determinado material, así el volumen del material del resorte determina su costo. Consideramos óptimo un diseño cuando este tiene el menor volumen posible. El espacio de soluciones factibles es prácticamente ilimitado, en sí este óptimo que se encuentra en el espacio de soluciones factibles es una solución heurística que da solución al problema de diseño de resortes helicoidales de compresión, a diferencia de los métodos de optimización tradicional donde no se pueden resolver este tipo de problemas.

1.4.2.3. Cromosoma AGJava

Las variables que se utilizan en el diseño de resortes helicoidales de compresión deben tener la posibilidad de ser representadas en la solución del Diseño de Resortes Helicoidales de Compresión a través de un cromosoma. Para esta aplicación consideramos 3 genes (variables) que representarán respectivamente al diámetro del alambre del resorte (Dw), diámetro medio (Dm) y número de bobinas activas (Na) en un cromosoma.

1.4.2.4. Operador Genético AGJava

La soluciones factibles para optimizar el volumen en el diseño de resortes helicoidales de compresión deben cambiar para evolucionar en nuevas y

mejores soluciones, esto se logra a través de operadores genéticos que pueden ser aplicados al algoritmo genético en cuestión.

1.4.2.5. Población AGJava

Una solución en el diseño de resortes helicoidales de compresión, son los valores que se asignan en conjunto para las variables: tipo de material, tipo de servicio, tipo de extremo, tipo de sujeción, módulo del alambre del resorte, longitud libre, diámetro exterior, diámetro interior, diámetro del alambre, diámetro medio, fuerza en longitud comprimido, longitud instalado, fuerza en longitud instalado, fuerza en longitud libre, razón de resorte, índice de resorte, número total de bobinas, número de bobinas activas, número máximo de bobinas activas, espaciamiento, ángulo de espaciamiento, diámetro exterior en longitud comprimido, margen de bobina, factor de Wahl, carga máxima normal, esfuerzo o tensión en carga de operación, deflexión en longitud de operación, longitud de operación, longitud comprimido, tensión o esfuerzo en longitud comprimido, tensión de diseño, tensión máxima permisible, diámetro del orificio de instalación, diámetro de la varilla de instalación, razón crítica y volumen. La solución puede ser factible (cumple con todas las restricciones de diseño) o infactible (no cumple con una o mas restricciones de diseño).

Una iteración en el diseño de resortes helicoidales es el procedimiento que se sigue para obtener valores al total de variables resultado en una solución, si en una iteración se puede obtener más de una solución simultáneamente, diremos que hay más de una solución en una iteración en el diseño de resortes helicoidales de compresión. La población permite trabajar con más de una solución en una iteración.

1.4.2.6. Tipo de variables AGJava

Si bien un cromosoma permite representar una solución en el Diseño de Resortes Helicoidales de Compresión, este necesita ser interpretado, esto se logrará a través de la implementación de la clase Vista que permita manejar los diferentes tipos de variables de diseño. Las clases derivadas de Vista se

usan para imponer cierta interpretación en un cromosoma. La vista por defecto (tal como se provee por esta clase base), simplemente trata el cromosoma como una cadena de bits de genes simples, cada uno codificando un boolean. Interpretaciones más útiles se pueden codificar por las subclases.

1.5. MATRIZ DE CONSISTENCIA

Problema Principal	Objetivo General	Hipótesis Principal	Variables	Indicadores	Indices
¿En qué medida el componente AGJava mediante algoritmos genéticos mejora la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión?	Desarrollar el Componente AGJava y aplicarlo a un algoritmo genético para la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión.	El Algoritmo Genético basado en el componente AGJava mejora la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión.	Variable Independiente: Algoritmo Genético basado en el Componente AGJava..	Cromosoma AGJava	Nro. de variables que representa en un problema.
				Operador Genético AGJava	Nro. de operadores genéticos que se pueden usar en los cromosomas representados.
Problemas Secundarios	Objetivos Secundarios	Hipótesis Secundarias			
¿En qué medida los cromosomas del componente AGJava permiten representar las variables para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?	Desarrollar la funcionalidad de los cromosomas en el componente AGJava para representar las variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.	La funcionalidad de los cromosomas en el componente AGJava permite representar las variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión..		Población AGJava	Nro. de cromosomas de la población
				Tipo de variable AGJava	Nro. de tipos de variables
¿En qué medida la población del componente AGJava permite trabajar con más de una solución en una generación para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?	Desarrollar la funcionalidad de la población del componente AGJava para trabajar con más de una solución en una generación en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.	La funcionalidad de la población del componente AGJava permite trabajar con más de una solución en una generación en la solución óptima con algoritmos ge-néticos en problemas de Diseño de Resortes Helicoidales de Compresión.	Variable Dependiente: Solución óptima de problema de Diseño de Resortes Helicoidales de Compresión	Volumen óptimo del resorte helicoidal de compresión.	Ratio de Número de problemas de diseño de resortes helicoidales de compresión resueltos por algoritmos genéticos con relación al método tradicional.
¿En qué medida los operadores genéticos del componente AGJava permiten crear nuevas soluciones para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?	Desarrollar la funcionalidad de los operadores genéticos del componente AGJava para crear nuevas soluciones en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.	La funcionalidad de los operadores genéticos del componente AGJava permite crear nuevas soluciones en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.			
¿ En qué medida las vistas del componente AGJava permiten representar diferentes tipos de variables para la solución óptima con algoritmos genéticos de problemas de Diseño de Resortes Helicoidales de Compresión?	Desarrollar la funcionalidad de las vistas del componente AGJava para representar diferentes tipos de variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.	La funcionalidad de las vistas del componente AGJava permite representar diferentes tipos de variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.			

1.6. JUSTIFICACIÓN Y DELIMITACIÓN DE LA INVESTIGACIÓN

La importancia del poder de la evolución ya es conocida, y así los algoritmos genéticos se utilizan para abordar una amplia variedad de problemas en un conjunto de campos sumamente diversos, demostrando claramente su capacidad y su potencial.

Los algoritmos genéticos son algoritmos evolutivos que usan el principio de la selección natural para evolucionar un conjunto de soluciones hacia una solución óptima. Los algoritmos genéticos no solo son bastante potentes, sino que son fáciles de usar si la mayoría del trabajo se encapsula en componentes simples, y los usuarios solo tendrían que definir una función de "capacidad" o "aptitud" para determinar qué tan "apto" es una solución particular en relación a otras soluciones.

Los algoritmos genéticos han demostrado ser un método de solución exitoso para resolver problemas. Se han aplicado los algoritmos genéticos a una variedad de campos para encontrar soluciones a problemas muy complejos, con mejores resultados inclusive al de los expertos humanos, a veces dejando perplejos a los programadores que escribieron el algoritmo.

Esta tesis brindará un conjunto de componentes bien elaborados y documentados para aquellos que quieran aplicar los principios evolutivos de los algoritmos genéticos para la solución de problemas de optimización con espacios de soluciones ilimitados.

1.6.1. IMPORTANCIA DEL TEMA

La importancia del poder de la evolución ya es conocida, y así los algoritmos genéticos se utilizan para abordar una amplia variedad de problemas en un conjunto de campos sumamente diversos, demostrando claramente su capacidad y su potencial.

Los algoritmos genéticos son algoritmos evolutivos que usan el principio de la selección natural para evolucionar un conjunto de soluciones hacia una solución óptima. Los algoritmos genéticos no solo son bastante potentes,

sino que son fáciles de usar si la mayoría del trabajo se encapsula en componentes simples, y los usuarios solo tendrían que definir una función de "capacidad" o "aptitud" para determinar qué tan "apto" es una solución particular en relación a otras soluciones.

Los algoritmos genéticos han demostrado ser un método de solución exitoso para resolver problemas. Se han aplicado los algoritmos genéticos a una variedad de campos para encontrar soluciones a problemas muy complejos, con mejores resultados inclusive al de los expertos humanos, a veces dejando perplejos a los programadores que escribieron el algoritmo.

Esta tesis brindará un conjunto de componentes bien elaborados y documentados para aquellos que quieran aplicar los principios evolutivos de los algoritmos genéticos para la solución de problemas de optimización con espacios de soluciones ilimitados.

1.6.2. JUSTIFICACIÓN

Las investigaciones respecto a las aplicaciones que se le pueden dar los algoritmos genéticos como un método de optimización crecen aceleradamente, una de las dificultades con la que se encuentran investigadores y desarrolladores es el hecho de que el código desarrollado tiene escasa (a veces ninguna) documentación de la implementación de estos paquetes, librerías y programas, siendo esto una gran dificultad para su reutilización y extensibilidad para su uso en un proyecto serio.

Por las razones anteriormente expuestas se desarrollará AGJava (Algoritmo Genético en Java) que es un componente Java desarrollado con la Ingeniería de Software Orientada a Objetos para la aplicación de los Algoritmos Genéticos y adecuadamente documentado para su reutilización y extensibilidad, en la solución de problemas de optimización con espacios de soluciones ilimitado.

1.6.3. DELIMITACIÓN Y ALCANCE DEL TRABAJO

El componente AGJava (Algoritmo Genético en Java) a desarrollar se puede utilizar en cualquier algoritmo genético que resuelva un problema de optimización en los que la heurística representa una buena alternativa para encontrar la solución óptima.

El componente AGJava será desarrollado en el lenguaje de programación Java, por lo tanto el algoritmo genético que lo utilice debe estar desarrollado en Java, o lenguaje afín que permita el uso de estos componentes

El componente AGJava se probará en un algoritmo genético implementado para la solución óptima (de menor volumen y costo) de problemas de diseño de resortes Helicoidales de Compresión. El algoritmo genético desarrollado se aplicará a un conjunto de problemas de diseño de resortes Helicoidales que utilizará el método tradicional de solución con computadora para estos problemas, luego, el mismo conjunto de problemas utilizará como método de solución el algoritmo genético desarrollado.

CAPÍTULO II

MARCO TEÓRICO Y CONCEPTUAL

2.1. ESTADO DEL ARTE Y TRABAJOS PREVIOS

2.1.1. A NIVEL INTERNACIONAL

Durante las dos últimas décadas ha habido un creciente interés en los algoritmos que están basados en el principio de la evolución (supervivencia del más capaz). Un término común, aceptado recientemente, se refiere a tales técnicas como algoritmos evolutivos (EA) (o métodos de computación evolutiva). Los más conocidos algoritmos en esta clase incluyen los algoritmos genéticos, programación evolutiva, estrategias evolutivas, y programación genética. Existen también varios sistemas híbridos que incorporan varias características de los paradigmas anteriormente mencionados, y por lo tanto es difícil clasificarlos; de todas formas, nos referimos a ellas como métodos de computación evolutiva (EC).

El campo de la computación evolutiva ha alcanzado un estado de alguna madurez. Existen varias, conferencias bien establecidas que atraen cientos de participantes (International Conferences on Genetic Algorithms – ICGA, Parallel Problem Solving from Nature – PPSN, Annual Conferences on Evolutionary Programming – EP, IEEE International Conferences on Evolutionary Computation.) También hay muchos talleres, sesiones especiales, y conferencias locales cada año, en todo el mundo. Una revista, Evolutionary Computation (MIT Press), esta dedicada completamente a las técnicas de computación evolutiva. Otras revistas organizan temas especializados en

temas de computación evolutiva. Muchos artículos tutoriales excelentes y reportes técnicos ofrecen más o menos bibliografías completas de este campo. Tenemos también libros sobre este tema.

Con el creciente interés sobre los algoritmos evolutivos y sus aplicaciones aparecieron varios paquetes y bibliotecas de algoritmos genéticos, tales como:

- FORTRAN GA: Desarrollo de algoritmos genéticos para Fortran. Su dirección Web es http://cuaerospace.com/carroll/ga.html.
- GAA: Es un juego de herramientas de propósito general para algoritmos genéticos implementados en Java, donde el usuario puede definir y correr su propio problema de optimización. Disponible en http://www.aridolan.com/ga/gaa/gaa.html
- GAGS: Generador de aplicaciones basadas en algoritmos genéticos, escrito en C++. Desarrollado por el grupo de J.J. Melero. Excelente. Su dirección Web es http://geneura.ugr.es/GAGS/
- GAJIT: Es un conjunto simple de clases en Java que se escribieron para experimentar con los algoritmos genéticos. Este se encuentra en la dirección de Internet http://www.micropraxis.com/gajit
- Galib: Biblioteca de algoritmos genéticos de Matthew. Conjunto de clases en C++ de algoritmos genéticos. Su dirección Web es http://lancet.mit.edu/ga/, y su dirección para descargarlo vía FTP es lancet.mit.edu/pub/ga/. Podemos registrarlo en http://lancet.mit.edu/ga/dist/
- GALOPPS: Bastante bueno, es muy flexible y además nos permite con facilidad puede ser encontrado en GARAGe, esta implementado en C. Su dirección primaria en Internet es http://garage.cps.msu.edu/
- GAS: Paquete para desarrollar aplicaciones de algoritmos genéticos en Python. Su dirección Web es http://www.python.net/crew/gandalf/,

- GECO: Conjunto de herramientas para Lisp. Su dirección para descargarlo es http://common-lisp.net/project/geco/
- GPdata: Para desarrollar algoritmos genéticos en C++. Su dirección para descargarlo vía FTP es ftp://ftp.cs.bham.ac.uk/pub/authors/W.B.Langdon/gp-code/, y su documentación -GPdata-icga-95.ps- la podemos encontrar en el site de Internet cs.ucl.ac.uk/genetic/papers/
- JAGA: es un API extensible para implementar aplicaciones de algoritmos genéticos y de programación genética en Java.
- JGAP: Es un conjunto de clases en Java que pueden usarse para aplicar los principios evolutivos y la programación genética. Puede encontrarlo en http://jgap.sourceforge.net/

La dificultades principal de este código es la escasa (a veces ninguna) documentación de la implementación de estos paquetes y librerías, siendo esto una gran dificultad para su reutilización y extensibilidad, para su uso en un proyecto serio.

2.1.2. A NIVEL NACIONAL

En el ámbito nacional entre otros trabajos aplicativos e investigaciones podemos encontrar:

- "Expansión del sistema de transmisión eléctrico nacional utilizando algoritmos genéticos", XVII CONIMERA (2007), este trabajo presenta una metodología basada en un algoritmo genético y un flujo de potencia lineal para resolver el problema del planeamiento estático de la expansión del sistema de transmisión de largo plazo. La principal diferencia de la presente propuesta con trabajos similares reportados anteriormente, radica en la utilización de un flujo de potencia lineal y no un flujo de potencia optimo lineal dentro del cálculo de la aptitud de cada uno de los individuos dentro del algoritmo genético. La metodología propuesta fue implementada en un prototipo computacional utilizando el lenguaje de programación

C++. Esta implementación fue aplicada a un sistema práctico sumamente utilizado en la literatura especializada (Garver – 6 barras) y al Sistema Interconectado Nacional (SINAC – 118 barras)considerando una demanda futura de 10 años, presentando resultados satisfactorios.

- "Calibración del Modelo Lluvia-Escorrentía Gr4j usando Algoritmos Genéticos. Caso: Cuenca Del Rio Chili", XVII International Congress of Electronic, Electrical and Systems Engineering IEEE, 2010, Arequipa. Aquí se presenta la calibración del modelo lluvia-escorrentía GR4J para las subcuencas El Pañe y El Frayle de la cuenca del Chili Arequipa. Esta calibración usa 6 parámetros, El proceso consta de 5 etapas: Preparación de datos, calibración del modelo que incluye la selección de los eventos, la definición de la función de error, y estimación de parámetros los cuales son optimizados mediante el uso de Algoritmos Genéticos combinado con el método del gradiente. La calidad del ajuste se verifica comparando la serie de caudales generada por el modelo y una sección del registro histórico, además se usa las curvas de duración de caudales, y 4 criterios de eficiencia. Los resultados simulados muestran un ajuste significativo a los datos observados lo que permite establecer que el modelo encontrado representa matemáticamente los fenómenos hidrológicos de la cuenca.

- "Identificación y validación del modelo no lineal de la temperatura del aceite superior de transformadores de potencia aplicando algoritmos genéticos", el trabajo fue presentado en el IV Congreso Internacional de la IEEE ANDESCON 2008, realizado en la Ciudad de Cusco. Este trabajo presenta una técnica basada en Algoritmos Genéticos para la identificación y validación del modelo no lineal de la temperatura del aceite superior en los transformadores de potencia que está siendo utilizado en el sistema de monitoreo y diagnostico en línea instalado en un transformador de 100 MVA 230/115/24 kV OA/FA/FOA de la Subestación Barquisimeto de ENELBAR Venezuela desde el año 2003. Los resultados de la identificación por algoritmos genéticos se comparan con resultados obte-

nidos con identificación por mínimos cuadrados lineales y de las mediciones reales de la temperatura del aceite superior. Estos resultados además evidencian una reducción significativa del error en el modelo al realizar la identificación por algoritmos genéticos, lo cual mejora su desempeño como herramienta de diagnóstico para transformadores de potencia.

- "Mix del Producto Óptimo Usando Algoritmos Genéticos". XII Encuentro Científico Internacional de verano, 2005, Lima. Mix de producto óptimo significa determinar la cantidad de productos a producir para maximizar la ganancia. Para de-terminar el mix de producto óptimo de la Cooperativa Industrial Manufacturas del Centro de la ciudad de Huancayo, la empresa textil más importante de la región Andrés A. Cáceres, se ha construido el modelo híbrido que combina la simulación de eventos discretos y con los algoritmos genéticos. La simulación de eventos discretos se utiliza para inferir el costo unitario indirecto de cada producto debido al empleo de un sistema de costos basado en actividades. Para aplicar un sistema de costos basado en actividades se requiere información a posteriori, pero se puede conocerlo (aproximarlo) a priori aplicando la simulación de eventos discretos. Los algoritmos genéticos determinan el mix del producto óptimo que maximiza la utilidad. Estos algoritmos genéticos utilizan la codificación de valor para los cromosomas e incluyen técnicas para la solución de problemas con restricciones lineales. El mix de producto óptimo obtenido con el modelo disminuye las pérdidas con respecto al mix utilizado en el primer semestre, de aquellos productos cuyo costo unitario es superior a su precio, en un 43% e incrementan la utilidad en 123%

- "Reduciendo el Ancho de Banda de Matrices Dispersas Simétricas con Algoritmos Genéticos", UNMSM, 2009. El presente trabajo propone la reducción del ancho de banda en matrices dispersas y simétricas, usando la metaheurística Algoritmos Genéticos y un software desarrollado en MS Visual Studio 6.0.

- "Programa para la Síntesis Automática de un Ota Miller", Grupo de Microelectrónica de la Pontificia Universidad Católica del Perú (PUCP). El trabajo muestra el desarrollo y resultados de pruebas de un programa en MATLAB para la síntesis automática de un OTA tipo Miller en tecnología AMS 0.35u. El programa calcula las dimensiones de los transistores y las corrientes de polarización a partir de especificaciones de consumo de potencia, ganancia DC, producto ganancia por ancho de banda, Slew Rate, Margen de Fase y capacitancia de carga. En lugar de utilizar ecuaciones para los cálculos del punto de polarización, el programa toma en cuenta curvas características de transistores obtenidas con el simulador Spectre y el modelo BSIM3. En el cálculo de dimensiones geométricas se considera su influencia sobre la capacitancia parásita de salida.

Entre las tesis tenemos:

- Planificación de Horarios del Personal de Cirugía de un Hospital del Estado Aplicando Algoritmos Genéticos (Time Tabling Problem). Tesis para optar por el Título de Ingeniero Informático, 2009. Este proyecto intenta dar solución al problema de generación de horarios del personal de un hospital, para ser más exactos del servicio de Cirugía y Radioterapia un hospital del estado. La solución se construye con el uso de un algoritmo genético. Para facilitar la búsqueda de esta solución se aplicará los operadores de cruzamiento y mutación especialmente pensados para la estructura del cromosoma o individuo.
- "Análisis Comparativo entre el Algoritmo Cuántico de Grover y un Algoritmo Grasp, Aplicados a la Búsqueda de Individuos Óptimos en la Población Inicial de un Algoritmo Genético". Tesis para optar por el Título de Ingeniero Informático, 2010. Este trabajo trata sobre la aplicación de dos algoritmos de búsqueda a la selección de individuos óptimos en la población inicial de un algoritmo genético, y la consiguiente comparación entre ambos. El primero de ellos es el algoritmo meta-heurístico GRASP, y el segundo es el algoritmo cuántico de Grover. El algoritmo cuántico de Grover forma parte de una nueva generación en las ciencias de la

computación: la computación cuántica. Y por tanto hace uso de conceptos matemáticos y físicos completamente distintos a los usados en la programación clásica. En esta tesis se presenta un análisis general de ambos algoritmos, siendo de especial mención el análisis del algoritmo cuántico de Grover, ya que incluye un modelo matemático del funcionamiento del mismo. Este modelo será de suma importancia para simular la ejecución del algoritmo de Grover en una computadora clásica, dada la carencia de una computadora cuántica sobre la cual realizar esto. Luego, se preparan dos procesos experimentales, los cuales se usarán para realizar la comparación de eficacia y eficiencia entre las ejecuciones de los algoritmos. Posteriormente, se presenta el diseño e implementación de los algoritmos, ambos aplicados a la selección de individuos de un algoritmo genético genérico.

- "Sistema Informático basado en Algoritmos Evolutivos para mejorar el Proceso de Identificación Forense de Evidencias Digitales. Tesis para optar por el Título de Ingeniero Informático, 2010.
- "Un Algoritmo genético para optimizar la asignación de aulas y docentes en la generación de programas académicos en centros de idiomas". Tesis para optar el Grado de Ingeniería de Sistemas e Informática, 2009.
- "Optimización de celosias bidimensionales mediante algoritmos genéticos". Tesis de Maestría en Ciencias con mención en Ingeniería de Sistemas, 2011. En este trabajo se propone una nueva técnica basada en la codificación de instrucciones de ensamble en un cromosoma, la misma que además considera una zona intrónica (no representativa) que permite el intercambio genético de celosías de diferentes tamaños y complejidad con lo que se ha reducido dramáticamente la tasa de inviabilidad luego de las operaciones genéticas. El método es reforzado mediante el uso de un método matricial de cálculo de estructuras en la función de evaluación en lugar del MEF, lo cual permite liberarnos de la necesidad de la malla.

2.2. MARCO TEÓRICO

2.2.1. CONCEPTOS DE OBJETOS

Objeto

Conceptualmente, un objeto es una cosa con la que se puede interactuar: se le puede enviar varios mensajes y éste reacciona ante ellos.

Mensajes

Un mensaje incluye una palabra clave llamada selector. Un mensaje puede, pero no es necesario, incluir uno o más argumentos, cuyos valores se le pasan al objeto tal y como se pasan los valores a una función en una llamada normal. Normalmente, para un selector determinado hay un número "correcto" de argumentos que deben pasarse en un mensaje que empieza por ese selector; los valores aceptables de los argumentos se establecen en la interfaz del objeto.

Interfaces

La interfaz pública de un objeto define qué mensajes se aceptan sin importar de dónde vienen. Normalmente la interfaz almacena los selectores de estos mensajes junto con alguna información sobre qué argumentos se requieren y si se devuelve algo o no. Probablemente se preferiría que la interfaz incluyese algún tipo de especificación sobre las consecuencias de enviar un mensaje a un objeto, pero estas especificaciones normalmente sólo vienen en comentarios y documentación acompañante. De manera que un objeto normalmente tiene al menos dos interfaces: la interfaz pública, que cualquier parte del sistema puede utilizar, y la interfaz privada que la pueden utilizar el propio objeto y algunas partes privilegiadas del sistema.

Clases

Hasta aquí se ha hablado de objetos como si cada uno estuviese definido por separado, con su propia interfaz, su propia manera de controlar que otros objetos podrían enviarle y que mensajes. Por supues-

to, ésta no es la manera más coherente de construir un sistema típico, porque los objetos tienen mucho en común unos con otros. Una clase describe un conjunto de objetos que tiene un rol o roles equivalente en el sistema. En los lenguajes orientados a objetos basados en las clases, cada objeto pertenece a una clase, y la clase de un objeto es quien determina su interfaz. En realidad, la clase del objeto, junto con los valores de los atributos no están determinados por la clase, pudiendo variar. Por ejemplo, a lo mejor los objetos que pertenecen a la clase tienen un atributo privado. Si se envía el mismo mensaje a dos objetos que pertenecen a una misma clase, se ejecuta el mismo método en ambos casos, aunque el efecto de la ejecución del método puede ser muy diferente si los objetos tienen distintos valores en sus atributos. En resumen, los objetos de una misma clase tienen las mismas interfaces.

Al proceso de creación de un nuevo objeto que pertenece a una clase se le llama instanciación de la clase, y al objeto resultante se le denomina instancia de la clase; por esto, por supuesto, es por lo que las variables cuyos valores pertenecen al objeto y pueden variar a lo largo de su tiempo de vida se denominan variables instancia.

Objetos y componentes

La exageración que rodea la orientación a objetos a veces sugiere que cualquier clase es automáticamente un componente reutilizable. Esto, por supuesto, no es verdad, la reutilización de un componente no es simplemente un hecho del componente en sí, si no que depende del contexto de desarrollo y de la reutilización propuesta. Otro factor importante es que la estructura de una clase a menudo resulta estar demasiado detallada para la reutilización efectiva. Por ejemplo, para reutilizar una única clase de manera efectiva y rápida se tiene que escribir en el mismo lenguaje de programación y utilizar una arquitectura compatible. Esto no es siempre impracticable; existen bibliotecas de clases muy utilizadas con éxitos y sus clases pueden ser conside-

radas sensatamente como componentes reutilizables. A menudo, sin embargo, es más apropiado un componente formado por un grupo de clases relacionadas. Cuanto mayor es el esfuerzo para conectar un componente en un contexto, mayor es el beneficio que tiene que proporcionar el componente antes de que el esfuerzo merezca la pena. El componente en sí pretende ser una abstracción encapsulada, y probablemente es la manera apropiada de ocultar parte o toda su estructura interna; por ejemplo, si está compuesto por clases u objetos y cómo.

Herencia

En esta parte se considerará la herencia como una característica técnica de los lenguajes orientados a objetos, útil (de una forma limitada) para la reutilización de bajo nivel. Los lenguajes de programación orientados a objetos permiten definir una nueva clase en función de la antigua clase. Simplemente se especifica que la nueva clase es una subclase de otra clase. La nueva clase puede incluir un nuevo método que implemente la misma operación para la que la otra clase ya tiene un método. Cuando un objeto de la nueva clase recibe un mensaje solicitándole que ejecute la operación, el método invocado será la versión especializada proporcionada por la nueva clase. Una subclase es una versión extendida y especializada de su superclase. Incluye las operaciones y los atributos de la superclase, y posiblemente algunos más. Se dice que:

- La nueva clase hereda de la otra clase.
- La nueva clase es una subclase (o clase derivada) de la otra clase.
- La nueva clase es una especialización de la otra clase.
- La nueva clase está más especializada que la otra clase.
- La otra clase es una superclase (o clase base) de la nueva clase

- La otra clase es una generalización de la nueva clase.

Todo esto significa casi lo mismo. A menudo la gente utiliza subclase/superclase como una descripción de una relación concreta entre clases, y especificación/generalización como una descripción de la relación entre los conceptos representados por las clases. Normalmente ambos coinciden. Si bien, la relación conceptual no necesita mostrarse en la estructura de la clase. A la inversa, a veces la gente utiliza la herencia en una estructura de clase en casos donde la relación conceptual de generalización entre clases no existe. (No se recomienda esto: puede ser cómodo a corto plazo pero casi siempre se paga el desorden a largo plazo.)

Polimorfismo y ligadura dinámica

Polimorfismo: Este término, derivado del griego, significa tener varias formas. En los lenguajes de programación, se refiere a la situación en la que una entidad podría tener uno de entre varios tipos. En una situación orientada a objetos, una variable polimórfica podría hacer referencia (en diferentes momentos) a objetos de varias clases distintas. Una función polimórfica puede tomar argumentos de diferentes tipos. Ahora bien, se ha dicho que se supone que un objeto de una subclase se puede utilizar en cualquier sitio donde se puede utilizar un objeto de la superclase. En particular, esto querría decir que un objeto subclase debería aceptarse como valor de un variable, si lo es un objeto de la clase base. En otras palabras, en un lenguaje orientado a objetos cualquier variable es polimórfica, ¡al menos en este sentido limitado! De forma similar, si alguna función (por ejemplo, un método) puede obtener un argumento de la clase B debería poder también obtener un argumento de cualquier clase C que es una subclase de B; de manera que cualquier función es polimórfica en este sentido limitado. De esta manera el polimorfismo libera una gran cantidad de duplicación de código. Sin embargo, realmente se asocia/hereda consigo mismo cuando se combina con la ligadura dinámica.

Ligadura dinámica: El término "ligadura" en ligadura dinámica (o ligadura tardía, ambos se utilizan indistintamente) hace referencia al proceso de identificar el código que debería ejecutarse como respuesta a algún mensaje. Esto quiere decir, que la misma sintaxis por ejemplo debería producir dos partes de código que se ejecutarían en dos momentos diferentes. El envío del mensaje se enlaza dinámicamente al código apropiado.

2.2.2. INGENIERÍA DEL SOFTWARE CON COMPONENTES

Esta parte está centrada principalmente en la ingeniería del software orientada a objetos de manera que se piensa principalmente en los tipos de sistemas que se construyen con la tecnología orientada a objetos.

Un buen sistema

Fundamentalmente, un buen sistema (o de alta calidad) es aquel que cumple las necesidades del usuario. Es decir, tiene que ser:

- Útil y aprovechable: un buen software hace la vida de los usuarios más fácil o mejor.
- Fiable: un buen software tiene pocos errores.
- Flexibles: las necesidades de los usuarios cambian a lo largo del tiempo, incluso mientras el software se está desarrollando, de manera que es importante poder realizar cambios en el mismo más tarde. ¡Además tiene que ser posible no cometer más errores! Todos los cambios que se hacen en el software después de ser entregado, tradicionalmente se llama mantenimiento.
- Accesible: tanto para la compra como para el mantenimiento. Los costes de mano de obra son el elemento más significativo dentro de los costes del software, de manera que si este se reduce quiere decir relativamente sencillo y fácil de desarrollar y de mantener.

- Disponible: De otro modo, ¡no importa lo bueno que sea! Se considera dos aspectos en cuanto a disponibilidad
- el software tiene que poder ejecutarse en hardware disponible, con el sistema operativo disponible, etc. Esto implica, por ejemplo, que un sistema que debe ser suficientemente portable, y también nos lleva de nuevo a la facilidad de mantenimiento, ya que debe poderse realizar cualquier cambio provocado por cambios en el entorno del software.
- ¡El software debe ser lo primero que exista! De forma que un proyecto de software debe completarse con éxito, y entregar el software prometido.

A lo largo de las últimas décadas se ha alcanzado un conocimiento cada vez más profundo de qué sistemas son los más propensos a ser correctos. Todavía queda mucho por aprender, el problema fundamental es que hay un límite de cuánto puede entender, de una sola vez, un humano.

Los sistemas pequeños pueden construirse mediante "programación heroica" donde una única persona intenta tener en mente todo los aspectos relevantes del sistema; pero en general es imposible, para un desarrollador o encargado de mantenimiento comprender todo lo relacionado con el sistema de una sola vez. Esto quiere decir que es fundamental poder emprender una tarea de desarrollo o mantenimiento sin entender todo el sistema.

El siguiente paso es pensar en un sistema como conjunto de módulos e identificar dependencias entre módulos. En el más amplio sentido de la palabra, un módulo podría ser cualquier "elemento" identificable del sistema y que tiene sentido por sí mismo. Por ejemplo, serían módulos:

- Archivos (ficheros).

- Subrutinas.
- Funciones de biblioteca.
- Clases, en un lenguaje orientado a objetos.
- Otras estructuras conocidas como módulo o similar.
- Programas o subsistemas independientes o semi-independientes.

Por su puesto no todos los módulos son iguales; tomar un programa monolítico y separarlo de forma aleatoria en archivos no es óptimo. Se debe saber qué módulos característicos se debería tener para llevar a cabo el desarrollo y el mantenimiento del sistema de la manera más sencilla, barata y fiable posible. Se debe tener en cuenta los conceptos asociados de dependencia, acoplamiento, cohesión, interfaz, encapsulación y abstracción. (Los distintos autores utilizan definiciones ligeramente diferentes de estos términos, de manera que las que se utilizan aquí no son las únicas posibles.) Una vez identificado lo que es un buen módulo, se puede considerar la reutilización de un buen módulo como componente.

El Módulo A depende del módulo B si cualquier cambio en el Módulo B implica que el Modulo A también tenga que ser modificado.

A veces se dice que el Módulo A es un cliente del Modulo B, o que el módulo B actúa como servidor del Módulo A (no en el sentido que se le da a la "arquitectura cliente – servidor"). En general es normal que un mismo módulo sea tanto cliente y servidor. Esto significa, que depende de algunos módulos, mientras que otros módulos dependen de él. Incluso es posible que un par de módulos se tengan el uno al otro de cliente; sin embargo, éste es un ejemplo de dependencia circular, que debe evitarse cuando sea posible debido a que impide la reutilización.

La dependencia a veces se conoce como acoplamiento. Un sistema con muchas dependencias tiene fuerte acoplamiento. Los buenos sis-

temas tienen débil acoplamiento, porque en ese caso los cambios en una parte del sistema son menos probables de propagarse a través del sistema.

Encapsulación: débil acoplamiento

Entre otras cosas se pretende minimizar el número de casos en los que un cambio en un módulo necesite un cambio en otro módulo. Esto significa que hay que saber qué cambios dentro de un módulo pueden afectar al resto del sistema. Sin embargo, para poder aprovecharse del débil acoplamiento de un sistema es muy importante poder identificar qué módulos están acoplados; por otra parte puede que haya que gastar esfuerzo comprobando si se necesitan los cambios en un módulo, lo cual es caro incluso si la conclusión es que no se necesitan dichos cambios. Lo ideal es saber con certeza qué módulo de un sistema podría verse afectado por un cambio en un módulo dado.

En resumen, una vez que se han establecido los límites entre los módulos del sistema hay dos tipos de información que puede ser útil.

1. ¿Qué suposiciones pueden hacer los clientes de un determinado módulo sobre él? Por ejemplo, ¿qué servicios se supone que se van a proporcionar? Responder a esto nos permite saber qué tipo de cambios en un módulo pueden ser peligrosos.
2. ¿Qué módulos son clientes de uno dado? Contestar a estos nos dice qué módulos se tienen que cambiar, si se realizan un cambio peligroso en un módulo.

Se dirigirán estas preguntas de una en una.

Interfaces

Una interfaz de un módulo, define algunas características de dicho módulo de las que sus clientes pueden depender. El resto del sistema sólo puede utilizar el módulo de las maneras permitidas por la(s) interfaz(es); esto es, una interfaz encapsula conocimientos sobre el módu-

lo. Parnas escribe "Las conexiones entre módulos son las suposiciones que los módulos hacen unos de otros". Cualquier suposición que un cliente hace sobre un servidor corre el riesgo de se incorrecta; por lo que se debería documentar tales suposiciones en la interfaz con éxito y se podrá decir:

"Si un módulo cambia internamente sin cambiar su interfaz dicho cambio no necesitará que se realice ningún otro cambio en ninguna otra parte del sistema".

Hasta aquí una interfaz puede ser, por ejemplo, una sección de comentarios al inicio de un fichero el cual, por convención, todo aquél que cambie o utilice el archivo lo mantiene al día. El problema con esto es que no hay manera de garantizar que todo el mundo cumpla con las convenciones. Perfectamente, podría haber comprobaciones automáticas de que ningún otro módulo hace ninguna suposición sobre este módulo que no esté documentada en su interfaz, y también que el módulo siempre justifica las suposiciones que se encuentran documentadas.

Un lenguaje de programación, cuanto más permita la automatización de las comprobaciones se dice que soporta más la modularidad y la encapsulación.

La información que se puede almacenar en una interfaz y que se puede comprobar automáticamente depende, por supuesto, del lenguaje de programación. Normalmente habrá algunas funciones que podrán ser invocadas, posiblemente algunos nombres de valores de datos u objetos que pueden utilizarse; algunos lenguajes como el Java permiten que se nombren en una interfaz tipos o excepciones. Los módulos clientes no permitirán aludir a nada definido en el módulo servidor que no esté en la interfaz. En muchos lenguajes se harán comprobaciones de cómo los módulos cliente utilizan los nombre de los elementos en la interfaz, por ejemplo, comprobar sus suposiciones sobre qué

tipos tienen los elementos. No utilizar un nombre en una interfaz o utilizarlo suponiendo que tiene un tipo diferente al documentado provoca un error, en tiempos de compilación, de enlazado o de ejecución. Esto está relacionado con los aspectos sintácticos de dependencia.

Es más: lo ideal sería poder comprobar la dependencia semántica. Esto es, que si una interfaz documenta fielmente las suposiciones que se pueden hacer sobre los módulos, sería una verdadera especificación del módulo, que explica lo que los clientes pueden asumir sobre el comportamiento del mismo, no sólo la sintaxis de cómo interactuar con él. Desafortunadamente, hoy por hoy, los lenguajes de programación no proporcionan esta información (con unas pocas excepciones). La principal razón de este hecho, es que la informática teórica no ha avanzado lo suficiente para proporcionar estas características. Esta es un área muy activa de investigación.

Dependencias de contexto

Existen varias razones para querer saber no solo las dependencias que podrían existir, esto es, que características están documentadas en las interfaces de los módulos del sistema, sino qué dependencias existen realmente. Se ha mencionado la situación en la que se realiza un cambio a un módulo que podría afectar a sus clientes; sus clientes son los módulos que podrían cambiar, por lo que es importante decir cuáles son. Después supóngase que se considera reutilizar un módulo. Se necesita saber no sólo los servicios que van a proporcionar (cuál es su interfaz) sino que servicios requiere para funcionar. Los servicios que requiere un módulo a veces se denominan dependencias de su contexto. Se pueden expresar en función de interfaces, el módulo podría garantizar que si tiene interfaces seguras, entonces por turnos, se proporcionará su propia interfaz.

Entre ellas, las dependencias de contexto de un módulo y la interfaz propia del módulo constituyen un contrato que describe las responsa-

bilidades del módulo. Si el contexto proporciona lo que el módulo necesita, entonces éste garantiza proporcionar los servicios descritos en su interfaz.

Beneficios de la modularidad con interfaces definidas

Incluso una interfaz escasa para un módulo mal elegido, puede hacer que el sistema se comprenda y se modifique más fácilmente. ¿Por qué es esto? La razón fundamental es que cualquier cosa que permita reducir lo que se necesita saber del mismo, es beneficiosa de varias formas diferentes.

- En un desarrollo en equipo, los desarrolladores de código que utilizan un módulo sólo deberían comprender la interfaz del módulo, no cómo funciona, de manera que puedan ser más productivos.
- Debido a que los desarrolladores pueden ignorar de forma segura algunos aspectos del sistema, tiende a conocer más a fondo los aspectos que necesitan de verdad, por lo que se introducen menos errores.
- Los errores deberían ser más fáciles de encontrar (tanto en el desarrollo como en el mantenimiento), debido a que sería posible evitar examinar módulos irrelevantes.
- Una vez que existe un módulo, con documentación de lo que proporciona y de lo que requiere, es posible al menos considerar su reutilización.

El verdadero desafío, sin embargo, es definir buenos módulos con los elementos correctos en sus interfaces. Sólo en este casos pueden alcanzar todos los beneficios.

Un módulo puede tener varias interfaces

Si se sabe que el Módulo A puede verse afectado a veces por cambios del Módulo B, se quiere también identificar de la manera más sencilla posible los cambios requeridos, si hay alguno en nuestro caso

particular. A veces es conveniente documentar los servicios que proporcionan un módulo a través de varias interfaces diferentes, de forma que se pueda ser más preciso sobre qué servicios necesita un determinado cliente. Una vez más, es útil tanto para el mantenimiento como para la reutilización. Un buen sistema esta formado por módulos encapsulados.

Abstracción: fuerte cohesión

Los módulos correctos a menudo tienen la propiedad de que sus interfaces proporciona una abstracción de algún elemento conocido de manera intuitiva que puede, obstante, ser difícil de implementar. Este tipo de módulos se dice que tienen una fuerte cohesión.

La interfaz se abstrae de las cosas que el desarrollador no tiene que conocer para utilizar el módulo, dejando una fotografía ordenada y coherente de lo que el usuario del módulo quiere conocer. El módulo realiza un conjunto coherentes de cosas, pero dentro de lo posible el desarrollador del cliente está protegido de la información irrelevante relativa a cómo el módulo hace lo que hace. Este asunto de permitir al desarrollador concentrarse en lo esencial es sutilmente diferente del asunto de encapsulación para alcanzar el débil acoplamiento, que está implicando en la prevención del desarrollador de utilizar información oculta.

Abstracción es cuando un cliente de un módulo no necesita saber más de lo que hay en la interfaz.

Encapsulación es cuando un cliente de un módulo no es capaz de saber más de lo que hay en la interfaz.

La situación en la que la interfaz proporciona medios para interactuar con algunos datos, pero revela nada sobre el formato interno de los mismos, es normal tanto en el desarrollo orientado a objetos como en el estilo de tipo de datos abstractos. Algunos autores restringen el uso del término encapsulación a este tipo de módulo encapsulado.

Si un módulo, de cualquier tamaño y complejidad, es una buena abstracción (tiene fuerte cohesión y débil acoplamiento) puede ser factible reutilizarlo en sistemas posteriores, o sustituirlo en el sistema existente. Esto es, puede ser posible considerarlo como un componente conectable. Sin embargo, aunque esto es posible, también depende de la arquitectura en la que se desarrolla el componente y en la que vaya a ser utilizada. Esto lleva a pensar en el desarrollo basado en componentes centrado en la arquitectura (CBD). Los significados de estos términos, que son palabras de moda, más recientes que aquellas a las que se ha hecho alusión hasta este momento ¡son incluso más discutibles!.

Arquitectura y componentes

La mejor definición de la palabra componente que se puede encontrar es "cosa que se puede reutilizar o sustituir"; la visión escéptica es que el Desarrollo Basado en Componentes (CBD) alcanza por definición altos niveles de reutilización, ya que si no lo hace, ¡no es realmente CBD!.

Por ahora se considera un componente como una unidad de reutilización y sustitución. Un componente pues ser un módulo con propiedades que lo hacen reutilizable y sustituible. Mucha gente considera la composición tardía como una característica importante del desarrollo basado en componentes: el asunto es que es más fácil reemplazar un módulo si se puede hacer sin necesidad de recompilar el sistema. Sin embargo, hay diferentes opiniones, especialmente sobre qué es tarde, y se prefiere no imponer esta restricción. (Desafortunadamente, UML también utiliza el término "componente" como un concepto relacionado, pero diferente.)

¿Qué hace que un módulo sea reutilizable? No debería sorprender que un módulo reutilizable sea bueno de acuerdo con lo discutido anteriormente; tiene fuerte cohesión, débil acoplamiento con el resto del

sistema, una interfaz bien definida, y es una abstracción encapsulada de un elemento bien comprendido del contexto en el que sea desarrollado y en el que se vaya a utilizar.

Para ver la importancia del contexto técnico, considérese una organización que produce un conjunto de sistemas que tiene mucho en común; a menudo esto se conoce como desarrollo de producto en línea, por razones obvias. Si la estructura de alto nivel de dos sistemas consecutivos es la misma, es bastante más probable poder utilizar un módulo desarrollado para un sistema y conectarlo al otro sin alterarlo (reutilización de caja negra) que si los dos sistemas tiene distinta arquitectura. La arquitectura del sistema incluye decisiones para manejar siempre errores inesperados de una cierta manera es una decisión de arquitectura; un ejemplo de alto nivel es una decisión para utilizar una base de datos de un fabricante determinado o un cierto lenguaje de programación. Parte de la motivación que lleva a tomar estas decisiones es la misma que la lleva a identificar módulos apropiados: reducir la carga que tiene desarrollador o el encargado de mantenimiento. Si ciertas cosas están hechas de forma fiable y de la misma manera en todas las partes de un sistema, y mejor todavía de sistema a sistema, el desarrollador o encargado de mantenimiento puede esperar que las cosas sean de esta manera y se perdona el problema de resolver cómo son, en esta instancia concreta. Un componente desarrollado en el contexto de arquitectura es más probable que se reutilice en un nuevo contexto si ambos comparten las decisiones de arquitectura. (En este caso es la arquitectura en si lo que se reutiliza; pero también aumenta la posibilidad de reutilizar componentes. Así, cualquier interfaz es mejor que ninguna, y una decisión de arquitectura no óptima puede ser mejor que no tomar ninguna. Solo con interfaces, el verdadero reto es identificar las decisiones de arquitectura correctas.

Desarrollo basado en componentes: conectividad

La manera ideal de construir un nuevo sistema es tomar algunos componentes existentes y conectarlos unos con otros. Por supuesto, la conectividad es la propiedad que permite hacer esto. La metáfora sugiere correctamente que esto es sólo en parte una propiedad de los elementos conectados. Los componentes tienen que ser compatibles unos con otros, y esto depende de la presencia de una arquitectura adecuada.

Las decisiones de arquitectura: se tiene que tomar pronto en el proyecto, se ven afectadas por la naturaleza de los componentes en la arquitectura y pueden estar influenciadas por el entorno del proyecto.

En resumen, el desarrollo está centrado en la arquitectura y basado en componentes si se da máxima prioridad al seguimiento (y toma) de decisiones de arquitectura correctas, y a la utilización (y desarrollo) de componentes correctos.

La orientación a objetos, como se verá, es un paradigma adecuado (pero no el único) en el que hacer CBD centrado en la arquitectura.

Construcción de un buen sistema

Establezcamos ahora los antecedentes considerando el término ingeniería del software, cuya adopción sugiere que los sistemas podrían ser construidos por analogía con artefactos de ingeniería, por ejemplo motores o puentes. Una aproximación técnica:

- Utiliza un proceso definido con fases claras, cada una de las cuales tiene un producto final (quizá un documento, quizá algo construido).
- Está relacionado con encontrar un claro conjunto de requisitos, cuidadosamente definido tan pronto como sea posible.
- Considera los formularios de verificación y validación, tales como pruebas, como algo esencial para la construcción del producto en sí.

- Utiliza un almacenamiento de conocimientos, arquitecturas y componentes relevantes.
- Hace un uso coherente de las herramientas.

Se han identificado estos aspectos ya que los sistemas de software utilizan una arquitectura modular reutilizando componentes, tal y como hacen los artefactos de ingeniería.

Una metodología que promete al respecto es el Proceso Unificado de Rational que se empleará para el desarrollo de este proyecto.

2.2.3. PROCESO UNIFICADO DE RATIONAL (RUP)

RUP se repite a lo largo de una serie de ciclos que constituyen la vida de un producto. Cada ciclo concluye con una generación del producto para los clientes. Cada ciclo consta de cuatro fases: Inicio, Elaboración, Construcción y Transición. Cada fase se subdivide a la vez en iteraciones, el número de iteraciones en cada fase es variable.

Cada fase se concluye con un hito bien definido, un punto en el tiempo en el cual se deben tomar ciertas decisiones críticas y alcanzar las metas clave antes de pasar a la siguiente fase, ese hito principal de cada fase se compone de hitos menores que podrían ser los criterios aplicables a cada iteración. Los hitos para cada una de las fases son: Inicio - Lifecycle Objectives, Elaboración - Lifecycle Architecture, Construcción - Initial Operational Capability, Transición - Product Release

La duración y esfuerzo dedicado en cada fase es variable dependiendo de las características del proyecto. La tabla ilustra los porcentajes frecuentes al respecto.

	Inicio	Elaboración	Construcción	Transición
Esfuerzo	5 %	20 %	65 %	10%

Tiempo Dedicado	10 %	30 %	50 %	10%

Inicio

Durante la fase de inicio se define el modelo del negocio y el alcance del proyecto. Se identifican todos los actores y Casos de Uso, y se diseñan los Casos de Uso más esenciales (aproximadamente el 20% del modelo completo). Se desarrolla, un plan de negocio para determinar que recursos deben ser asignados al proyecto.

Los objetivos de esta fase son:

- Establecer el ámbito del proyecto y sus límites.
- Encontrar los Casos de Uso críticos del sistema, los escenarios básicos que definen la funcionalidad.
- Mostrar al menos una arquitectura candidata para los escenarios principales.
- Estimar el coste en recursos y tiempo de todo el proyecto.
- Estimar los riesgos, las fuentes de incertidumbre.

Los resultados de la fase de inicio deben ser:

- Un documento de visión: Una visión general de los requerimientos del proyecto, características clave y restricciones principales.
- Modelo inicial de Casos de Uso (10-20% completado).
- Un glosario inicial: Terminología clave del dominio.
- El caso de negocio.
- Lista de riesgos y plan de contingencia.
- Plan del proyecto, mostrando fases e iteraciones.
- Modelo de negocio, si es necesario

- Prototipos exploratorios para probar conceptos o la arquitectura candidata.

Al terminar la fase de inicio se deben comprobar los criterios de evaluación para continuar:

- Todos los interesados en el proyecto coinciden en la definición del ámbito del sistema y las estimaciones de agenda.
- Entendimiento de los requisitos, como evidencia de la fidelidad de los Casos de Uso principales.
- Las estimaciones de tiempo, coste y riesgo son creíbles.
- Comprensión total de cualquier prototipo de la arquitectura desarrollado.
- Los gastos hasta el momento se asemejan a los planeados.

Si el proyecto no pasa estos criterios hay que plantearse abandonarlo o repensarlo profundamente.

Elaboración

El propósito de la fase de elaboración es analizar el dominio del problema, establecer los cimientos de la arquitectura, desarrollar el plan del proyecto y eliminar los mayores riesgos.

En esta fase se construye un prototipo de la arquitectura, que debe evolucionar en iteraciones sucesivas hasta convertirse en el sistema final. Este prototipo debe contener los Casos de Uso críticos identificados en la fase de inicio. También debe demostrarse que se han evitado los riesgos más graves.

Los objetivos de esta fase son:

- Definir, validar y cimentar la arquitectura.
- Completar la visión.

- Crear un plan fiable para la fase de construcción. Este plan puede evolucionar en sucesivas iteraciones. Debe incluir los costes si procede.
- Demostrar que la arquitectura propuesta soportará la visión con un coste razonable y en un tiempo razonable.

Al terminar deben obtenerse los siguientes resultados:

- Un modelo de Casos de Uso completa al menos hasta el 80%: todos los casos y actores identificados, la mayoría de los casos desarrollados.
- Requisitos adicionales que capturan los requisitos no funcionales y cualquier requisito no asociado con un Caso de Uso específico.
- Descripción de la arquitectura software.
- Un prototipo ejecutable de la arquitectura.
- Lista de riesgos y caso de negocio revisados.
- Plan de desarrollo para el proyecto.
- Un caso de desarrollo actualizado que especifica el proceso a seguir.
- Un manual de usuario preliminar (opcional).

En esta fase se debe tratar de abarcar todo el proyecto con la profundidad mínima. Sólo se profundiza en los puntos críticos de la arquitectura o riesgos importantes.

En la fase de elaboración se actualizan todos los productos de la fase de inicio.

Los criterios de evaluación de esta fase son los siguientes:

- La visión del producto es estable.
- La arquitectura es estable.

- Se ha demostrado mediante la ejecución del prototipo que los principales elementos de riesgo han sido abordados y resueltos.
- El plan para la fase de construcción es detallado y preciso. Las estimaciones son creíbles.
- Todos los interesados coinciden en que la visión actual será alcanzada si se siguen los planes actuales en el contexto de la arquitectura actual.
- Los gastos hasta ahora son aceptables, comparados con los previstos.

Si no se superan los criterios de evaluación quizá sea necesario abandonar el proyecto o replanteárselo considerablemente.

Construcción

La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones. Durante esta fase todos los componentes, características y requisitos deben ser implementados, integrados y probados en su totalidad, obteniendo una versión aceptable del producto.

Los objetivos concretos incluyen:

- Minimizar los costes de desarrollo mediante la optimización de recursos y evitando el tener que rehacer un trabajo o incluso desecharlo.
- Conseguir una calidad adecuada tan rápido como sea práctico.
- Conseguir versiones funcionales (alfa, beta, y otras versiones de prueba) tan rápido como sea práctico.

Los resultados de la fase de construcción deben ser:

- Modelos Completos (Casos de Uso, Análisis, Diseño, Despliegue e Implementación)

- Arquitectura íntegra (mantenida y mínimamente actualizada)
- Riesgos Presentados Mitigados
- Plan del Proyecto para la fase de Transición.
- Manual Inicial de Usuario (con suficiente detalle)
- Prototipo Operacional – beta
- Caso del Negocio Actualizado

Los criterios de evaluación de esta fase son los siguientes:

- El producto es estable y maduro como para ser entregado a la comunidad de usuario para ser probado.
- Todos los usuarios expertos están listos para la transición en la comunidad de usuarios.
- Son aceptables los gastos actuales versus los gastos planeados.

Transición

La finalidad de la fase de transición es poner el producto en manos de los usuarios finales, para lo que se requiere desarrollar nuevas versiones actualizadas del producto, completar la documentación, entrenar al usuario en el manejo del producto, y en general tareas relacionadas con el ajuste, configuración, instalación y facilidad de uso del producto.

Se citan algunas de las cosas que puede incluir esta fase:

- Prueba de la versión Beta para validar el nuevo sistema frente a las expectativas de los usuarios
- Funcionamiento paralelo con los sistemas legados que están siendo sustituidos por nuestro proyecto.
- Conversión de las bases de datos operacionales.
- Entrenamiento de los usuarios y técnicos de mantenimiento.

- Traspaso del producto a los equipos de marketing, distribución y venta.

Los principales objetivos de esta fase son:

- Conseguir que el usuario se valga por si mismo.
- Un producto final que cumpla los requisitos esperados, que funcione y satisfaga suficientemente al usuario.

Los resultados de la fase de transición son:

- Prototipo Operacional
- Documentos Legales
- Caso del Negocio Completo
- Línea de Base del Producto completa y corregida que incluye todos los modelos del sistema
- Descripción de la Arquitectura completa y corregida
- Las iteraciones de esta fase irán dirigidas normalmente a conseguir una nueva versión.

Los criterios de evaluación de esta fase son los siguientes:

- El usuario se encuentra satisfecho.
- Son aceptables los gastos actuales versus los gastos planificados.

2.2.4. ASPECTOS BIOLÓGICOS[1]

El Acido Desoxirribonucleico (ADN) es el material genético esencial de todos los organismos vivos. El ADN es una macro molécula doblemente trenzada que tiene una estructura helicoidal (Figura II-1). Ambos filamentos trenzados

[1] Descripción tomada de: Coello, C. A., Introducción a la Computación Evolutiva (Notas de Curso). CINVESTAV-IPN. México, 2006.

son moléculas de ácido nucleico lineales y sin ramificaciones, formadas de moléculas alternadas de azúcar y fosfato.

El ADN tiene 4 bases de nucleótido: Adenina (A), Timina (T), Citosina (C) y Guanina (G), estos son el alfabeto de información genética. Las secuencias de estas bases en la molécula de ADN determinan el plan constructor de cualquier organismo.

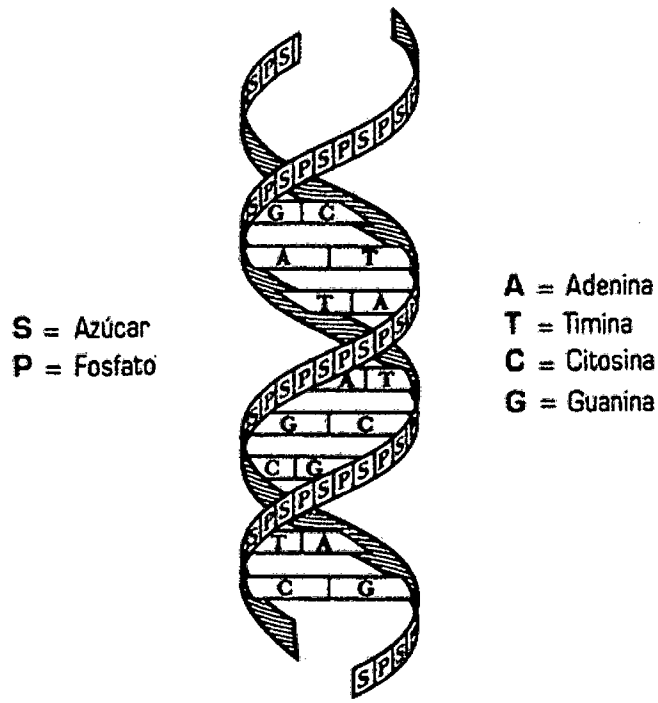


Figura II-1. Estructura del ADN

Un gene es una sección de ADN que codifica una cierta función bioquímica definida, usualmente la producción de una proteína. Es fundamentalmente una unidad de herencia. El ADN de un organismo puede contener desde una docena de genes (como en un virus), hasta decenas de miles (como en los humanos).

Se denomina cromosoma a una de las cadenas de ADN que se encuentra en el núcleo de las células. Los cromosomas son responsables de la transmisión de información genética. Cada gene es capaz de ocupar sólo una región en particular de un cromosoma (su "lugar" o "locus"). En cada deter-

minado lugar pueden existir, en la población, formas alternativas del gene. A estas formas alternativas se les llama alelos.

Se llama Genoma a la colección total de genes (y por tanto, cromosomas) que posee un organismo.

Se denomina Gametos a las células que llevan información genética de los padres con el propósito de efectuar reproducción sexual. En los animales, se denomina esperma a los gametos masculinos y óvulos a los gametos femeninos.

Se denomina individuo a un solo miembro de una población. Se denomina población a un grupo de individuos que pueden interactuar juntos, por ejemplo, para reproducirse.

Se denomina fenotipo a los rasgos (observables) específicos de un individuo. Se denomina genotipo a la composición genética de un organismo (la información contenida en el genoma), es decir, es lo que potencialmente puede llegar a ser un individuo. El genotipo se origina, tras el desarrollo fetal y posterior al fenotipo del organismo.

Durante la reproducción sexual ocurre la recombinación (o cruce). En el caso haploide, se intercambian los genes entre los cromosomas (haploides). En el caso diploide, en cada padre, se intercambian los genes entre cada par de cromosomas para formar un gameto, y posteriormente los gametos de los 2 padres se aparean para formar un solo conjunto de cromosomas diploides.

Durante la mutación, se cambian nucleótidos individuales de padre a hijo. La mayoría de estos cambios se producen por errores de copiado.

La capacidad (aptitud) de un individuo se define como la probabilidad de que este viva para reproducirse (viabilidad), o como una función del número de descendientes que éste tiene (fertilidad).

Se denomina ambiente a todo aquello que rodea a un organismo. Puede ser "físico" (abiótico) o biótico. En ambos casos, el organismo ocupa un nicho que ejerce una influencia sobre su capacidad dentro del ambiente total.

Un ambiente biótico puede presentar funciones de aptitud dependientes de la frecuencia dentro de una población. En otras palabras, la capacidad del comportamiento de un organismo puede depender de cuántos más estén comportándose igual.

A través de varias generaciones, los ambientes bióticos pueden fomentar la co-evolución, en la cual la capacidad se determina mediante la selección parcial de otras especies.

La selección es el proceso mediante el cual algunos individuos en una población son seleccionados para reproducirse, típicamente en base a su capacidad.

La selección dura se da cuando sólo los mejores individuos se mantienen para generar progenie futura.

La selección blanda se da cuando se usan mecanismos probabilísticos para mantener como padres a individuos que tengan aptitudes relativamente bajas.

Se llama pleitropía al efecto en el cual un solo gene puede afectar simultáneamente a varios rasgos fenotípicos. Un ejemplo es un problema con la célula responsable de formar la hemoglobina. Al fallar, se afecta la circulación sanguínea, las funciones del hígado y las acciones capilares.

Cuando una sola característica fenotípica de un individuo puede ser determinada mediante la interacción simultánea de varios genes, se denomina poligenia. El color del cabello y de la piel son generalmente rasgos poligénicos.

Aunque no existe una definición universalmente aceptada de especie, diremos que es una colección de criaturas vivientes que tienen características similares, y que se pueden reproducir entre sí. Los miembros de una especie ocupan el mismo nicho ecológico.

Se denomina especiación al proceso mediante el cual aparece una especie. La causa más común de especiación es el aislamiento geográfico.

Si una sub población de una cierta especie se separa geográficamente de la población principal durante un tiempo suficientemente largo, sus genes divergirán. Estas divergencias se deben a diferencias en la presión de selección en diferentes lugares, o al fenómeno conocido como desvío genético.

Se llama desvío genético a los cambios en las frecuencias de genes/alelos en una población con el paso de muchas generaciones, como resultado del azar en vez de la selección. El desvío genético ocurre más rápidamente en poblaciones pequeñas y su mayor peligro es que puede conducir a que algunos alelos se extingan, reduciendo en consecuencia la variabilidad de la población.

En los ecosistemas naturales, hay muchas formas diferentes en las que los animales pueden sobrevivir (en los árboles, de la cacería, en la tierra, etc.) y cada estrategia de supervivencia es llamada un "nicho ecológico". Dos especies que ocupan nichos diferentes (p.ej. una que se alimenta de plantas y otra que se alimenta de insectos) pueden coexistir entre ellas sin competir, de una manera estable. Sin embargo, si dos especies que ocupan el mismo nicho se llevan a la misma zona, habrá competencia, y a la larga, la especie más débil se extinguirá (localmente). Por lo tanto, la diversidad de las especies depende de que ocupen una diversidad de nichos (o de que estén separadas geográficamente).

Se denomina reproducción a la creación de un nuevo individuo, si es a partir de un progenitor la reproducción es asexual y si es a partir de dos progenitores la reproducción es sexual.

Se denomina migración a la transferencia de (los genes de) un individuo de una sub población a otra.

Se dice que un gene es epistático cuando su presencia suprime el efecto de un gene que se encuentra en otra posición. Los genes epistáticos son llamados algunas veces genes de inhibición por el efecto que producen sobre otros genes.

2.2.5. ALGORITMOS GENÉTICOS

En 1859, Darwin publica su libro «El origen de las especies». Este libro causó una agria polémica en el mundo científico por las revolucionarias teorías en él contenidas: que las especies evolucionan acorde al medio, para adaptarse a éste. Con ello, el universo pasaba de ser una creación de Dios, estática y perfecta desde su inicio, a un conjunto de individuos en constante competición y evolución para poder perpetuar su especie en el tiempo. La existencia de una especie pasa así a ser algo dinámico; las especies se crean, evolucionan y desaparecen si no se adaptan. Para cada especie animal, la naturaleza proponía un crudo filtro: sólo los mejores, los más aptos, los que mejor se adapten al medio conseguían sobrevivir lo suficiente para llegar a la madurez, y encontrar una pareja para perpetuar sus aptitudes que le hacían más apto.

La informática ve aquí un claro proceso de optimización. Tomamos los individuos mejor adaptados -mejores soluciones temporales-, los cruzamos -mezclamos-, generando nuevos individuos -nuevas soluciones- que contendrán parte del código genético -información- de sus dos antecesores, y, por lo tanto, aunque el nuevo individuo no tenga que estar forzosamente mejor adaptado --de hecho, puede que ni la probabilidad de que el nuevo individuo generado esté mejor adaptado que los padres sea alta-, el promedio de la adaptación de toda la población si mejora, ya que tienden a perpetuarse y extenderse las mejores características, y a extinguirse las poco beneficiosas o perjudiciales. Aquí vemos que, a diferencia de los métodos anteriormente citados, no tenemos porque desarrollar un individuo mejor. En los algoritmos genéticos creamos una nueva abstracción -la población- cuya función de coste mejorará globalmente, por lo que puede que nos encontremos algún individuo con mejores características.

John Holland en 1975 en su célebre artículo , intuyó la posibilidad de incorporar la semántica de la evolución natural a procesos de optimización, y comenzaron así los estudios en algoritmos genéticos.

¿Qué es un algoritmo genético?

Analicemos el modelo biológico que desde el punto de vista puramente informático. ¿Podemos crear un algoritmo con la misma filosofía que emplea la naturaleza? Parece que a la naturaleza le va bien el método. Y con este planteamiento nacieron los algoritmos genéticos.

La idea básica es la siguiente: generemos un conjunto con algunas de las posibles soluciones. Cada una va a ser llamada individuo, y a dicho conjunto se le denominará población.

Cada individuo tiene una información asociada a él. En un problema de optimización corresponde a las variables libres, es decir, aquellas a las que el algoritmo tiene que asignar un valor para que una función sea mínima o máxima para esos valores. Esta función en nuestro modelo biológico, se denominará función de adaptación (también llamados función de aptitud o de capacidad) en nuestro modelo; y determina el grado de adaptación de un individuo. A dicha información se la va a denominar código genético.

Las características de los individuos, sean beneficiosas o no, se van a denominar fenotipos. La información asociada a un individuo se compone de partes indivisibles denominados cromosomas.

Un fenotipo puede estar en más de un cromosoma, en cuyo caso puede ser que el hijo herede un fenotipo que no tenía ni el padre ni la madre, sino una combinación de ambos. Un ejemplo en el humano es el color de la piel o la estructura del cráneo. En caso de que el hijo tenga parte de los genes del padre y parte de los genes de la madre que intervienen en un fenotipo, se va a crear una característica nueva asociada a ese fenotipo. De todas formas, no es un enfoque muy frecuente, ya que debemos asegurar que el conjunto de los genomas tendrá ley de composición interna respecto al operador de cruce definido sobre el alfabeto cromosómico. Por otro lado, que un cromosoma codifique más de un fenotipo es más raro todavía. El cromosoma de-

be tener en dicho caso tantos valores como el producto del número de valores posibles que tenga cada fenotipo del cromosoma. Esto es un inconveniente para los problemas de naturaleza discreta y de conjunto de soluciones acotado, ya que puede ocurrir que no podamos dar nunca soluciones que pertenezcan a alguna parte del espacio de estados -ley de composición interna-. En el caso de los problemas continuos o de alfabeto no acotado es peor, ya que en muy contadas ocasiones podemos asegurar la ley de composición interna del operador de cruce. Obsérvese que, tanto la forma de codificar los fenotipos en los cromosomas, como la determinación de qué es fenotipo -o dicho de otra forma, como la información va a ser almacenada en el código genético- son de vital importancia en los algoritmos genéticos. Escoger equivocadamente la forma de almacenar la información puede ralentizar la convergencia -es decir, que tardemos más en encontrar la solución-, o que no converja de ninguna forma -es decir, que la población esté errando aleatoriamente por efecto de las mutaciones y de los cruzamientos, sin llegar nunca a un punto estable, en un fenómeno que se denomina deriva genética-. Puede ser peor, ya que, si existen variables libres en la función de adaptación no controladas dentro del genoma, podemos llegar a una solución estable que no es el mínimo de la función, y que puede que ni siquiera sea el mínimo local. Es interesante que parte de la información heurística de la que disponemos haya de ser suficiente como para permitirnos una codificación razonablemente buena; si no, la implementación de este algoritmo y su convergencia no será satisfactoria.

Características de los algoritmos genéticos

Algunas de las características de los algoritmos genéticos son:

- Son algoritmos estocásticos. Dos ejecuciones distintas pueden dar dos soluciones distintas.

- Son algoritmos de búsqueda múltiple, luego dan varias soluciones. Aunque habitualmente las energías de los individuos de la población final es similar, los individuos suelen ser distintos entre sí. Con el modelo de paralelización empleado -genético multipoblacional- la probabilidad de obtener muchas soluciones distintas es más alta todavía. Por ello, nos podemos quedar con la solución que más nos convenga según la naturaleza del problema.
- Son los algoritmos que hacen una barrida mayor al sub espacio de posibles soluciones válidas, y con diferencia. De hecho, se considera que, de todos los algoritmos de optimización estocásticos, los algoritmos genéticos son de los más exploratorios disponibles.
- A diferencia de los otros algoritmos, cuya convergencia y resultado final son fuertemente dependientes de la posición inicial, en los algoritmos genéticos -salvo poblaciones iniciales realmente degeneradas, en los que el operador de mutación va a tener mucho trabajo- la convergencia del algoritmo es poco sensible a la población inicial si esta se escoge de forma aleatoria y es lo suficientemente grande.
- Por su grado de penetración casi nulo, la curva de convergencia asociada al algoritmo presenta una convergencia excepcionalmente rápida al principio, que casi enseguida se bloquea. Esto se debe a que el algoritmo genético es excelente descartando sub espacios realmente malos. Cada cierto tiempo, la población vuelve dar el salto evolutivo, y se produce un incremento en la velocidad de convergencia excepcional. La razón de esto es que algunas veces aparece una mutación altamente beneficiosa, o un individuo excepcional, que propaga algún conjunto de cromosomas excepcional al resto de la población. La pobre penetración es el agujero más importante en los algoritmos genéticos, y la razón fundamental de la aparición de los algoritmos híbridos y miméticos, para

acelerar los saltos evolutivos, además de incrementarse la velocidad de convergencia.

- La optimización es función de la representación de los datos. Este es el concepto clave dentro de los algoritmos genéticos, ya que una buena codificación puede hacer la programación y la resolución muy sencillas, mientras que una codificación errada nos va a obligar a estudiar que el nuevo genoma cumple las restricciones del problema, y en muchos problemas tendremos que abortar los que no cumplan las restricciones, por ser estas demasiado complejas. Además, la velocidad de convergencia va a estar fuertemente influenciada por la representación.
- Es una búsqueda paramétricamente robusta. Eso quiere decir que hemos de escoger realmente mal los parámetros del algoritmo para que no converja. Con tasas razonables, va a converger -mejor o peor- en una solución razonablemente buena si la representación es la adecuada. Esto es muy importante por la naturaleza de nuestra búsqueda. Nosotros no podemos hacer comparativas buscando los mejores números mágicos para que nuestro algoritmo converja, ya que el objetivo es llegar a la solución de un problema que ya de por sí es muy complejo. Por ello necesitamos un algoritmo en el cual podamos equivocarnos en los parámetros de partida, y estos son los algoritmos genéticos.
- Por último, los algoritmos genéticos son intrínsecamente paralelos. Esto significa que, independientemente de que lo hayamos implementado de forma paralela o no, buscan en distintos puntos del espacio de soluciones de forma paralela. Ese paralelismo intrínseco permite que sean fácilmente paralelizables, es decir, que sea fácil modificar el código para que se ejecute simultáneamente en varios procesadores.

Criterios para implementar un algoritmo genético

Los criterios que hay que tomar para implementar un algoritmo genético son:

- Criterio de codificación. Como se va a almacenar la información en el genoma.
- Criterio de tratamiento de individuos no factibles. Como se van a tratar a los individuos que no cumplan las restricciones.
- Criterio de inicialización. Cómo se va a construir la población inicial del algoritmo genético.
- Criterio de parada. Determina cuándo el algoritmo ha llegado a una solución aceptable.
- Función de adaptación. Corresponde a la función de costo de la investigación operativa tradicional.
- Operadores genéticos. Se emplean para determinar cómo va a ser la nueva generación. Básicamente son los operadores de cruce y mutación, aunque pueden ser empleados otros adicionales.
- Criterios de reemplazo. Los criterios que determinan quiénes se van a cruzar.
- Parámetros de funcionamiento. Determinados parámetros que, sin poder ser englobados en ninguno de los anteriores, son fundamentales para el funcionamiento de un algoritmo genético. Es el caso, por ejemplo, del tamaño de la población, la probabilidad de la aplicación de los operadores genéticos

2.2.6. DISEÑO DE RESORTES HELICOIDALES DE COMPRESIÓN

Spotts[2] y Shoups señalan que el diseño de resortes helicoidales depende de numerosas variables y que es difícil encontrar el diseño más eficiente por

[2] Spotts M. F. y Shoup, T.E. Elementos de Máquinas. Prentice Hall, México, 1999.

métodos de aproximaciones sucesivas. Para un resorte cargado estáticamente, si δ es la deflexión, G el módulo, Q el número de espiras inactivas en ambos extremos, P la carga , τ el esfuerzo y B una constante definida como:

$$B = \frac{\delta G}{Q\sqrt{8P\pi\tau}}$$

Spotts indica que puede demostrarse que si B se determina con la siguiente ecuación, en términos del índice c del resorte, el resorte resultante cargado estáticamente contendrá la cantidad mínima posible de material.

$$B = \frac{c^3(5c+1.23)}{2.46\sqrt{c+0.615}}$$

Donde el factor de esfuerzo para cortante transversal k_s, se incluye en este resultado.

Respecto al diseño óptimo de resortes helicoidales, Spotts manifiesta que en muchas aplicaciones, un resorte funciona a través de una gama de operaciones y debe diseñarse para satisfacer las siguientes condiciones:

(1) En la condición más extendida, el resorte debe ser capaz de ejercer una fuerza dada P_1.

(2) En su condición más comprimida, el esfuerzo cortante torsional no debe exceder un valor específico τ_2:

Si sólo se consideran el esfuerzo cortante torsional y las espiras activas, es fácil demostrar que si el resorte se diseña de manera que la carga, el esfuerzo y la deflexión mínimas sean exactamente la mitad de la carga, del esfuerzo y de la deflexión máximas, el resorte contendrá la menor cantidad posible de material.

Para el diseño de resortes helicoidales, Shigley[3] considera:

---

[3] Mischke, C. R. y Shigley, J. E. Diseño en Ingeniería Mecánica, McGraw-Hill, México, 1990.

• El espacio en el que debe adaptarse y operar.

• Valores de las fuerzas y deformaciones que se producirán.

• Exactitud y confiabilidad necesarias.

• Tolerancias y variaciones permisibles de las especificaciones.

• Condiciones ambientales, como temperatura y entorno corrosivo.

• Costo y cantidades que se necesitan.

Shigley utiliza estos factores a fin de seleccionar el material y especificar los valores adecuados para el tamaño de alambre, el número de espiras, el diámetro y la longitud libre, el tipo de extremos y el módulo del resorte necesarios para satisfacer los requisitos de carga y deformación de trabajo. Refiere que Samónov expresa que, para resortes de compresión, las restricciones de diseño primario son que el tamaño del alambre esté disponible comercialmente, y que el esfuerzo que se produce según la longitud cerrada no sea mayor que la resistencia de fluencia a la torsión. Su meta, en una solución completa mediante computadora, es utilizar totalmente el material.

Shigley manifiesta:

"Diagramas y nomogramas se han empleado en muchos casos para simplificar el problema de diseño de un resorte.

Existen casi tantas maneras de producir un programa para diseño de resortes como programadores; y no hay nada inusitado en el que se presenta aquí, y que funciona bien. Tal programa, que sirve para diseñar resortes helicoidales de compresión, puede usarse como punto partida para la elaboración de otros programas. El programa en cuestión consta de siete subrutinas independientes, todas las cuales utilizan las mismas localidades de memoria. Las subrutinas que se deben usar en el orden en el que se exponen, son:

- Ingresar y hacer que se visualice (en la pantalla o se imprima) el diámetro exterior.

- Ingresar y visualizar el número total de espiras. Ingresar y visualizar el número de espiras inactivas. Calcular y visualizar las espiras activas
- Seleccionar un material, e ingresar y visualizar el exponente y el coeficiente.
- Ingresar y visualizar el diámetro del alambre. Calcular y visualizar la resistencia de fluencia a la torsión.
- Ingresar y visualizar el esfuerzo de torsión máximo deseado cuando el resorte esté completamente cerrado. Calcular y visualizar la longitud cerrada, la longitud libre y la fuerza requerida para cerrar el resorte por completo.
- Calcular y visualizar la constante del resorte.
- Ingresar y visualizar cualquier fuerza de trabajo deseada F. Calcular y visualizar los valores correspondientes del esfuerzo por torsión y la deformación del resorte.

Se utilizan subrutinas independientes en este programa a fin de evitar el reingreso de todos e los datos cuando sólo ha de cambiarse un parámetro. De esta manera es fácil observar el efecto de un solo cambio. Por ejemplo, después de correr el programa una vez puede ser deseable ensayar con un diámetro diferente del alambre. Esto es posible hacerlo dando como dato el nuevo diámetro del material en la subrutina 4 y proseguir desde ese punto."

En el diseño de resortes helicoidales a la compresión para cargas estáticas, según Norton[4], los requisitos funcionales para un diseño de resorte llegan a ser bastantes diversos. Pudiera existir un requisito para una fuerza en particular a cierta deflexión o se define la tasa de resorte a un rango de deflexión. En algunos casos hay limitaciones de diámetro exterior, diámetro interior o longitud de trabajo. El procedimiento para el diseño variará dependiendo de estos requisitos. En cualquier caso, el diseño de resortes es en sí un pro-

[4]Norton R. L. Diseño de Máquinas. Prentice Hall, México, 1999.

blema iterativo. Deberán efectuarse ciertas suposiciones o hipótesis para establecer los valores de suficientes variables a fin de calcular esfuerzos, deflexiones y tasa de resorte. Dado que en las ecuaciones de esfuerzo y de deflexiones el tamaño del alambre aparece a la tercera o cuarta potencia, y en vista que la resistencia del material depende del tamaño del alambre, la seguridad del diseño se toma muy sensible a este parámetro.

Para Norton es posible recurrir a muchos procedimientos para el diseño de un resorte y más de una combinación de parámetros de resortes llegan a satisfacer cualquier conjunto de requisitos funcionales. Es posible optimar parámetros como el peso del resorte para un conjunto dado de especificaciones de rendimiento. A fin de minimizar peso y costo, los niveles de esfuerzos deberán diseñarse tan elevados como posible, sin causar fluencia estática durante el servicio. Deberá suponerse un diámetro de alambre de prueba d y un índice razonable de resorte C a partir de los cuales se calcula el diámetro de la espira D con la ecuación: C=D/d . Luego se escogerá un material de prueba para el resorte y se calcularán las resistencias importantes del material para el diámetro del alambre de prueba. Se calcula el esfuerzo antes de calcular la deflexión dado que, aunque ambos implican d y D, sólo la deflexión depende de N_a (número de espiras activas). Si está definida una fuerza requerida F, el esfuerzo a esa fuerza se calcula con la ecuación

$\tau_{\max} = K_s \frac{8FD}{\pi d^3}$ donde $K_S = \left(1 + \frac{0.5}{C}\right)$ K_S = factor de cortante directo

o

$\tau_{\max} = K_w \frac{8FD}{\pi d^3}$ donde $K_w = \frac{4C-1}{4C-4} + \frac{0.615}{C}$ K_w = factor Wahl a la torsión,

según resulte apropiado. Si se definen dos fuerzas de operación con una deflexión especificada entre ambas, ellas definirán la tasa de resorte.

Norton compara el estado del esfuerzo con el límite elástico para cargas estáticas. El factor de seguridad para una carga estática es:

$$N_S = \frac{S_{ys}}{\tau}$$

donde S_{ys} es el límite elástico al cortante y τ es el esfuerzo cortante, si el esfuerzo calculado resulta demasiado elevado en comparación con la resistencia del material, para mejorar el resultado se modifica el diámetro, la tasa de resorte o el material del alambre. Cuando parezcan razonables los esfuerzos calculados y la fuerza requerida de operación, en comparación con la resistencia del material, es posible suponer un número de espiras y de holgura de golpeo de prueba, y efectuar cálculos posteriores para la tasa de resorte o la deflexión y la longitud libre usando las ecuaciones:

$$y = \frac{8FD^3N_a}{d^4G} \qquad k = \frac{F}{y} = \frac{d^4G}{8D^3N_a}$$

siendo "y" la deflexión y G el módulo de corte del material. Valores fuera de lo razonable de cualquiera de estos parámetros requerirá una iteración adicional con hipótesis modificadas. Después de varias iteraciones, por lo general se podrá encontrar una combinación razonable de parámetros. Algunas de las cosas que se verifican antes de pensar que el diseño está completo será el esfuerzo a la altura de cierre, el D_i, el D_o y la longitud libre de la espira, respecto a consideraciones volumétricas. Además, es necesario verificar la posibilidad de pandeo.

Norton manifiesta respecto al diseño de resortes helicoidales a la compresión que se diseñan para cargas a la fatiga, donde las cargas del resorte son dinámicas (varían en el tiempo), se da una situación de esfuerzo por fatiga. El proceso de diseño es similar al de la carga estática, con la diferencia, que se considera en un resorte cargado dinámicamente dos niveles de fuerzas F_{min} (carga fluctuante mínima) y F_{max} (carga fluctuante máxima). Se considera entonces que los componentes de fuerza alternante y medio como:

$$F_a = \frac{F_{max} - F_{min}}{2} \qquad F_a = \frac{F_{max} + F_{min}}{2}$$

también define una razón de fuerzas R_F de la forma:

$$R_F = \frac{F_{\min}}{F_{\max}}$$

Norton emplea en este caso también el límite de resistencia a la fatiga y el factor de seguridad fatiga-torsión. El procedimiento de diseño a la fatiga es en esencia el mismo que el de la carga estática. Norton, ofrece el pseudocódigo para el procedimiento mencionado y lo enfoca como un proceso iterativo.

Mott[5] precisa respecto al diseño de resortes helicoidales de compresión, que este consiste en especificar la geometría del resorte para que este opere bajo límites específicos de carga y deflexión, y puede ser con limitaciones de espacio. El material y el tipo de servicio los especifica considerando el ambiente y aplicación.

Mott muestra dos métodos de solución, para el problema de diseño y los implementa en un programa de computación usando el lenguaje de programación BASIC.

Mott en su método de solución 1, se enfoca directamente hacia la geometría del resorte a nivel general especificando el diámetro medio que satisfacerá las limitaciones en cuanto a espacio. El proceso exige que el responsable del diseño disponga de tablas con información relativa a diámetros de alambre (tabla de calibres y diámetros para resortes) y tensiones de diseño para el material con que se va a fabricar el resorte (gráficas de tensiones por esfuerzo de corte de diseño para alambre de resortes que se utilizan en resortes de compresión y en resortes helicoidales de compresión). Es necesario hacer una estimación inicial para las tensiones de diseño consultando las gráficas de las mismas contra diámetro de alambre para tomar una decisión razonada. En general deberá realizarse más de una prueba pero los resultados de pruebas previas le ayudarán a decidir los valores que debe utilizar pruebas posteriores.

---

[5] Mott, R. L. Diseño de Elementos de Máquinas. Prentice Hall, México, 1995

El método de solución 1 de Norton exige los siguientes pasos:

Paso 1. Especificación del material y su módulo de elasticidad ante esfuerzo de corte, G.

Paso 2. A partir del enunciado del problema, se identifica la fuerza en operación, F_o; la longitud en operación a la que debe ejercerse esa fuerza, L_o, la fuerza a alguna otra longitud, que se denomina fuerza instalada, F_i; y la longitud de instalado, L_i.

Paso 3. Cálculo de la razón de resorte, k:

$$k = \frac{F_o - F_i}{L_i - L_o}$$

Paso 4. Cálculo de la longitud libre, L_f.

$$L_f = L_i + \frac{F_i}{k}$$

El segundo término en la ecuación anterior es la cantidad de deflexión a partir de la longitud libre hasta la longitud instalada para poder desarrollar la fuerza F_i instalado. Desde luego este paso no es necesario si en la información original se especifica la longitud libre.

Paso 5. Especificación de un estimado inicial para el diámetro medio, D_m.

Teniendo en mente que el diámetro medio será más pequeño que el diámetro externo y más grande que el diámetro interior, es necesario empezar a aplicar el criterio.

Paso 6. Especificación de una tensión de diseño inicial.

Consulta de las gráficas para las tensiones de diseño de los materiales seleccionados, considerando a su vez, el servicio. Estrictamente, este es un estimado que se basa en la resistencia del material. El proceso incluye verificar la tensión más adelante.

Paso 7. Cálculo del diámetro del alambre para la prueba, Dw.

$$D_w = \left[\frac{8KF_oD_m}{\pi\tau_d}\right]^{1/3}$$

Observe que se conoce todo lo demás en la ecuación excepto el factor de Wahl, K, porque depende del propio diámetro del alambre. Pero K. varía poco a lo largo del rango de índices de resorte, C. K = 1.2 es el valor nominal. Esto, también, se verificara más adelante. Con el valor que se supuso para K, es posible simplificar en alguna medida

Paso 8. Selección de un diámetro de alambre estándar de las tablas, luego determine la tensión de diseño y la tensión máxima tolerable para el material con ese diámetro. En condiciones normales, la tensión de diseño será para servicio promedio, a menos que tasas altas de ciclaje o choque indiquen que se garantiza servicio severo. La curva de servicio ligero debe utilizarse con cuidado porque se acerca mucho a la resistencia a punto cedente. En realidad se utiliza la curva de servicio ligero como estimación del esfuerzo o tensión máxima permisible.

Paso 9. Cálculo de los valores reales de C y K, el índice de resorte y el factor de Wahl

$$C = \frac{D_m}{D_w} \qquad K = \frac{4C-1}{4C-4} + \frac{0.615}{C}$$

Paso 10. Cálculo de la tensión real que se espera debido a la fuerza de operación, F_o

$$\tau_o = \frac{8KF_oD_m}{\pi D_w^3}$$

Se compara esto con la tensión de diseño para ver si es segura.

Paso 11. Cálculo del número de bobinas activas que se necesitan para darle las características de deflexión adecuadas al resorte.

$N_a = \frac{GD_w}{8kC^3}$ donde k, es la razón del resorte k=F/f

Paso 12. Cálculo de la longitud comprimido, L_S; la fuerza en el resorte en longitud comprimido, F_S; y el esfuerzo o tensión en el resorte en longitud comprimido, τ_S; Este cálculo dará la tensión máxima que recibirá el resorte.

$$L_S = D_w(N_a + 2) \qquad F_S = k(L_f + L_S) \qquad \tau_S = \tau_o \frac{F_S}{F_o}$$

Se compara con la tensión máxima permisible, si es menor, es segura y el resorte no presentará cedencia cuando se comprima hasta su longitud estando totalmente comprimida.

Paso 13. Cálculo de las características geométricas

$OD = D_m + D_w$ $\qquad$ $OD = D_m + D_w$

Donde OD es el diámetro exterior e ID es el diámetro interior del alambre, estos se comparan con limitaciones de espacio y operación.

Tras este último paso se pone fin al diseño, se hacen otras pruebas para que el resorte se acerque más a lo óptimo. Norton sugiere que se debe verificar la tendencia al pandeo junto con el margen para las bobinas y los analiza como parte de un método que él denomina método 2.

En el método 2, Norton presenta un procedimiento de diseño más abierto pues proporciona un margen extenso para que el diseñador haga diseños que satisfagan las necesidades básicas en cuanto fuerza/longitud/tensión o esfuerzo. El inconveniente principal de este segundo método radica en que no se controla la geometría del resorte de manera estricta, y este método resulta siendo más un resultado del procedimiento que un factor que deba especificarse al inicio como en el método 1.

De todo lo anterior percibimos que el diseño es el proceso mediante el cual se proporciona la información necesaria para permitir la realización de un ente y determinar el resultado obtenido.

La optimización del ente es uno de los objetivos fundamentales del ente, para esto es necesario contar con un mecanismo para comparar porque una solución es mejor que otra, este mecanismo se denomina criterio. También

se debe comprobar si el diseño es o no aceptable a través del cumplimiento de las restricciones del problema.

En la optimización de diseño frecuentemente se usan modelos con los que se obtendrá una solución para un conjunto dado de datos de entrada, luego se evalúa la solución en función de las restricciones y los criterios. Se repite este proceso tantas veces como sea necesaria para obtener una solución satisfactoria basándose en las evaluaciones de soluciones previas.

2.3. MARCO CONCEPTUAL

2.3.1. CONCEPTOS EN ALGORITMOS GENÉTICOS

2.3.1.1. Cromosoma

Estructura de datos que contiene una cadena de parámetros de diseño o genes. Esta estructura de datos puede almacenarse, por ejemplo, como una cadena de bits o un arreglo de enteros (ver Figura II-2).

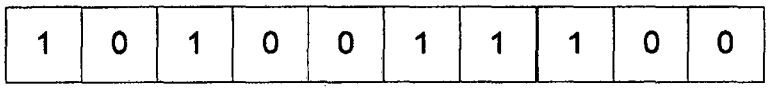


Figura II-2. Ejemplo de Cadena cromosómica.

2.3.1.2. Gene

Es una sub sección de un cromosoma que (usualmente) codifica el valor de un solo parámetro (Figura II-3).

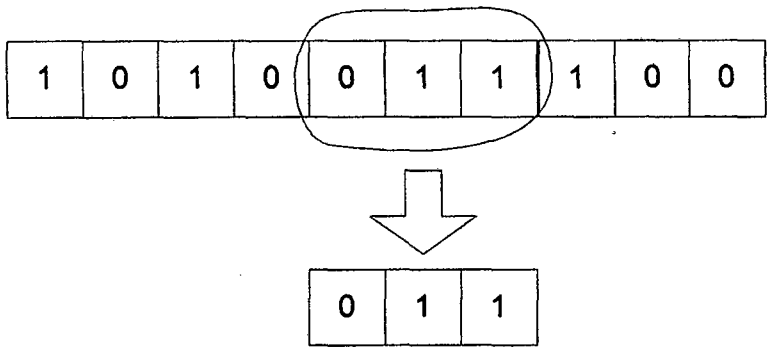


Figura II-3. Ejemplo de gene.

2.3.1.3. *Genotipo*

Es la codificación (por ejemplo, binaria) de los parámetros que representan una solución del problema a resolverse (ver Figura II-2).

2.3.1.4. *Fenotipo*

Es la decodificación del cromosoma. Es decir, a los valores obtenidos al pasar de la representación (binaria) a la usada por la función objetivo (ver Figura II-4).

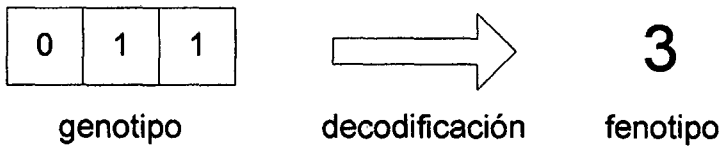


Figura II-4. Ejemplo de fenotipo.

2.3.1.5. *Individuo*

Se denomina individuo a un solo miembro de la población de soluciones potenciales a un problema. Cada individuo contiene un cromosoma (o de manera más general, un genoma) que representa una solución posible al problema a resolverse (ver Figura II-5).

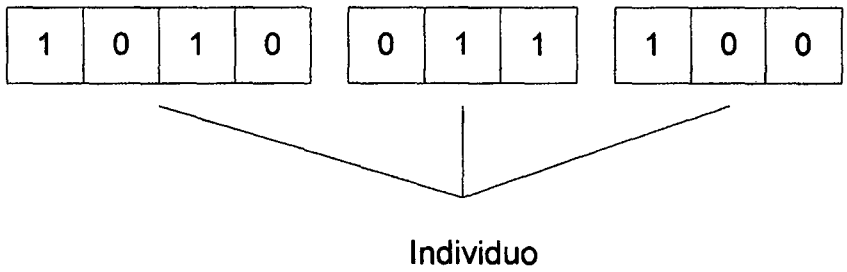


Figura II-5. Ejemplo de individuo.

2.3.1.6. *Capacidad (aptitud)*

Es el valor que se asigna a cada individuo y que indica que tan bueno es este con respecto a los demás para la solución de un problema. Por ejemplo,

si $f(x)=x^3$, entonces $f(101_2)=125$, donde $f(x)$ es la función de capacidad (aptitud).

2.3.1.7. *Paisaje de Capacidad (Fitness Landscape)*

Es la híper superficie que se obtiene al aplicar la función de aptitud a cada punto del espacio de búsqueda.

2.3.1.8. *Alelo*

Es cada valor posible que puede adquirir una cierta posición genética. Si se usa representación binaria, un alelo puede valer 0 ó 1 (ver Figura II-6).

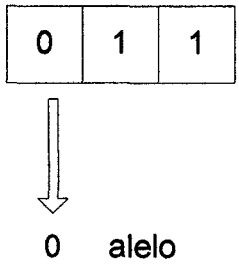


Figura II-6. Ejemplo de alelo.

2.3.1.9. *Generación*

Es una iteración de la medida de aptitud y a la creación de una nueva población por medio de operadores de reproducción.

2.3.1.10. *Sub Poblaciones*

Son subdivisiones en grupos de una población. Normalmente, sólo pueden cruzarse entre sí los individuos que pertenezcan a la misma sub población. En los esquemas con sub poblaciones, suele permitirse la migración de una sub población a otra (sobre todo en el contexto de algoritmos evolutivos paralelos).

2.3.1.11. *Especiación*

Es el hecho de permitir la cruza sólo entre individuos de la misma sub población.

2.3.1.12. Migración

Se llama migración a la transferencia de (los genes de) un individuo de una sub población a otra.

2.3.1.13. Población Panmítica

Población en la que cualquier individuo puede reproducirse con otro con una probabilidad que depende sólo de su capacidad.

Lo opuesto de la población panmítica es permitir la reproducción sólo entre individuos de la misma sub población. La mayor parte de los algoritmos evolutivos (EAs) convencionales usan poblaciones panmíticas.

Debido a ruidos estocásticos, los EAs tienden a converger a una sola solución. Para evitar eso, y mantener la diversidad, existen técnicas que permiten crear distintos nichos para los individuos.

2.3.1.14. Epístasis

Es la interacción entre los diferentes genes de un cromosoma. Se refiere a la medida en que la contribución de aptitud de un gene depende de los valores de los otros genes.

Cuando un problema tiene poca epístasis (o ninguna), su solución es trivial (un algoritmo escalando la colina es suficiente para resolverlo). Cuando un problema tiene una epístasis elevada, el problema será deceptivo, por lo que será muy difícil de resolverlo por un EA.

2.3.1.15. Bloque Constructor

Es un grupo pequeño y compacto de genes que han co-evolucionado de tal forma que su introducción en cualquier cromosoma tiene una alta probabilidad de incrementar la capacidad de dicho cromosoma.

2.3.1.16. Decepción

Se llama decepción a la condición donde la combinación de buenos bloques constructores lleva a una reducción de capacidad, en vez de un incremento.

Este fenómeno fue sugerido originalmente por Goldberg[6] para explicar el mal desempeño del GA en algunos problemas.

2.3.1.17. Operador de reproducción

Es aquel mecanismo que influencia la forma en que se pasa la información genética de padres a hijos. Los operadores de reproducción caen en tres amplias categorías: cruce, mutación y reordenamiento.

Operador de Cruce

> Es un operador que forma un nuevo cromosoma combinando partes de cada uno de sus cromosomas padres.

Operador de Mutación

> Es un operador que forma un nuevo cromosoma a través de alteraciones (usualmente pequeñas) de los valores de los genes de un solo cromosoma padre.

Operador de reordenamiento

> Es un operador que cambia el orden de los genes de un cromosoma, con la esperanza de juntar los genes que se encuentren relacionados, facilitando así la producción de bloques constructores. La inversión es un ejemplo de un operador de reordenamiento en el que se invierte el orden de todos los genes comprendidos entre 2 puntos seleccionados al azar en el cromosoma.

En un algoritmo genético (GA), cuando una población no tiene variedad de requisito, la cruza no será útil como operador de búsqueda, porque tendrá propensión a simplemente regenerar a los padres.

---

[6] Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

Es importante aclarar que en los GAs los operadores de reproducción actúan sobre los genotipos y no sobre los fenotipos de los individuos.

2.3.1.18. Elitismo

Se denomina elitismo al mecanismo utilizado en algunos EAs para asegurar que los cromosomas de los miembros más capaces (aptos) de una población se pasen a la siguiente generación sin ser alterados por ningún operador genético. Usar elitismo asegura que la capacidad máxima de la población nunca se reducirá de una generación a la siguiente. Sin embargo, no necesariamente mejora la posibilidad de localizar el óptimo global de una función. No obstante, es importante hacer notar que se ha demostrado que el uso de elitismo es vital para poder demostrar convergencia de un algoritmo genético[7].

2.3.1.19. Explotación

Cuando se atraviesa un espacio de búsqueda, se denomina explotación al proceso de usar la información obtenida de los puntos visitados previamente para determinar qué lugares resulta más conveniente visitar a continuación.

2.3.1.20. Exploración

Es el proceso de visitar completamente nuevas regiones del espacio de búsqueda, para ver si puede encontrarse algo prometedor. La exploración involucra grandes saltos hacia lo desconocido. La explotación normalmente involucra movimientos finos. La explotación es buena para encontrar óptimos locales. La exploración es buena para evitar quedar atrapado en óptimos locales.

---

[7] Rudolph, G., Convergence Analysis of Canonical Genetic Algorithms. IEEE Transactions on Neural Networks, 5:96-101, January 1994.

2.3.1.21. Esquema

Es un patrón de valores de genes de un cromosoma que puede incluir estados "comodín". Usando un alfabeto binario, los esquemas se forman del alfabeto 0,1,#. Por ejemplo, el cromosoma 0110 es una instancia del esquema #1#0 (donde # significa "comodín").

2.3.1.22. Algoritmo Genético aplicado

La idea básica es la siguiente: generemos un conjunto con algunas de las posibles soluciones. Cada una va a ser llamada individuo, y a dicho conjunto se le denominará población. Cada individuo tiene una información asociada a él.

En un problema de optimización corresponde a las variables libres, es decir, aquellas a las que el algoritmo tiene que asignar un valor para que una función sea mínima o máxima para esos valores. Esta función se denominará función de capacidad y determina el grado de adaptación de un individuo. A dicha información se la va a denominar código genético (genoma).

Las características de los individuos, sean beneficiosas o no, se van a denominar fenotipos. La información asociada a un individuo se compone de partes indivisibles denominados cromosomas.

Un fenotipo puede estar en más de un cromosoma, en cuyo caso puede ser que el hijo herede un fenotipo que no tenía ni el padre ni la madre, sino una combinación de ambos. Un ejemplo en el humano es el color de la piel o la estructura del cráneo. En caso de que el hijo tenga parte de los genes del padre y parte de los genes de la madre que intervienen en un fenotipo, se va a crear una característica nueva asociada a ese fenotipo. De todas formas, no es un enfoque muy frecuente, ya que debemos asegurar que el conjunto de los fenomas tendrá ley de composición interna respecto al operador de cruce definido sobre el alfabeto cromosómico. Por otro lado, que un cromosoma codifique más de un fenotipo es más raro todavía. El cromosoma debe tener en dicho caso tantos valores como el producto del número de valores

posibles que tenga cada fenotipo del cromosoma. Esto es un inconveniente para los problemas de naturaleza discreta y de conjunto de soluciones acotado, ya que puede ocurrir que no podamos dar nunca soluciones que pertenezcan a alguna parte del espacio de estados -ley de composición interna-. En el caso de los problemas continuos o de alfabeto no acotado es peor, ya que en muy contadas ocasiones podemos asegurar la ley de composición interna del operador de cruce. Obsérvese que, tanto la forma de codificar los fenotipos en los cromosomas, como la determinación de qué es fenotipo -o dicho de otra forma, como la información va a ser almacenada en el código genético- son de vital importancia en los algoritmos genéticos. Escoger equivocadamente la forma de almacenar la información puede ralentizar la convergencia -es decir, que tardemos más en encontrar la solución-, o que no converja de ninguna forma -es decir, que la población esté errando aleatoriamente por efecto de las mutaciones y de los cruzamientos, sin llegar nunca a un punto estable, en un fenómeno que se denomina deriva genética-. Puede ser peor, ya que, si existen variables libres en la función de coste no controladas dentro del genoma, podemos llegar a una solución estable que no es el mínimo de la función de coste, y que puede que ni siquiera sea el mínimo local. Es interesante que parte de la información heurística de la que disponemos ha de ser suficiente como para permitirnos una codificación razonablemente buena; si no, la implementación de este algoritmo y su convergencia no será satisfactoria.

El algoritmo genético enfatiza la importancia del cruce sexual (operador principal) sobre el de la mutación (operador secundario), y usa selección probabilística.

El algoritmo básico es el siguiente:

- Generar (aleatoriamente) una población inicial.
- Calcular la capacidad (aptitud) de cada individuo.
- Seleccionar (probabilísticamente) en base a la capacidad.

- Aplicar operadores genéticos (cruza y mutación) para generar la siguiente población.
- Iterar hasta que cierta condición se satisfaga.

La representación tradicional es la binaria. Recordemos que a la cadena binaria se le llama "cromosoma", a cada posición de la cadena se le denomina "gene" y a cada valor dentro de esta posición se le llama "alelo".

Para poder aplicar el algoritmo genético se requiere de los 5 componentes básicos siguientes:

- Una representación de las soluciones potenciales del problema.
- Una forma de crear una población inicial de posibles soluciones (normalmente un proceso aleatorio)
- Una función de evaluación que juegue el papel del ambiente, clasificando las soluciones en términos de su "aptitud".
- Operadores genéticos que alteren la composición de los hijos que se producirán para las siguientes generaciones.
- Valores para los diferentes parámetros que utiliza el algoritmo genético (tamaño de la población, probabilidad de cruce, probabilidad de mutación, número máximo de generaciones, etc.)

2.3.1.23. Características de los Algoritmos Genéticos

Los algoritmos genéticos presentan ventajas que los hacen preferibles para determinado tipo de aplicaciones. Algunas de las características de los algoritmos genéticos son:

- Son algoritmos estocásticos. Dos ejecuciones distintas pueden dar dos soluciones distintas
- Son algoritmos de búsqueda múltiple, dan varias soluciones. Aunque habitualmente las capacidades de los individuos de la población final es similar, los individuos suelen ser distintos entre si. Con el modelo de para-

lelización empleado -genético multipoblacional- la probabilidad de obtener muchas soluciones distintas es más alta todavía. Por ello, nos podemos quedar con la solución que más nos convenga según la naturaleza del problema.

- Hacen una barrida profunda al sub espacio de posibles soluciones válidas, y con diferencia. De hecho, se considera que, de todos los algoritmos de optimización estocásticos, los algoritmos genéticos son de los más exploratorios disponibles.

- La convergencia y resultado final no son fuertemente dependientes de la posición inicial, -salvo poblaciones de inicio realmente degeneradas, en los que el operador de mutación va a tener mucho trabajo- la convergencia del algoritmo es poco sensible a la población inicial si esta se escoge de forma aleatoria y es lo suficientemente grande.

- Por su grado de penetración casi nulo, la curva de convergencia asociada al algoritmo presenta una convergencia excepcionalmente rápida al principio, que casi enseguida se bloquea. Esto de debe a que el algoritmo genético es excelente descartando subespacios realmente malos. Cada cierto tiempo, la población vuelve dar el salto evolutivo, y se produce un incremento en la velocidad de convergencia excepcional. La razón de esto es que algunas veces aparece una mutación altamente beneficiosa, o un individuo excepcional, que propaga algún conjunto de cromosomas excepcional al resto de la población. La pobre penetración es el agujero más importante en los algoritmos genéticos, y la razón fundamental de la aparición de los algoritmos híbridos y miméticos.

- La optimización es función de la representación de los datos. Este es el concepto clave dentro de los algoritmos genéticos, ya que una buena codificación puede hacer la programación y la resolución muy sencillas, mientras que una codificación errada nos va a obligar a estudiar que el nuevo genoma cumple las restricciones del problema, y en muchos problemas tendremos que abortar los que no cumplan las restricciones, por

ser estas demasiado complejas. Además, la velocidad de convergencia va a estar fuertemente influenciada por la representación.

- Es una búsqueda paramétricamente robusta. Eso quiere decir que hemos de escoger realmente mal los parámetros del algoritmo para que no converja. Con tasas razonables, va a converger -mejor o peor- en una solución razonablemente buena si la representación es la adecuada.
- Los algoritmos genéticos son intrínsecamente paralelos. Esto significa que, independientemente de que lo hayamos implementado de forma paralela o no, buscan en distintos puntos del espacio de soluciones de forma paralela. Ese paralelismo intrínseco permite que sean fácilmente paralelizables, es decir, que sea fácil modificar el código para que se ejecute simultáneamente en varios procesadores.

2.3.2. CODIFICACIÓN DEL GENOMA

2.3.2.1. Representación Binaria

La representación tradicional usada para codificar un conjunto de soluciones es el esquema binario en el cual un cromosoma es una cadena de la forma $\langle b_1, b_2, ..., b_m \rangle$ (ver Figura II-7), donde $b_1, b_2, ..., b_m$ se denominan *alelos* (ya sea ceros o unos).

1	0	1	0	0	1	1	1	0	0

Figura II-7. Ejemplo de cadena binaria

Hay varias razones por las cuales suele usarse la codificación binaria en los GAs, aunque la mayoría de ellas se remontan al trabajo de Holland en el área. En su libro[8], Holland dio una justificación teórica para usar codificaciones binarias.

[8] Holland J. H., Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Michigan, 1975.

2.3.2.2. *Códigos de Gray*

Un problema que fue notado desde los inicios de la investigación en GAs fue que el uso de la representación binaria no mapea adecuadamente el espacio de búsqueda con el espacio de representación[9]. Por ejemplo, si codificamos en binario los enteros 5 y 6, los cuales están adyacentes en el espacio de búsqueda, sus equivalentes en binario serían el 101 y el 110, los cuales difieren en 2 bits (el primero y el segundo de derecha a izquierda) en el espacio de representación. A este fenómeno se le conoce como el *risco de Hamming* (*Hamming cliff*)[10], y ha conducido a los investigadores a proponer una representación alternativa en la que la propiedad de adyacencia existente en el espacio de búsqueda pueda preservarse en el espacio de representación. La codificación de Gray es parte de una familia de representaciones que caen dentro de esta categoría. Podemos convertir cualquier número binario a un código de Gray haciendo XOR a sus bits consecutivos de derecha a izquierda. Por ejemplo, dado el número 0101 en binario, haríamos[11]: $1 \oplus 0 = 1$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, produciéndose (el último bit de la izquierda permanece igual) 0111, el cual es el código de Gray equivalente. Algunos investigadores han demostrado empíricamente que el uso de códigos de Gray mejora el desempeño del AG al aplicarse a las funciones de prueba clásicas de De Jong (De hecho, Mathias y Whitley[12] encontraron que la codificación de Gray no sólo elimina los riscos de Hamming, sino que también altera el número de óptimos locales en el espacio de búsqueda así como el tamaño de las buenas regiones de búsqueda (aquellas que nos conducirán a la ve-

---

[9] Hollstien R. B., Artificial Genetic Adaptation in computer control systems, PhD thesis, University of Michigan, Ann Harbor, Michigan, 1971

[10] Caruana R. y J. D. Schaffer J. D., Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms. Procedente de: Fifth International Conference on Machine Learning, . MorganKauffman Publishers, pages 132-161, San Mateo, California, 1988

[11] $\oplus$ indica XOR

[12] Mathias K. E. y Whitley L. D., Changing Representations During Search: A Comparative Study of Delta Coding. Evolutionary Computation, Págs. 249-278, 1994.

cindad del óptimo global). En su trabajo, Mathias y Whitley mostraron empíricamente que un mutador aleatorio del tipo "escalando la colina" es capaz de encontrar el óptimo global de la mayor parte de las funciones de prueba utilizadas cuando se emplea la codificación de Gray, a pesar de que algunas de ellas fueron diseñadas explícitamente para presentar dificultades a los algoritmos de búsqueda tradicionales (sean evolutivos o no).

2.3.2.3. Codificación de Números Reales

Aunque los códigos de Gray pueden ser muy útiles para representar enteros, el problema de mapear correctamente el espacio de búsqueda en el espacio de representación se vuelve más serio cuando tratamos de codificar números reales. En el enfoque tradicional, se usa un número binario para representar un número real, definiendo límites inferiores y superiores para cada variable, así como la precisión deseada. Por ejemplo, si queremos codificar una variable que va de 0.35 a 1.40 usando una precisión de 2 decimales, necesitaríamos $\log_2(140-35) \approx 7$ bits para representar cualquier número real dentro de ese rango. Sin embargo, en este caso, tenemos el mismo problema del que hablamos anteriormente, porque el número 0.38 se representaría como 0000011, mientras que 0.39 se representaría como 0000101.

Aunque se usen códigos de Gray, existe otro problema más importante cuando tratamos de desarrollar aplicaciones del mundo real: la alta dimensionalidad. Si tenemos demasiadas variables, y queremos una muy buena precisión para cada una de ellas, entonces las cadenas binarias que se produzcan se volverían extremadamente largas, y el AG tendería a tener un desempeño pobre. Si en vez de usar este tipo de mapeo adoptamos algún formato binario estándar para representar números reales, como por ejemplo el estándar del IEEE para precisión simple, en el cual un número real se representa usando 32 bits, de los cuales 8 se usan para el exponente usando una notación en exceso-127, y la mantisa se representa con 23 bits (ver Figura II-8), podríamos manejar un rango relativamente grande de números reales usando una cantidad fija de bits (por ejemplo, de 2^{-126} a 2^{127} si usa-

mos el estándar de precisión simple antes descrito). Sin embargo, el proceso de decodificación sería computacionalmente más costoso y el mapeo entre el espacio de representación y el de búsqueda sería mucho más complejo que cuando se usa una representación binaria simple, porque cualquier pequeño cambio en el exponente produciría grandes saltos en el espacio de búsqueda, mientras que perturbaciones en la mantisa podrían no cambiar de manera significativa el valor numérico codificado.

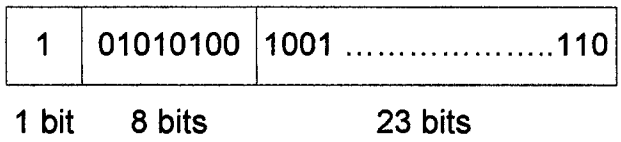


Figura II-8. Ejemplo de Notación IEEE.

Mientras los teóricos afirman que los alfabetos pequeños son más efectivos que los alfabetos grandes, los prácticos han mostrado a través de una cantidad significativa de aplicaciones del mundo real (particularmente problemas de optimización numérica) que el uso directo de números reales en un cromosoma funciona mejor en la práctica que la representación binaria tradicional[13] [14]. El uso de números reales en una cadena cromosómica (Figura II-9) ha sido común en otras técnicas de computación evolutiva tales como las estrategias evolutivas y la programación evolutiva, donde la mutación es el operador principal.

0.78	1.25	0.03	4.79	5.00	3.28	6.31

Figura II-9. Cromosoma con Representación Real.

Sin embargo, los teóricos de los GAs han criticado fuertemente el uso de valores reales en los genes de un cromosoma, principalmente porque esta

---

[13] Davis L., editor. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, New York, 1991.

[14] Larry J. Eshelman and J. Davis Schaffer. Real-coded Genetic Algorithms and Interval-Schemata. L. Darrell Whitley editores, Foundations of Genetic Algorithms 2, pages 187-202. Morgan Kaufmann Publishers, San Mateo, California, 1993.

representación de cardinalidad más alta tiende a hacer que el comportamiento del GA sea más errático y difícil de predecir. Debido a esto, se han diseñado varios operadores especiales en los años recientes, para emular el efecto de la cruza y la mutación en los alfabetos binarios[15].

Los prácticos argumentan que una de las principales capacidades de los Gas que usan representación real es la de explotar la "gradualidad"[16] de las funciones de variables continuas. Esto significa que los GAs con codificación real pueden lidiar adecuadamente con los "riscos" producidos cuando las variables utilizadas son números reales, porque un cambio pequeño en la representación es mapeado como un cambio pequeño en el espacio de búsqueda. En un intento por reducir la brecha entre la teoría y la práctica, algunos investigadores han desarrollado un marco teórico que justifique el uso de alfabetos de más alta cardinalidad, pero ha habido poco consenso en torno a los problemas principales, por lo que el uso de GAs con codificación real sigue siendo una elección que se deja al usuario.

Se han usado también otras representaciones de los números reales. Por ejemplo, el uso de enteros para representar cada dígito ha sido aplicado exitosamente a varios problemas de optimización[17]. La Figura II-10 muestra un ejemplo en el cual se representa el número 1.45679 usando enteros. En este caso, se supone una posición fija para el punto decimal en cada variable, aunque esta posición no tiene que ser necesariamente la misma para el resto de las variables codificadas en la misma cadena. La precisión está limitada por la longitud de la cadena, y puede incrementarse o decrementarse según se desee. Los operadores de cruce tradicionales (un punto, dos puntos y

---

[15] Wright A. H., Genetic Algorithms for Real Parameter Optimization. Gregory J. E. Rawlins editor, Foundations of Genetic Algorithms, pages 205-218. Morgan Kaufmann Publishers, San Mateo, California, 1991.

[16] Gradualidad se refiere a los casos en los cuales un cambio pequeño en las variables se traduce en un cambio pequeño en la función.

[17] Coello C. A., Christiansen A. D., and Hernández A., Using a New GA-Based Multiobjective Optimization Technique for the Design of Robot Arms. Robotica, 16(4):401-414, 1998.

uniforme) pueden usarse directamente en esta representación, y la mutación puede consistir en generar un dígito aleatorio para una cierta posición o bien en producir una pequeña perturbación (por ejemplo ±1) para evitar saltos extremadamente grandes en el espacio de búsqueda. Esta representación pretende ser un compromiso entre un AG con codificación real y una representación binaria de números reales, manteniendo lo mejor de ambos esquemas al incrementar la cardinalidad del alfabeto utilizado, pero manteniendo el uso de los operadores genéticos tradicionales casi sin cambios.

1	4	5	6	7	9

Figura II-10. Ejemplo de Representación Entera de Números Reales.

Alternativamente, podríamos también usar enteros largos para representar números reales (ver Figura II-11), pero los operadores tendrían que redefinirse de la misma manera que al usar números reales. El uso de este esquema de representación como una alternativa a los GAs con codificación real parece, sin embargo, un tanto improbable, ya que se tendrían que hacer sacrificios notables en la representación, y los ´únicos ahorros importantes que se lograrían serían en términos de memoria (el almacenamiento de enteros toma menos memoria que el de números reales). No obstante, este esquema ha sido usado en algunas aplicaciones del mundo real[18].

64852	12366	47821	45785

Figura II-11. Ejemplo de representación entera de números reales como un entero largo

2.3.2.4. Representaciones de Longitud Variable

En algunos problemas el uso de alfabetos de alta cardinalidad puede no ser suficiente, pues además puede requerirse el empleo de cromosomas de lon-

[18] Davis L., editor de. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, New York, 1991.

gitud variable para lidiar con cambios que ocurran en el ambiente con respecto al tiempo (por ejemplo, el decremento/incremento de la precisión de una variable o la adición/remoción de variables). Algunas veces, puede ser posible introducir símbolos en el alfabeto que sean considerados como posiciones "vacías" a lo largo de la cadena, con lo que se permite la definición de cadenas de longitud variable aunque los cromosomas tengan una longitud fija. Ese es, por ejemplo, el enfoque utilizado por Coello[19] para diseñar circuitos eléctricos combinatorios. En ese caso, el uso de un símbolo llamado WIRE, el cual representa la ausencia de compuerta, permitió cambiar la longitud de la expresión Booleana generada a partir de una matriz bidimensional.

(2,1)	(2,0)	(3,0)	(3,1)	
(1,1)	(1,0)	(1,1)	(4,1)	(4,0)

Figura II-12. Ejemplo de cromosoma en GA desordenados

Sin embargo, en otros dominios, este tipo de simplificación puede no ser posible y deben idearse representaciones alternativas. Por ejemplo, en problemas que tienen decepción parcial o total[20] (es decir, en aquellos problemas en los que los bloques constructores de bajo orden no guían al GA hacia el óptimo y no se combinan para formar bloques constructores de orden mayor), un GA no tendría un buen desempeño sin importar cuál sea el valor de sus parámetros (tamaño de población, porcentajes de cruza y mutación, etc.). Para lidiar con este tipo de problemas en particular, Goldberg[21] propu-

---

[19] Coello C. A., Christiansen A.D., and Hernández A. Automated Design of Combinational Logic Circuits using Genetic Algorithms, D. G. Smith, N. C. Steele, and R. F. Albrecht, editores, Procedentede: International Conference on Artificial Neural Nets and Genetic Algorithms ICANNGA 97, pages 335-338, Norwich, England, April 1997. University of East Anglia, Springer-Verlag.

[20]. Grefenstette J., Deception Considered Harmful, editor: Darrell Whitley,Foundations of Genetic Algorithms 2, pages 75-91.Morgan Kaufmann, SanMateo, California, 1993.

[21] Goldberg D., Deb K., y Korb B., Messy genetic algorithms revisited: Studies in mixed size and scale. Complex Systems, 4:415-444, 1990.

sieron el uso de un tipo especial de GA de longitud variable el cual usa poblaciones de tamaño variable. A este GA especial se le denominó 'desordenado' (*messy* GA o mGA) en contraposición con el GA estándar (u ordenado), que tiene longitud y tamaño de población fijos. La idea básica de los GAs desordenados es empezar con cromosomas cortos, identificar un conjunto de buenos bloques constructores y después incrementar la longitud del cromosoma para propagar estos buenos bloques constructores a lo largo del resto de la cadena.

La representación usada por los GAs desordenados es muy peculiar, puesto que cada bit está realmente asociado con una posición en particular a lo largo de la cadena, y algunas posiciones podrían ser asignadas a más de un bit (a esto se le llama *sobre-especificación*) mientras que otras podrían no ser asignadas a ninguno (a esto se le llama *sub-especificación*). Consideremos, por ejemplo, a las dos cadenas mostradas en la Figura III-12, las cuales constituyen cromosomas válidos para un AG desordenado (estamos suponiendo cromosomas de 4 bits). La notación adoptada en este ejemplo usa paréntesis para identificar a cada gene, el cual se define como un par consistente de su posición a lo largo de la cadena (el primer valor) y el valor del bit en esa posición (estamos suponiendo un alfabeto binario). En el primer caso, la primera y la cuarta posición no están especificadas, y la segunda y la tercera están especificadas dos veces. Para lidiar con la sobre especificación pueden definirse algunas reglas determinísticas muy sencillas. Por ejemplo, podemos usar sólo la primera definición de izquierda a derecha para una cierta posición. Sin embargo, para la sub-especificación tenemos que hacer algo más complicado, porque una cadena sub-especificada realmente representa a un "esquema candidato" en vez de un cromosoma completo. Por ejemplo, la primera cadena de las antes descritas representa al esquema *10* (el * significa "no me importa"). Para calcular la aptitud de una cadena sub-especificada, podemos usar un explorador local del tipo "escalando la colina" que nos permita localizar el óptimo local y la información obtenida la podemos utilizar para reemplazar a los "no me importa" del esquema. A esta

técnica se le denomina "plantillas competitivas"[22]. Los AGs desordenados operan en 2 fases[23]: la "fase primordial" y la "fase yuxtaposicional". En la primera, se generan esquemas cortos que sirven como los bloques constructores en la fase yuxtaposicional en la cual éstos se combinan. El problema es cómo decidir qué tan largos deben ser estos esquemas "cortos". Si son demasiado cortos, pueden no contener suficiente material genético como para resolver el problema deseado; si son demasiado largos, la técnica puede volverse impráctica debido a la "maldición de la dimensionalidad" (tendríamos que generar y evaluar demasiados cromosomas).

Durante la fase primordial generamos entonces estos esquemas cortos y evaluamos sus aptitudes. Después de eso, aplicamos sólo selección a la población (sin cruce o mutación) para propagar los buenos bloques constructores, y se borra la mitad de la población a intervalos regulares. Después de un cierto número (predefinido) de generaciones, terminamos la fase primordial y entramos a la fase yuxtaposicional. A partir de este punto, el tamaño de la población permanecerá fijo, y usaremos selección y dos operadores especiales llamados "corte" y "unión". El operador de corte simplemente remueve una porción del cromosoma, mientras que el de unión junta dos segmentos cromosómicos.

Debido a la naturaleza del AG desordenado, las cadenas producidas por los operadores de corte y unión siempre serán válidas. Si los bloques constructores producidos en la fase primordial acarrean suficiente información, entonces el GA desordenado será capaz de arribar al óptimo global aunque el problema tenga decepción. Aunque es sin duda muy prometedor, los inconvenientes prácticos del AG desordenado han impedido su uso extendido, y actualmente se reportan relativamente pocas aplicaciones.

---

[22] Ídem anterior

[23] Ídem anterior

2.3.3. TÉCNICAS DE SELECCIÓN

Para aplicar los operadores genéticos tendremos que seleccionar un subconjunto de la población. Algunas de las técnicas que disponemos son:

2.3.3.1. Selección directa

Toma elementos de acuerdo a un criterio objetivo, como son «los x mejores», «los x peores»... los del tipo «el cuarto individuo a partir del último escogido» son empleados con mucha frecuencia cuando se quieren seleccionar dos individuos distintos, y se selecciona el primero por un método aleatorio o estocástico.

2.3.3.2. Selección aleatoria:

Puede ser realizado por selección equiprobable o selección estocástica.

2.3.3.3. Selección Equiprobable

Es una técnica de selección aleatoria donde todos tienen la misma probabilidad de ser escogidos.

2.3.3.4. Selección estocástica

Es otra técnica de selección aleatoria. La probabilidad de que un individuo sea escogido depende de una heurística. Los distintos procedimientos estocásticos son:

Selección por Sorteo

> Cada individuo de la población tiene asignado un rango proporcional - o inversamente proporcional- a su capacidad. Se escoge un número aleatorio dentro del rango global, y el escogido es aquel que tenga dicho número dentro de su rango. La probabilidad de ser escogido es proporcional/inversamente proporcional al grado de adaptación del individuo.

Selección por escaños

Se divide el rango del número aleatorio en un número predeterminado de escaños. Los escaños se reparten de acuerdo con la ley d'Hont, tomando como «puntuación» para repartir los escaños el grado de capacidad. Observamos que es más probable escoger un elemento de baja probabilidad por este método que en el de selección por sorteo.

Selección por restos estocásticos

Igual que el método de selección de escaños, sólo que los escaños no asignados directamente -es decir, aquellos en que se aplica directamente la ley d'Hont- se asignan de forma aleatoria. La probabilidad de escoger un elemento de muy baja probabilidad es más alta que en el de selección por escaños.

Por ruleta

Definimos un rango con las características de la selección por sorteo. El número al azar será un número aleatorio forzosamente menor que el tamaño del rango. El elemento escogido será aquel en cuyo rango esté el número resultante de sumar el número aleatorio con el resultado total que sirvió para escoger el elemento anterior. El comportamiento es similar al de una ruleta, donde se define un avance cada tirada a partir de la posición actual. Tiene la ventaja de que no es posible escoger dos veces consecutivas el mismo elemento, y que puede ser forzado a que sea alta la probabilidad de que no sean elementos próximos en la población -esto último no es una ventaja de por sí; salvo que algunos de los otros operadores genéticos emplee un método de selección directa basado en la posición relativa de los individuos de la población-. En la bibliografía más antigua se emplea este término para definir lo que aquí hemos definido como selección por sorteo.

Por torneo

Escoge un subconjunto de individuos de acuerdo con una de las técnicas anteriores -habitualmente, aleatoria o estocástica- y de entre ellos selecciona el más adecuado por otra técnica -habitualmente, determinística de tipo «el mejor» o «el peor»-. Esta técnica tiene la ventaja de que permite un cierto grado de elitismo -el mejor nunca va a morir, y los mejores tienen más probabilidad de reproducirse y de emigrar que los peores- pero sin producir una convergencia genética prematura, si la población es, al menos, un orden de magnitud superior al del número de elementos involucrados en el torneo. En caso de que la diferencia sea menor no hemos observado mucha diferencia entre emplear el torneo o no. La selección por torneo ha sido la técnica empleada en nuestro algoritmo para decidir tanto el padre -en unión con el criterio «el mejor»-, como quién va a emigrar -en unión con el criterio «el mejor»- y quién va a morir -en unión con el criterio «el peor».

2.3.3.5. Elitismo

Se denomina elitismo al proceso por el cual determinados elementos con una adaptación especialmente buena tienen determinados privilegios -nunca mueren, proporción alta de pasos en que se reproduce uno de la élite con otro al azar. Sin embargo, en fases de inicio es peligroso, ya que puede producirse que una *élite* de súper individuos acabe con la diversidad genética del problema. Para ello, lo que se puede hacer es escalar la función de adaptación en las primeras fases del algoritmo -de forma que las diferencias entre la élite y el pueblo sean menores- y súper escalar la función de adaptación al final del algoritmo, para evitar un bloqueo de la convergencia.

2.3.4. TÉCNICAS DE CRUCE

Se denomina *técnica de cruce* a la forma de calcular el genoma del nuevo individuo en función del genoma del padre y de la madre. El operador de cruce es fuertemente responsable de las propiedades del algoritmo genético, y determinará en gran medida la evolución de la población.

Existen gran cantidad de técnicas de cruce. Las técnicas básicas son:

2.3.4.1. Cruce básico

Se selecciona un punto al azar de la cadena. La parte anterior del punto es copiada del genoma del padre y la posterior del de la madre.

2.3.4.2. Cruce multipunto

Igual que el cruce básico, sólo que estableciendo más de un punto de cruce.

2.3.4.3. Cruce segmentado

Existe una probabilidad de que un cromosoma sea punto de un cruce. Conforme se va formando la nueva cadena del descendiente, para cada gen, se verifica si ahí se va producir un cruce.

2.3.4.4. Cruce uniforme

Para cada gen de la cadena del descendiente existe una probabilidad de que el gen pertenezca al padre, y otra de que pertenezca a la madre.

2.3.4.5. Cruces para permutación

Existe una familia de cruces específicas para los problemas de permutación, siendo algunos de ellos:

2.3.4.6. Cruce de mapeamiento parcial

Toma una subsecuencia del genoma del padre y procura preservar el orden absoluto de los fenotipos -es decir, orden y posición en el genoma- del resto del genoma lo más parecido posible de la madre. Aparece también en la bibliografía como PMX.

2.3.4.7. Cruce de orden

Toma una sub secuencia del genoma del padre y procura preservar el orden relativo de los fenotipos del resto del genoma lo más parecido posible de la madre. Lo podemos encontrar en la bibliografía como OX.

2.3.4.8. Cruce de ciclo

Tomamos el primer gen del genoma del padre, poniéndolo en la primera posición del hijo, y el primer gen del genoma de la madre, poniéndolo dentro del genoma del hijo en la posición que ocupe en el genoma del padre. El fenotipo que está en la posición que ocupa el gen del genoma del padre igual al primer gen del genoma de la madre se va a colocar en la posición que ocupe en el genoma del padre, y así hasta rellenar el genoma del hijo. Este método también es conocido en la bibliografía como CX.

Es una buena idea que, tanto la codificación como la técnica de cruce, se hagan de manera que las características buenas se hereden; o, al menos, no sea mucho peor que el peor de los padres. En problemas en los que, por ejemplo, la adaptación es función de los pares de genes colaterales, el resultante del cruce uniforme tiene una adaptación completamente aleatoria.

2.3.5. TÉCNICAS DE MUTACIÓN

Se define *mutación* como una variación de las informaciones contenidas en el código genético -habitualmente, un cambio de un gen a otro producido por algún factor exterior al algoritmo genético-. En Biología se definen dos tipos de mutaciones: las generativas, que se heredan y las somáticas, que no se heredan. En los algoritmos genéticos sólo nos serán interesantes las mutaciones generativas.

Algunas de las razones que pueden motivar a incorporar mutaciones en nuestro algoritmo son:

Desbloqueo del algoritmo. Si el algoritmo se bloqueó en un mínimo parcial, una mutación puede sacarlo al incorporar nuevos fenotipos de otras zonas del espacio.

Acabar con poblaciones degeneradas. Puede ocurrir que, bien por haber un cuasi-mínimo, bien porque en pasos iniciales apareció un individuo demasiado bueno que acabó con la diversidad genética, la población tenga los mismos fenotipos. A priori se pueden plantear algunas soluciones, como el

escalamiento de la función de capacidad; mas, si ya se ha llegado a una población degenerada, es preciso que las mutaciones introduzcan nuevos genomas. Como analizaremos en el operador de cruce, ésto se hace implícitamente en cada cruce.

Incrementar el número de saltos evolutivos. Los saltos evolutivos -aparición de un fenotipo especialmente valioso, o, dicho de otra forma, salida de un mínimo local- son muy poco probables en un genético *puro* para un problema genérico. La mutación permite explorar nuevos sub espacios de soluciones, por lo que, si el sub espacio es bueno en términos de capacidad, se producirá un salto evolutivo después de la mutación que se expandirá de forma exponencial por la población.

Enriquecer la diversidad genética. Es un caso más *suave* que el de una población degenerada -por ejemplo, que la población tenga una diversidad genética pobre-, la mutación es un mecanismo de prevención de las poblaciones degeneradas.

Sin embargo, si la tasa de mutación es excesivamente alta tendremos la ya conocida deriva genética. Una estrategia muy empleada es una tasa de mutación alta al inicio del algoritmo, para aumentar la diversidad genética, y una tasa de mutación baja al final del algoritmo, para conseguir que converja.

Existen varias técnicas distintas de mutación. Algunas de éstas son:

2.3.5.1. Mutación de bit

Existe una única probabilidad de que se produzca una mutación de algún bit. De producirse, el algoritmo toma aleatoriamente un bit, y lo invierte.

2.3.5.2. Mutación multibit

Cada bit tiene una probabilidad de mutarse o no, que es calculada en cada pasada del operador de mutación multibit.

2.3.5.3. *Mutación de gen*

Igual que la mutación de bit, solamente que, en vez de cambiar un bit, cambia un gen completo. Puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatório nuevo.

2.3.5.4. *Mutación multigen*

Igual que la mutación de multibit, solamente que, en vez de cambiar un conjunto de bits, cambia un conjunto de genes. Puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo. Esta mutación es la que se produce implícitamente en nuestra implementación de cruce.

2.3.5.5. *Mutación de intercambio*

Existe una probabilidad de que se produzca una mutación. De producirse, toma dos bits/genes aleatoriamente y los intercambia.

2.3.5.6. *Mutación de barajado*

Existe una probabilidad de que se produzca una mutación. De producirse, toma dos bits o dos genes aleatoriamente y baraja de forma aleatoria los bits -o genes, según hubiéramos escogido- comprendidos entre los dos. La bibliografía en inglés emplea el término *scramble mutation*, en mención a un juego de mesa, mas la operación que realizamos realmente es un barajado entre los dos genes.

2.3.6. *MANEJO DE RESTRICCIONES*

En la práctica normalmente tenemos problemas con restricciones de diferentes tipos (igualdad, desigualdad, lineales y no lineales), tales como:

$$Minimizar \quad f(x)$$

Sujeto a:

$$g(i) \leq 0 \quad i = 1,2,...,n$$

Pero el algoritmo genético opera como una técnica de optimización sin restricciones, por lo que tenemos que diseñar algún mecanismo que permita

incorporar la información pertinente sobre la violación de restricciones en la función de capacidad.

Esta sección muestra diferentes esquemas para hacer esto a través de Funciones de Penalización que son la técnica más común de incorporación de restricciones en la función de capacidad. La idea básica es extender el dominio de la función de aptitud usando la ecuación (3-1).

$$capacidad_i(X) = f_i(X) \pm Q_i \qquad (3\text{-}1)$$

donde:

$$Q_i = c \times \sum_{i=1}^{n} g_i(X)^2 \qquad (3\text{-}2)$$

para todas las restricciones violadas. En esta expresión, *c* es un factor de penalización definido por el usuario.

Hay al menos 3 formas de penalizar a un individuo de acuerdo a su violación de las restricciones según Richardson[24]:

Puede penalizársele simplemente por no ser factible, sin usar ninguna información sobre qué tan cerca se encuentra de la región factible.

Puede usarse la "cantidad" de infactibilidad de un individuo para determinar su penalización correspondiente.

Puede usarse el esfuerzo de "reparar" al individuo (o sea, el costo de hacerlo factible) como parte de la penalización.

Richardson et al. definieron algunas de las reglas básicas para diseñar una función de penalización:

---

[24] Richardson J. T., Palmer M. R., Liepins G., y Hilliard Mike. Some Guidelines for Genetic Algorithms with Penalty Functions, editor: J. David Schaffer, Procedente de: Third International Conference on Genetic Algorithms, pages 191-197, George Mason University, 1989. Morgan Kaufmann Publishers.

Las penalizaciones que son funciones de la distancia a la zona factible son mejores que aquellas que son sólo funciones del número de restricciones violadas.

Para un problema con pocas variables y pocas restricciones, una penalización que sea sólo función del número de restricciones violadas no producirá ninguna solución factible.

Pueden construirse buenos factores de penalización a partir de 2 factores: el costo de cumplimiento máximo y el costo de cumplimiento esperado. El primero de ellos se refiere al costo de hacer factible a una solución infactible.

Las penalizaciones deben estar cerca del costo de cumplimiento esperado, pero no deben caer frecuentemente por debajo de él. Entre más preciso sea el factor de penalización, mejores resultarán las soluciones producidas. Cuando una penalización frecuentemente subestime el costo de cumplimiento, el proceso de búsqueda fallaría.

Existen, sin embargo, varios problemas para definir una función de penalización.

No es obvio combinar los 2 factores de los que habla Richardson en una función de penalización.

Definir los valores óptimos del factor de penalización es una tarea virtualmente imposible a menos que conozcamos el problema a fondo, en cuyo caso puede diseñarse una función de penalización a la medida, pero hacerlo resultarla de cualquier forma innecesaria ya que el óptimo podría determinarse por métodos analíticos exactos.

El costo del cumplimiento esperado normalmente tiene que estimarse mediante métodos alternativos (por ejemplo, estimando el grado de violación de las restricciones) que pueden ser difíciles de implementar.

A continuación revisaremos rápidamente diversas variantes de la función de penalización que se han propuesto.

2.3.6.1. *Pena de Muerte*

En las estrategias evolutivas ha sido popular una técnica a la que se le conoce como "pena de muerte", y que consiste en asignar una aptitud de cero a un individuo que no sea factible, y tomar el valor de la función de aptitud para los que sí lo sean[25].

2.3.6.2. *Penalizaciones estáticas*

En esta técnica, introducida por Homaifar, Lai y Qi[26], la idea es definir varios niveles de violación y elegir un coeficiente de violación para cada uno de ellos, de manera que el coeficiente de penalización se incremente conforme alcanzamos niveles más altos de violación de las restricciones.

Un individuo se evalúa utilizando la ecuación (3-3).

$$capacidad_i = f_i(X) + \sum_{j=1}^{n} R_{k,j} g_j(X) \qquad (3\text{-}3)$$

donde, $R_{i,j}$ son los coeficientes de penalización utilizados y $k = 1, 2, 3, \ldots, l$ siendo l los niveles de violación definidos por el usuario.

2.3.6.3. *Penalizaciones Dinámicas*

Joines y Houck[27] propusieron una técnica en la que los factores de penalización cambian con respecto al tiempo.

Los individuos de la generación t se evalúan de acuerdo a la ecuación (3-4).

---

[25] Schwefel H. P.,Artículo, Numerical Optimization of Computer Models. Wiley, Chichester, UK, 1981.

[26] Homaifar A., Lai S. H. Y., y Qi X., Constrained Optimization via Genetic Algorithms.Simulation, 62(4):242-254, 1994.

[27] Joines J. y Houck C., On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with Gas, editor: David Fogel, Procedente de: The First IEEE Conference on Evolutionary Computation, pages 579-584, Orlando, Florida, 1994. IEEE Press.

$$capacidad_i(X) = f_i(X) + (C \times t)^{\alpha} \sum_{j=1}^{n} |g_i(X)|^{\beta} \qquad (3\text{-}4)$$

donde C, α, β son constantes definidas por el usuario. Los valores sugeridos por los autores son $C = 0.5$, $\alpha=1$ y $\beta=1$ ó 2

2.3.6.4. *Uso de recocido simulado*

Michalewicz y Attia[28] consideraron una técnica para manejo de restricciones que usa el concepto de recocido simulado: los coeficientes de penalización se cambian cada cierto número de generaciones (cuando el algoritmo ha quedado atrapado en un óptimo local).

Los individuos se evalúan usando la ecuación (3-5).

$$capacidad_i(X) = f_i(X) + \frac{1}{2\tau} \sum_{j=1}^{n} g_j(X)^2 \qquad (3\text{-}5)$$

donde: τ representa la hora de enfriamiento y es una función que debe ser definida por el usuario.

Michalewicz y Attia sugieren usar: $\tau_0 = 1$ y $\tau_f = 0.000001$, con incrementos $\tau_{i+1} = 0.1 \times \tau_i$.

2.3.6.5. *Penalizaciones Adaptativas*

Bean y Hadj-Alouane[29] desarrollaron una técnica para adaptar penalizaciones en base a un proceso de retroalimentación del ambiente durante la corrida de un algoritmo genético.

Cada individuo es evaluado usando la ecuación (3-6).

---

[28] Michalewicz Z. and Attia N. F., Evolutionary Optimization of Constrained Problems,.Procedente de: 3rd Annual Conference on Evolutionary Programming, pages 98-108. World Scientific, 1994.

[29]. Bean J. C. Genetics and random keys for sequencing and optimization. ORSA Journal on Computing, 6(2):154-160, 1994.

$$capacidad_i(X) = f_i(x) + \lambda(t)\sum_{j=1}^{n} g_j(X)^2 \qquad (3\text{-}6)$$

donde: $\lambda(t)$ se actualiza cada generación usando las reglas de la ecuación (3-7).

$$\lambda(t) = \begin{cases} \left(\frac{1}{\beta_1}\right)\lambda(t) & si\ caso\#1 \\ \beta_2\lambda(t) & si\ caso\#2 \\ \lambda(t) & si\ caso\#3 \end{cases} \qquad (3\text{-}7)$$

donde β_1, $\beta_2 > 1$ y con valores diferentes (para evitar ciclos).

El caso # 1 ocurre cuando el mejor individuo en las últimas *k* generaciones fue siempre factible. El caso # 2 ocurre cuando dicho individuo no fue nunca factible.

Esta técnica lo que hace entonces es disminuir la penalización cuando el mejor individuo resulta consistentemente factible y aumentar la penalización cuando resulta infactible. Si estos cambios son intermitentes (es decir, el mejor individuo es a veces factible y a veces no), entonces la penalización no se cambia.

Obviamente, es necesario definir el valor inicial λ_0.

2.3.6.6. Algoritmo genético segregado

Esta técnica fue propuesta por Le Richel[30] y consiste en usar 2 funciones de penalización en vez de una, empleando para ello dos poblaciones. Estas 2 funciones intentan balancear las penalizaciones moderadas con las fuertes.

Inicialmente, se divide la población en 2 grupos, de manera que los individuos de cada grupo se evalúan usando un factor de penalización distinto.

---

[30] Le Riche R. G., Knopf-Lenoir C., y Haftka R. T., A Segregated Genetic Algorithm for Constrained Structural Optimization. Editor: Larry J. Eshelman, Procedente de: Sixth International Conference on Genetic Algorithms, pages 558-565, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

Después se elige a los mejores individuos de cada grupo para ser padres de la siguiente generación, lo que hace que se combinen individuos factibles con infactibles (si se usa un factor grande y otro pequeño), manteniendo la diversidad y evitando quedar atrapado en máximos locales.

En la implementación original de esta técnica se utilizaron jerarquías lineales para decrementar la presión de selección y se le aplicó a un problema de diseño de elementos compuestos con gran éxito.

2.3.6.7. Penalización con base en la factibilidad

Esta técnica fue propuesta por Deb[31] y consiste en evaluar un individuo de acuerdo a la ecuación (3-8).

$$capacidad_i(X) = \begin{cases} f_i(X) & Si\ la\ solución\ es\ factible \\ f_{peor} + \sum_{j=1}^{n} g_j(X) & de\ lo\ contrario \end{cases} \qquad (3\text{-}8)$$

Deb [66] usa torneo binario aplicando las siguientes reglas:

- Una solución factible siempre es preferida sobre una no factible.
- Entre 2 soluciones factibles, la que tenga mejor valor de su función objetivo es seleccionada.
- Entre 2 soluciones no factibles, la que viole el menor número de restricciones es elegida.

2.3.7. PRINCIPIOS DE COMPONENTES

2.3.7.1. Objetivos

El desarrollo basado en componentes es una aplicación de la técnica de divide y conquistaras para manejar la complejidad. La diferencia principal con los métodos estructurados es principalmente que el análisis y diseño es rea-

---

[31] Deb K., An Efficient Constraint Handling Method for Genetic Algorithms, Computer Methods in Applied Mechanics and Engineering, 1999. (in Press).

lizado dentro del mismo paradigma que la implementación. Esta implementación queda relegada a un segundo plano, siendo importante dar una solución lógica al problema, previo a su codificación. Este principio fue utilizado en el paradigma de orientación a objetos, el hecho de combinar operaciones e información en una misma unidad, y de contar con técnicas de modelado dentro del mismo paradigma, hizo que la orientación a objetos tuviera un éxito importante. El principal objetivo que se persiguió con la introducción de este paradigma fue la reutilización. A pesar de contar con técnicas de buenas prácticas de diseño, como los patrones, no es sencillo mantener las unidades de software (i.e. clases) con el nivel de acoplamiento y cohesión deseables. La necesidad de reusar una clase implica llevar consigo otros artefactos que en un principio pueden no ser necesarios para el nuevo escenario donde se quiere reaprovechar la clase.

Por esta razón, el paradigma de componentes no se focaliza en el principio de reutilización sino que ataca principalmente la mantenibilidad. La reutilización es un objetivo admirable pero no es sencillo de obtener. Bajo el enfoque de componentes se busca construir para el cambio. Los sistemas actuales cambian sus requerimientos incluso cuando el sistema ya está en producción. El principal objetivo de un componente no es la reutilización sino que sea fácilmente reemplazable. El hecho de ser reemplazable implica que una nueva implementación de un componente pueda ser utilizada en lugar de una implementación anterior sin afectar el funcionamiento del resto de los componentes. Nuevas implementaciones pueden por ejemplo mejorar su performance o proveer nuevos servicios; el único requerimiento es que provea los mismos servicios provistos por la implementación anterior.

El enfoque de componentes enfatiza en la arquitectura del sistema y en la capacidad de manejar al sistema completo, de forma tal que es en base a esa arquitectura que se evalúa el impacto del cambio y no en base a información local. Las decisiones internas a los componentes son un objetivo secundario, siendo lo primordial su interacción con el resto de los componen-

tes del sistema. El enfoque propone concentrarse en el todo y no en las partes.

2.3.7.2. Principios

Los componentes son unidades de software que se rigen por ciertos principios. Éstos son los mismos que los presentes en el paradigma de orientación a objetos: unificación de datos y comportamiento, identidad y encapsulamiento. A estos principios se le agrega el del uso obligatorio de interfaces. Cada cliente de un componente depende exclusivamente de la especificación del componente y no de su implementación. Esta importante separación es la clave para reducir el acoplamiento y el buen manejo de las dependencias.

La especificación de un componente está formada por un conjunto de interfaces que describen el comportamiento del componente. Las interfaces describen este comportamiento en función de un modelo de información, el cual es una proyección del modelo de información del propio componente. Las dependencias entre componentes son dependencias de uso de interfaces, no son dependencias directas sobre el componente. Muchas implementaciones pueden realizar una especificación de componente permitiendo de esta forma contar con la propiedad de ser reemplazable.

2.3.8. METODOLOGÍA DE DESARROLLO BASADA EN COMPONENTES

La metodología propuesta está basada en casos de uso y está centrada en la arquitectura. Estos lineamientos generales, propuestos por el Rational Unified Proccess, encajan fuertemente con los objetivos de nuestro paradigma.

2.3.8.1. Arquitectura

donde f_{peor} es el valor de la función objetivo de la peor solución factible de la población. Si no hay ninguna solución factible en la población, entonces f_{peor} se hace igual a cero.

El término 'arquitectura' es heredado de otras disciplinas de la ciencia. Se entiende por arquitectura a un conjunto de piezas de distintos tipos, que encajan entre sí y cumplen una función determinada. La arquitectura presenta además el impacto del cambio de una de las piezas. Dentro del paradigma de componentes, las piezas (o building blocks) son los componentes. La arquitectura de componentes dirá con que tipos de componentes y en qué relación de dependencia se encuentran.

La metodología busca utilizar el paradigma de componentes a sistemas empresariales de gran porte. Para ello consideramos arquitecturas distribuidas, en múltiples capas, que incorporan fuentes de datos heterogéneos, sistemas legados y paquetes adquiridos.

El estilo de arquitectura en capas es aplicable a este tipo de sistemas. Cada capa sugiere un tipo diferente de componentes, e indica el rol que juegan los componentes que residan en ella. La arquitectura propuesta se presenta en Figura II – 13.

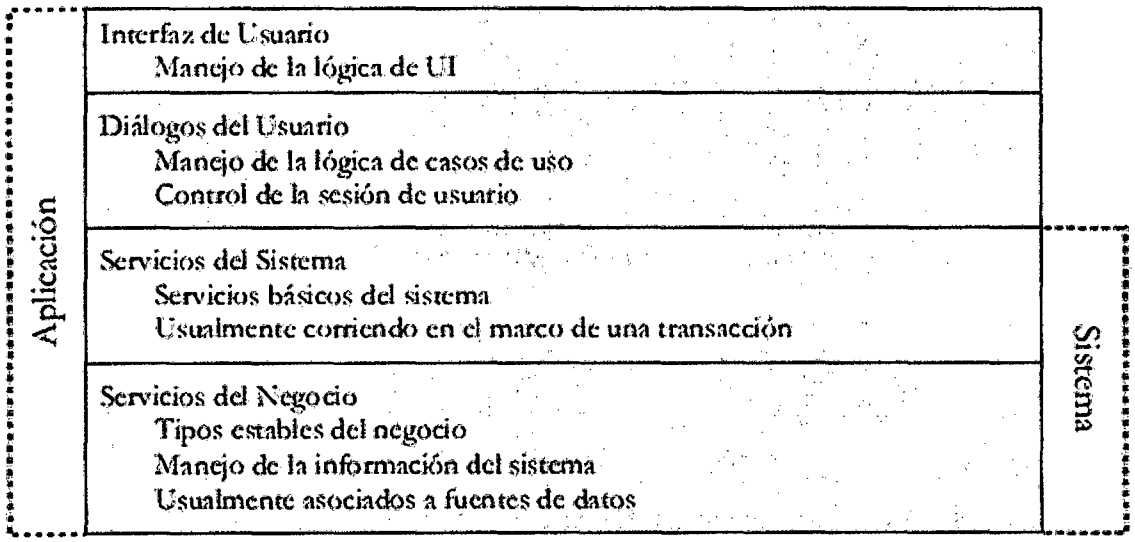


Figura II-13. Arquitectura.

El enfoque metodológico se centra en aquellas capas que representan las funcionalidades del sistema, a saber, la capa de Servicios del Sistema y la capa de Servicios del Negocio.

La definición de la arquitectura de componentes cubre aspectos únicamente lógicos y es totalmente independiente de la tecnología con la cual se implementarán los componentes y sobre la cual se hará el despliegue del sistema. Esta vista lógica nos permite medir el nivel de acoplamiento del sistema y razonar sobre los efectos de modificar o reemplazar un componente. La independencia de la tecnología nos permite abstraernos de los tecnicismos de éstas así como elegir la más apta dependiendo del sistema que se esté desarrollando.

2.3.8.2. Metodología

La metodología propuesta abarca las tres primeras fases propuestas en el RUP, y propone actividades correspondientes solamente a las disciplinas del Modelado del Negocio, Requerimientos y Análisis & Diseño.

Recordar que nuestro enfoque es independiente de la tecnología por lo cual no son consideradas las disciplinas de Implementación, Prueba y Despliegue, y tampoco la fase de Transición. Asimismo, este enfoque refiere a actividades exclusivamente de desarrollo de software y no a actividades de gestión y gerenciamiento del mismo. Así, las otras disciplinas del RUP tampoco fueron consideradas.

Para una mayor aplicabilidad del enfoque se han reformulado los workflows que ocurren en la metodología. Los mismos no corresponden directamente a cada disciplina sino que corresponden a las fases. Cada workflow indica claramente el lugar donde ocurren las iteraciones.

Las actividades presentes en el workflow de la fase Incepción refieren principalmente a las actividades de la disciplina de Modelamiento de Negocio propuesta por RUP (Figura II – 14).

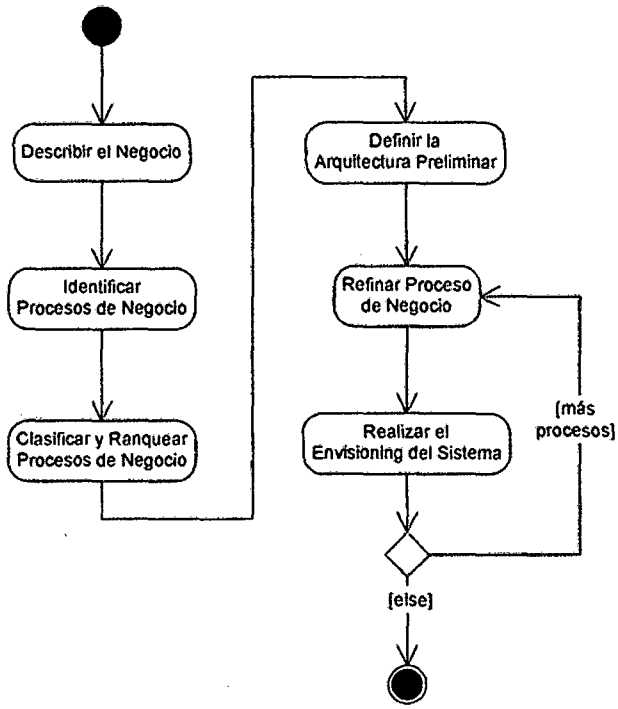


Figura II-14. Workflow de la Fase de Incepción.

El workflow de la fase de Elaboración ataca los procesos en el orden dado por el ranqueo de procesos realizado en la fase anterior. No es necesario atacar todos los procesos, sino aquellos críticos desde el punto de vista de la arquitectura. Las actividades presentes en este workflow son similares a las propuestas en el RUP.. (Figura II – 15).

El workflow para la fase de Construcción es análoga al de la fase de Elaboración. Además, dado que en la fase Construcción la arquitectura está lo suficientemente estable, una nueva actividad debe llevarse adelante. La misma lleva el nombre de Especificación de Componentes.

La siguiente sección presentará en más detalle cada una de las actividades particulares al enfoque aquí presentado: Identificación de Componentes, Interacción de Componentes y Especificación de Componentes.

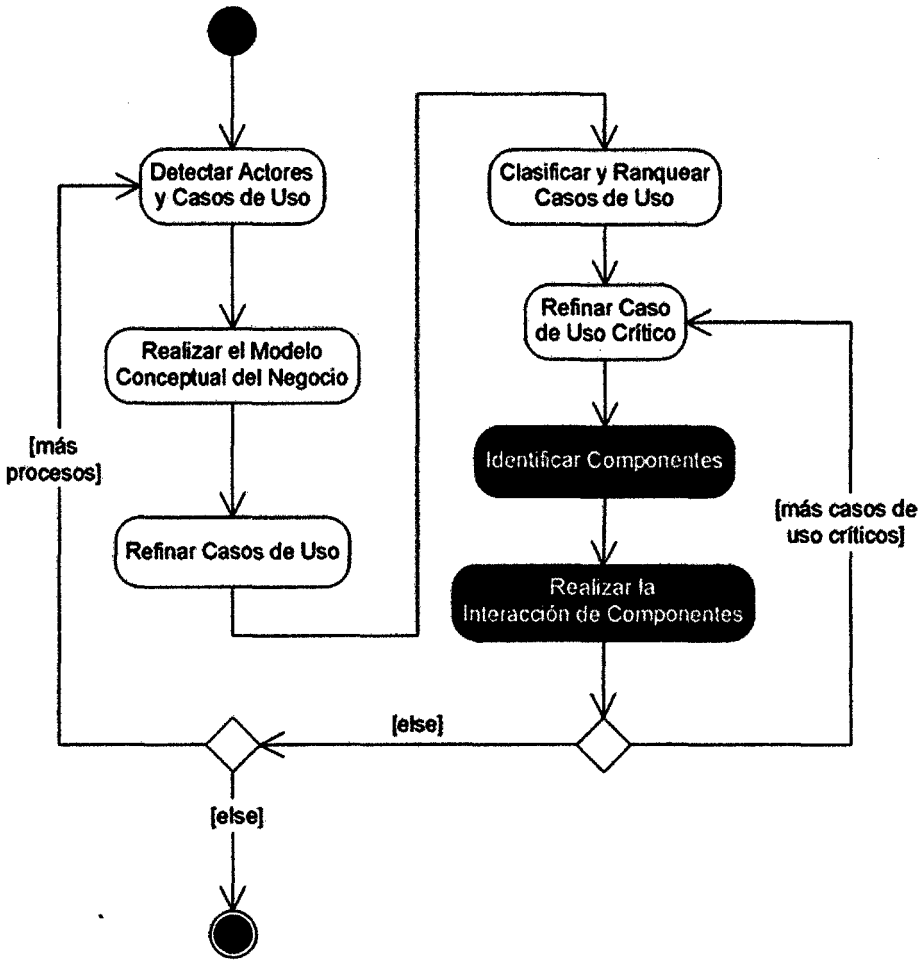


Figura II-15. Workflow de la Fase de Elaboración.

2.3.9. ACTIVIDADES PARTICULARES AL ENFOQUE BASADO EN COMPONENTES

2.3.9.1. Identificación de Componentes

Esta etapa identifica a partir de los artefactos generados en las actividades anteriores el conjunto de interfaces y especificaciones de componentes que poblaran la arquitectura.

Esta actividad tiene como objetivos:

- Crear un conjunto inicial de interfaces y especificaciones de componentes, tanto a nivel de componentes de sistema como de componentes de negocio.

- Producir el modelo de tipos del negocio inicial, partiendo del modelo conceptual preliminar.
- Presentar las interfaces y especificaciones de componentes en una arquitectura de componentes inicial, decidiendo de qué forma se agrupan las interfaces en especificaciones de componentes.

La Figura II – 16 muestra las tareas que se proponen para llevar adelante esta actividad.

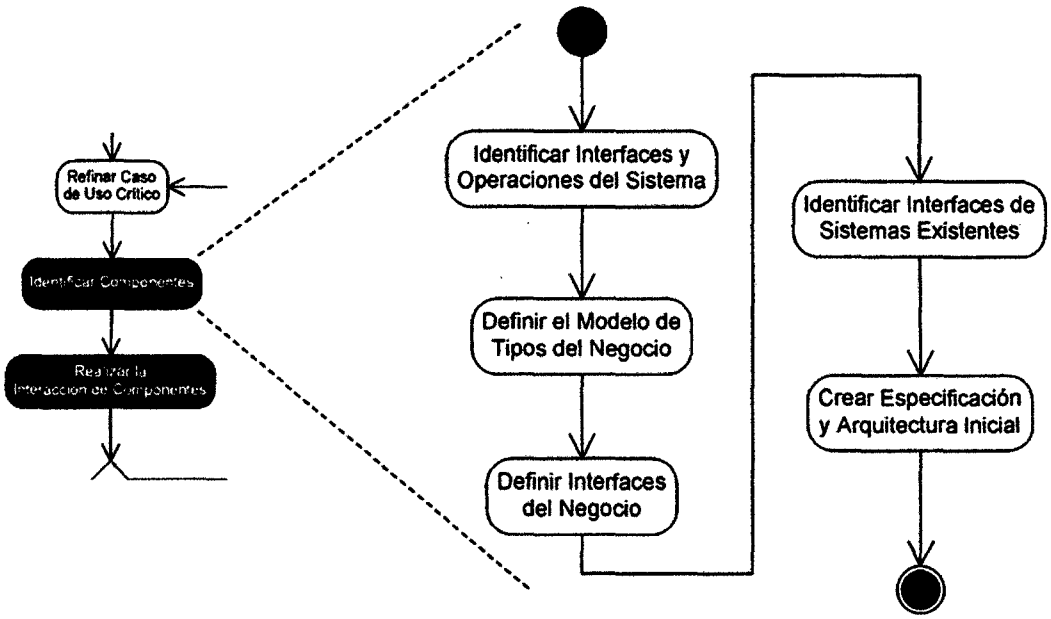


Figura II-16. Tareas que componen la actividad Identificar Componentes.

2.3.9.2. Interacción de Componentes

En esta etapa se decide cómo trabajarán juntos los componentes detectados en la etapa anterior de forma de satisfacer las funcionalidades deseadas. Los objetivos particulares de esta actividad son:

- Refinar las definiciones de las interfaces de sistema.
- Definir las interacciones entre los componentes identificando operaciones en las interfaces de los componentes de negocio y determinando las dependencias entre componentes.
- Definir políticas de manejo de integridad referencial.

Las tareas a realizarse en esta actividad se presentan en la Figura II-17.

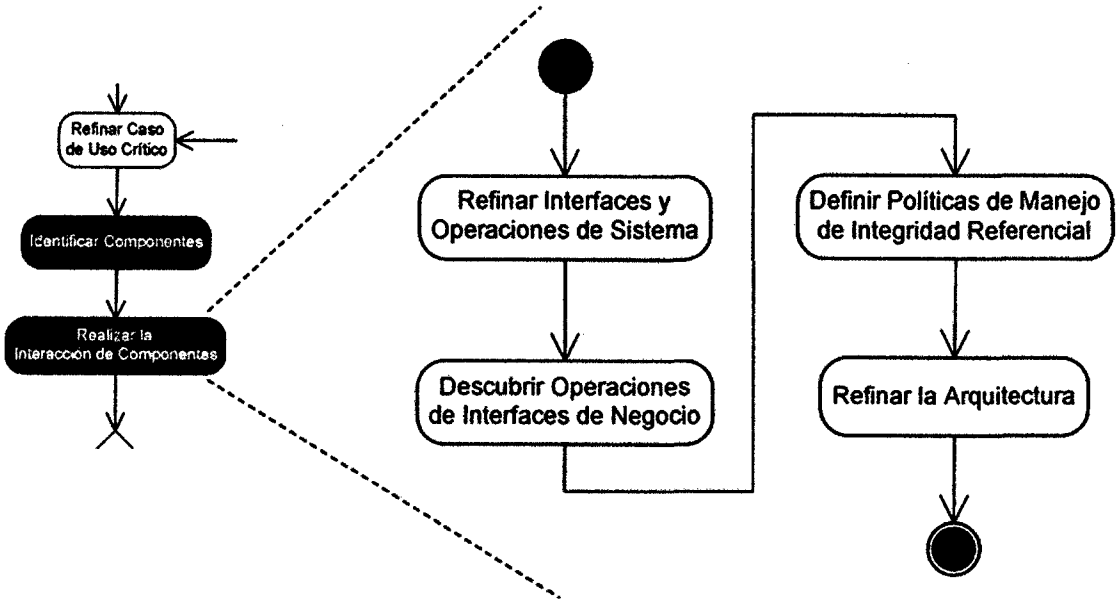


Figura II-17. Tareas que componen la actividad Interacción de Componentes.

2.3.9.3. Especificación de Componentes

Como se mencionó antes, esta actividad es realizada una vez que la arquitectura e interfaces de los componentes estén estables. La especificación de los componentes es una actividad fundamental la cual favorece fuertemente a la reemplazabilidad así como posibilita el reuso de componentes en futuros proyectos.

Los objetivos de esta actividad son:

- Definir el modelo de información de cada interfaz; este modelo representa una vista abstracta de la información manejada por el componente.
- Especificar formalmente las operaciones de las interfaces; esta especificación se realiza con contratos de software.
- Capturar y documentar las restricciones entre los componentes.

La Figura II-18 presenta las tareas a realizar en esta actividad.

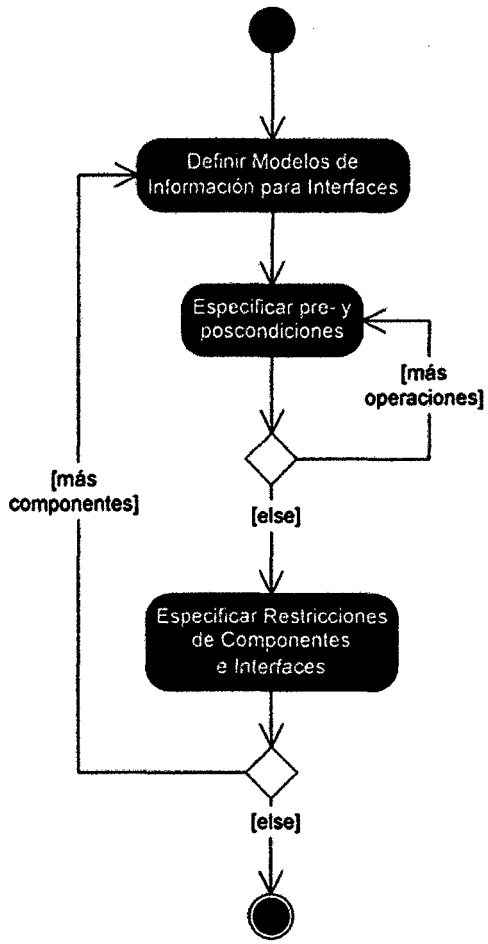


Figura II-18. Tareas que componen la actividad Especificación de Componentes.

2.3.10. RESORTES HELICOIDALES DE COMPRESIÓN

Un resorte[32] es un elemento activo que se utiliza para ejercer una fuerza o un torque y, al mismo tiempo, almacenar energía. La fuerza puede ser de empuje o de tracción lineal, o puede ser radial. Los resortes almacenan energía cuando se deflexionan y la regresan cuando se elimina la fuerza que provoca la deflexión.

Se clasifican de acuerdo al sentido y la naturaleza de la fuerza que ejercen cuando se deflexionan. Se pueden clasificar como de empuje, tracción, ra-

[32] Las descripciones y notaciones presentadas en este capítulo se basan en el libro "Diseño de elementos de máquinas" de Robert Mott, primera edición.

dial y torsión (tabla II-1) y la Figura II-19 muestra algunos de los diseños más comunes.

Tabla II-1. Tipos de resortes

Usos	Tipos de resortes
Empujar	Resortes de compresión helicoidal Resorte Belleville Resorte de torsión: fuerza que actúa en el extremo del brazo del torque Resorte plano, como cantiléver o resorte de hoja
Jalar	Resorte de extensión helicoidal Resorte de torsión: fuerza que actúa en el extremo del brazo de torque Resorte plano, como cantiléver o resorte de hoja Resorte de barra de torsión (caso especial del resorte de compresión) Resorte de fuerza constante
Radial	Resorte Garter, banda elastomérica, grapa de resorte
Torque	Resorte de torsión, resorte de potencia

Fuente: Norton R. L. Diseño de Elementos de Máquinas. México: Prentice Hall; 1995. Pág. 207

Los resortes helicoidales de compresión se fabrican, generalmente, de alambre redondo, enrollado en forma cilíndrica recta con un espaciamiento constante entre bobinas adyacentes. Puede utilizarse también alambre cuadrado o rectangular. La Figura II-20 muestra cuatro de las configuraciones más usadas en los extremos.

Sin una fuerza aplicada la longitud del resorte recibe el nombre de *longitud libre*. Si se aplica una fuerza de tal manera que las bobinas se compriman hasta que todas están en contacto entre sí, en ese momento la longitud es la mínima y se denomina *longitud comprimida*.

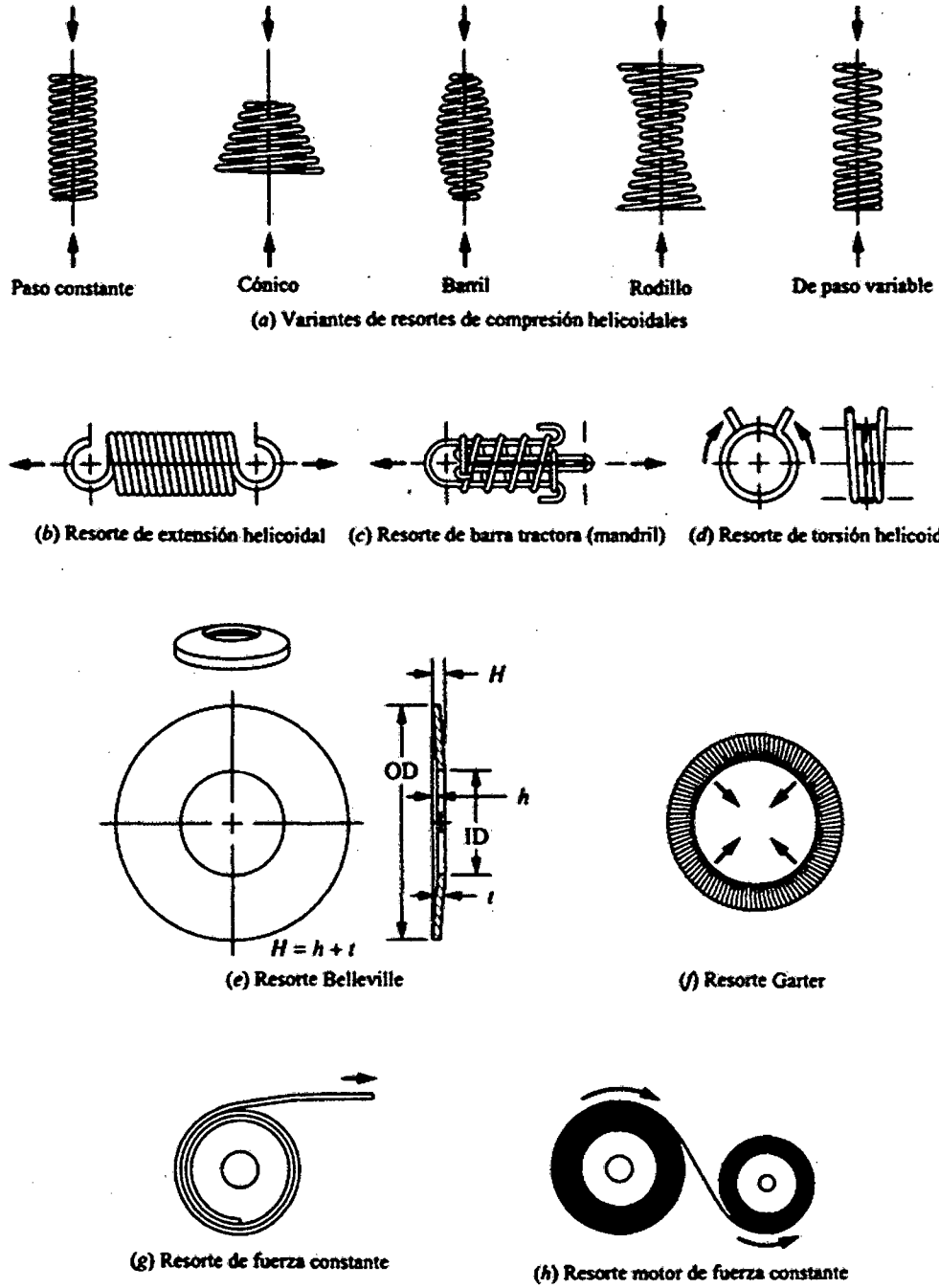


Figura II-19. Diferentes tipos de resortes.

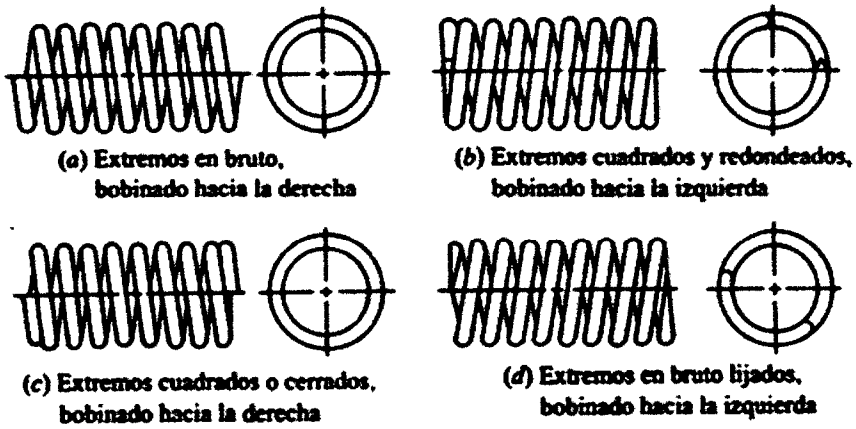


Figura II-20. Extremos en resortes helicoidales de compresión.

La forma más común de un resorte helicoidal de compresión es la de un alambre redondo enrollado en forma cilíndrica con un espaciamiento constante entre bobinas adyacentes. Además el resorte puede tener diferentes tratamientos para los extremos, como se ilustra en la Figura II-20.

En los resortes de tamaño mediano a grande, como los que se utilizan en maquinaria, en los extremos se tiene un tratamiento redondo y cuadrado para ofrecer una superficie plana para asentar el resorte. La bobina del extremo se junta a la bobina adyacente, cuadrada, y la superficie se posa hasta que por lo menos 270° de la última bobina entra en contacto con la superficie de apoyo. Con los resortes que se fabrican con alambres más delgados, menos de 0.020" o 0.50" aproximadamente, casi siempre son cuadrados y no se lijan. Es poco común que los extremos se lijen pero no a escuadra, o se dejan en bruto, se cortan a la medida después de bobinarse.

A continuación se definen las variables que se emplean para describir y analizar a los resortes helicoidales de compresión.

2.3.10.1. Diámetros

La Figura II-21 muestra la notación utilizada para hacer referencia a los diámetros característicos de los resortes helicoidales de compresión[33].

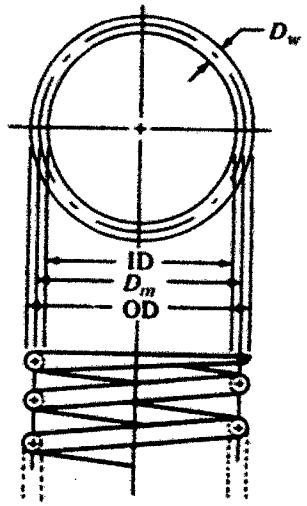


Figura II-21. Notación para resortes helicoidales de compresión.

OD = Diámetro exterior

ID = Diámetro interior

D_w = Diámetro del alambre.

D_m = Diámetro medio

Existen las siguientes relaciones entre estas medidas:

$$OD = D_m + D_w$$
$$ID = D_m - D_w$$

En general los alambres para resortes son de diferentes tipos de materiales y en diámetros estándar que abarcan un rango muy amplio. La tabla II-2 muestra los calibres estándar de alambre más comunes. En la mayor parte de los casos, excepto en el alambre para instrumentos musicales el tamaño del alambre disminuye a medida que se aumente el calibre. Para los alam-

---

[33] Moot, Robert; "Diseño de elementos de máquinas"; Segunda edición, Editorial Prentice Hall; México 1992, Pág. 210.

bres de acero ASTM que pertenecen a la serie AXXX deberíamos usar la segunda columna de la tabla (Número de calibre de alambre U.S. Steel) a excepción del alambre para instrumentos musicales A228 que corresponde a la tercera columna.

Tabla II-2. Calibres de alambre y diámetros para resorte.

Calibres de alambre y diámetros para resorte[34]				
Número de calibre	Número de calibre de alambre U.S. Steel (pulg)[a]	Calibre de alambre instrumentos musicales (pulg)[b]	Calibre Brown & Sharpe (pulg)[c]	Diámetros métricos recomendables (mm)[d]
7/0	0.490 0	-	-	13.0
6/0	0.461 5	0.004	0.580 0	12.0
5/0	0.430 5	0.005	0.516 5	11.0
4/0	0.393 8	0.006	0.460 0	10.0
3/0	0.362 5	0.007	0.409 6	9.0
2/0	0.331 0	0.008	0.364 8	8.5
0	0.306 5	0.009	0.324 9	8.0
1	0.283 0	0.010	0.289 3	7.0
2	0.262 5	0.011	0.257 6	6.5
3	0.243 7	0.012	0.229 4	6.0
4	0.225 3	0.013	0.204 3	5.5
5	0.207 0	0.014	0.181 9	5.0
6	0.192 0	0.016	0.162 0	4.8
7	0.177 0	0.018	0.144 3	4.5
8	0.162 0	0.020	0.128 5	4.0
9	0.148 3	0.022	0.114 4	3.8
10	0.135 0	0.024	0.101 9	3.5
11	0.120 5	0.026	0.090 7	3.0
12	0.105 5	0.029	0.080 8	2.8
13	0.091 5	0.031	0.072 0	2.5
14	0.080 0	0.033	0.064 1	2.0
15	0.072 0	0.035	0.057 1	1.8
16	0.062 5	0.037	0.050 8	1.6
17	0.054 0	0.039	0.045 3	1.4
18	0.047 5	0.041	0.040 3	1.2
19	0.041 0	0.043	0.035 9	1.0

[34] Fuente: Associated Spring Barnes Group Inc., Engineering Guide to Spring Design. Bristol, Conn., 1987; Carlson Harold. Spring Designer's Handbook . Nueva york: Marcel Dekker, 1978; Oberg E., et al. Machinery's Handbook, 23ª edición. Nueva York: Industrial Press, 1988.

20	0.034 8	0.045	0.032 0	0.90
21	0.031 7	0.047	0.028 5	0.80
22	0.028 6	0.049	0.025 3	0.70
23	0.025 8	0.051	0.022 6	0.65
24	0.023 0	0.055	0.020 1	0.60 o 0.55
25	0.020 4	0.059	0.017 9	0.50 o 0.55
26	0.018 1	0.063	0.015 9	0.45
27	0.017 3	0.067	0.014 2	0.45
28	0.016 2	0.071	0.012 6	0.40
29	0.015 0	0.075	0.011 3	0.40
30	0.014 0	0.080	0.010 0	0.35
31	0.013 2	0.085	0.008 93	0.35
32	0.012 8	0.090	0.007 95	0.30 o 0.35
33	0.011 8	0.095	0.007 08	0.30
34	0.010 4	0.100	0.006 30	0.28
35	0.009 5	0.106	0.005 01	0.25
36	0.009 0	0.112	0.005 00	0.22
37	0.008 5	0.118	0.004 45	0.22
38	0.008 0	0.124	0.003 96	0.20
39	0.007 5	0.130	0.003 53	0.20
40	0.007 0	0.138	0.003 14	0.18

Fuente: Associated Spring Barnes Group Inc., Engineering Guide to Spring Design. Bristol, Conn., 1987; Carlson Harold. Spring Designer's Handbook . Nueva york: Marcel Dekker, 1978; Oberg E., et al. Machinery's Handbook, 23a edición. Nueva York: Industrial Press, 1988

[a] Utilice el calibre de alambre U.S. Steel para alambre de acero, excepto alambre para instrumentos musicales. A este calibre también se le da el nombre de calibre Washburn & Moen, calibre American Steel Wire Co., calibre Roebling Wire

[b] Sólo utilice el calibre de alambre para instrumentos musicales para ese alambre (ASTM A228).

[c] Utilice el calibre Brown & Sharpe para alambre no ferrosos como de latón y de bronce con fósforo.

[d] Los tamaños métricos que se recomiendan son de Associated Spring, Barnes Group, Inc., y se enumeran como los tamaños métricos que más se aproximan al calibre del alambre U.S. Steel. Los números de calibre no se aplican.

2.3.10.2. *Longitudes*

De acuerdo a la fuerza que se ejerce se pueden señalar varias longitudes del resorte (véase la Figura II-22).

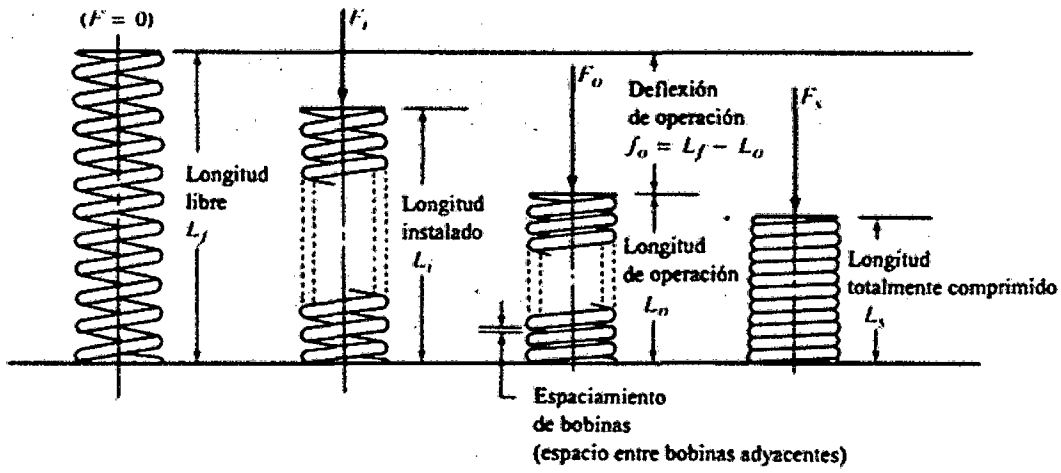


Figura II-22. Notación para longitudes y fuerzas.

Longitud libre, L_f, es la longitud del resorte cuando no ejerce fuerza alguna.

Longitud Comprimido, L_s, es la longitud cuando el resorte esta comprimido hasta el punto en que todas las bobinas se encuentran en contacto entre sí.

Longitud de operación, L_o, es la longitud más corta del resorte durante su funcionamiento normal.

Longitud instalado, L_i, es la longitud que generalmente tiene el resorte cuando se ejerce alguna fuerza en el lugar donde se instala.

2.3.10.3. *Fuerzas*

Un resorte puede ejercer varias fuerzas y tenemos:

F_s = Fuerza en longitud comprimido, la fuerza máxima que se observa en el resorte.

F_o = Fuerza en longitud de operación, la fuerza máxima que ejerce el resorte en operación normal.

F_i = Fuerza en longitud instalado, la fuerza que ejerce el resorte cuando esta instalado.

F_f = Fuerza en longitud libre, esta fuerza es cero.

2.3.10.4. Razón de Resorte

La razón de resorte es la relación entre la fuerza que ejerce un resorte y su deflexión. Así tenemos que esta razón la podemos representar de diversas formas[35] (Ecuación (2-1)).

$$k = \frac{\Delta F}{\Delta L} = \frac{F_o - F_i}{L_i - L_o} = \frac{F_o}{L_f - L_o} = \frac{F_i}{L_f - L_i} \qquad (2 - 1)$$

Esto implica que si conocemos la razón de resorte, se puede calcular la fuerza a cualquier deflexión.

2.3.10.5. Índice de Resorte

Es la relación del diámetro medio del resorte con el diámetro del alambre y se representa con la letra, C[36] (Ecuación (2-2)) .

$$C = \frac{D_m}{D_w} \qquad (2\text{-}2)$$

Se recomienda que *C* este entre 5 y 12, para resortes típicos para maquinaria. Con valores de *C* menores a 5, la formación del resorte será muy difícil y la deformación severa que se requiere puede generar fracturas en el resorte. Un *C* muy grande contribuirá a la tendencia al pandeo en el resorte. Las tensiones o esfuerzos de los resortes dependen de *C*.

2.3.10.6. Número de bobinas

Con *N* se representará al número total de bobinas en un resorte. Cuando se realiza el cálculo de esfuerzos y deflexiones de un resorte, las bobinas que permanecen inactivas se omiten. Así tenemos en resortes con extremos lijados y a escuadra o sólo extremos a escuadras, cada extremo de bobina está

---

[35] Moot, Robert; "Diseño de elementos de máquinas"; Segunda edición, Editorial Prentice Hall; México 1992. Pág. 212, 213.

[36] Moot, Robert; "Diseño de elementos de máquinas"; Segunda edición, Editorial Prentice Hall; México 1992. Pág. 214.

inactivo y el número de bobinas activas, N_a, es $N - 2$. Para bobinas en bruto, $N_a = N$. Para bobinas en bruto con extremos lijados, $N_a = N - 1$.

2.3.10.7. Espaciamiento

Se representa por p, y es la distancia axial de un punto en una bobina al punto correspondiente de la bobina adyacente siguiente. Tenemos fórmulas de la longitud libre que relacionan estas con el diámetro del alambre y el número de bobinas activas.

Extremos a escuadra y lijados (ecuación (2-3)):

$$L_f = pN_a + 2D_w \qquad (2\text{-}3)$$

Sólo extremos a escuadra (ecuación (2-4)):

$$L_f = pN_a + 3D_w \qquad (2\text{-}4)$$

Extremos en bruto y lijados (ecuación (2-5)):

$$L_f = p(N_a + 1)a \qquad (2\text{-}5)$$

Extremos en bruto (ecuación (2-6)):

$$L_f = pN_a + D_w \qquad (2\text{-}6)$$

2.3.10.8. Angulo de espaciamiento

En la mayor parte de los diseños prácticos el ángulo de espaciamiento[37], λ, es aproximadamente de menos de 12°, este ángulo de se muestra en la Figura II-23. Con ángulos de espaciamiento mayores a 12°, se obtendrán tensiones por comprensión indeseables en el alambre, y las fórmulas que utilicemos más adelante resultarán imprecisas. El ángulo de espaciamiento puede calcularse con la ecuación (2-7).

---

[37] Moot, Robert; "Diseño de elementos de máquinas"; Segunda edición, Editorial Prentice Hall; México 1992. Pág. 215.

$$\lambda = \arctan\left[\frac{p}{\pi D_m}\right] \qquad (2\text{–}7)$$

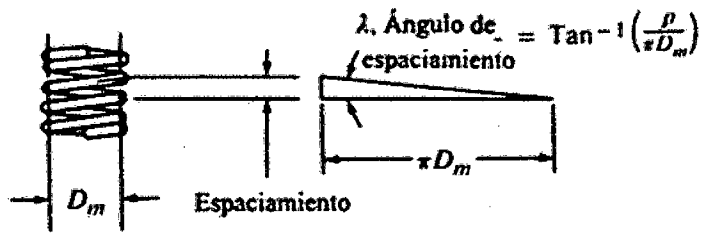


Figura II-23. Notación para longitudes y fuerzas.

2.3.10.9. Consideraciones en la instalación

Al instalar un resorte se deben tomar en consideración donde se hace esta instalación (en un orificio, en una varilla, etc.) para prever los márgenes adecuados. Un resorte de compresión cuando se comprime, aumenta su diámetro, por ello, si el resorte esta en un orificio, el diámetro interior del orificio debe ser mayor que el diámetro exterior del resorte para evitar la fricción. Se sugiere un margen de diámetro inicial equivalente a 1/10 del diámetro del alambre para resortes que tienen un diámetro de 0.50" (12mm) o mayor. La ecuación (2-8) puede dar una estimación más precisa para el diámetro exterior real del resorte en la longitud de comprimido (OD_s)[38].

$$OD_s = \sqrt{D_m^2 + \frac{p^2 - D_w^2}{\pi^2}} + D_w \qquad (2\text{-}8)$$

El diámetro interior (ID) del resorte también se hace más grande cuando se comprime, Mott sugiere que el margen de ID sea de 0.10 D_W.

2.3.10.10. Margen de bobina

El margen de bobina, cc, se refiere al espacio entre bobinas adyacentes cuando el resorte se comprime hasta su longitud de operación, L_o. Se puede estimar con la ecuación (2-9).

---

[38] Moot, Robert; "Diseño de elementos de máquinas"; Segunda edición, Editorial Prentice Hall; México 1992. Pág. 215.

$$cc = \frac{(L_o - L_s)}{N_a} \qquad (2\text{-}9)$$

Una recomendación a considerar según Mott es que el margen de bobina debe ser mayor que $D_w/10$, más aún cuando soportan cargas cíclicas.

Otra de las recomendaciones que hace Mott acerca de la deflexión total del resorte es la siguiente:

$$(L_o - L_s) > 0.15(L_f - L_s)$$

2.3.10.11. Materiales utilizados para fabricar resortes

En la mayor parte de las aplicaciones mecánicas se emplea alambre metálico, que puede ser acero al alto carbón o acero con aleación de acero inoxidable, latón, bronce, cobre con berilio o aleaciones con base de níquel. Casi todos los materiales para fabricar resortes cumplen con las especificaciones de la ASTM. En la tabla II-3 se presentan algunos de los más comunes.

Tabla II-3. Materiales para resortes.

Tipo de Material	No. ASTM	Costo relativo	Límites de temperatura
Aceros al alto carbón			
Extruido en frío Acero para uso general con 0.60% a 0.70% de carbón; bajo costo.	A227	1.0	0-250
Alambre para instrumentos musicales Acero de alta calidad con 0.80% a 0.95% de carbón; muy alta resistencia; excelente acabado superficial; extruido duro; buen rendimiento en cuanto a fatiga; se usa sobre todo en tamaños pequeños de hasta 0.125"	A228	1.0	0-250
Templado en aceite Acero de uso general con 0.60% a 0.70% de carbón; se utiliza sobre todo en tamaños grandes por arriba de 0.125"; no es bueno para choque o impacto.	A229	1.3	0-350
Aceros con aleación			
Cromo y vanadio Buena resistencia, resistencia a la fatiga, resistencia al impacto, rendimiento de alta temperatura; calidad de resortes para válvulas.	A231	3.1	0-425

Cromo y silicio Muy alta resistencia y buena resistencia a la fatiga y al choque.	A401	4.0	0-475
Aceros inoxidables			
Tipo 302 Muy buena resistencia a la corrosión y rendimiento a alta temperatura; casi amagnético; extruido en frío; los tipos 304 y 306 también caen en esta categoría ASTM, tienen mayor susceptibilidad para ser trabajados pero su resistencia es más baja.	A313 (302)	7.6	<0-550
Tipo 17-7 PH Buen rendimiento a alta temperatura	A313 (631)	11.0	0-600
Aleaciones de cobre Todas tienen una buena resistencia a la corrosión y conductividad eléctrica.			
Latón para resortes	B134	Alto	0-150
Bronce con fósforo	B159	8.0	<0-212
Cobre con berilio	B197	27.0	0-300
Aleaciones con base de níquel Todas son resistentes a la corrosión, tienen buenas propiedades de alta y baja temperatura, son amagnéticos o casi amagnéticos (marcas comerciales de la internacional Nickel Company)			
Monel	-	-	-100-425
K-Monel	-	-	-100-450
Inconel	-	-	Arriba de 700
Inconel – X	-	44.0	Arriba de 850

Fuente: Associated Spring Barnes Gruoup Inc., Enginering Guide to Spring Design. Bristol, Conn., 1987; Carlson, Harold. Spring Designer's Handbook. Nueva York: Marcel Dekker, 1978; Oberg E., et al Machinery's Handbook, 23ava edición. Nueva York: Industrial Press; 1988.

Además tenemos gráficos[39] para tensiones por esfuerzo de corte de diseño para alambres de resortes que se utilizan en resortes de compresión y resortes helicoidales de compresión.

Para tensiónes de diseño, de alambre de acero ASTM A227, extruido en caliente, usar la Figura II-24.

[39] Harold Carlson, Spring Designer´s Handbook, cortesía de Marcel Dekker, Inc.

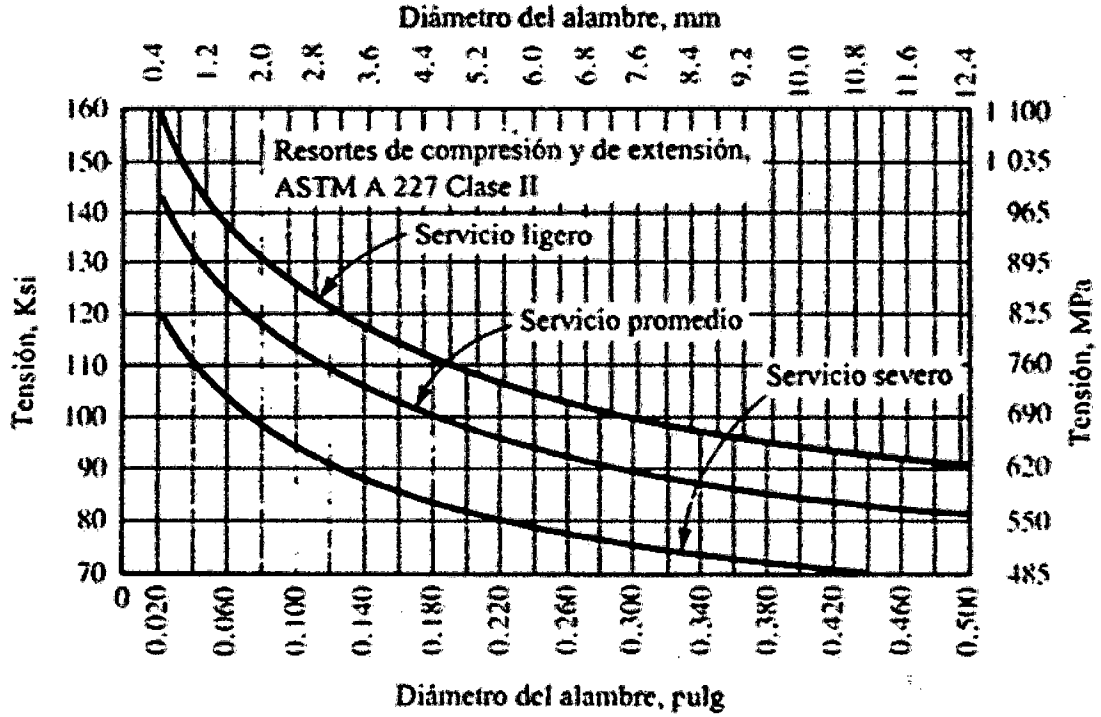


Figura II-24. Tensiones de diseño, alambre de acero ASTM A227.

Para tensiónes de diseño, de alambre para instrumentos musicales ASTM A228, extruido en caliente, usar la Figura II-25. Estas tensiones de diseño son similares para alambre de acero inoxidable, 17-7 PH, ASTM A 313, tipo 631.

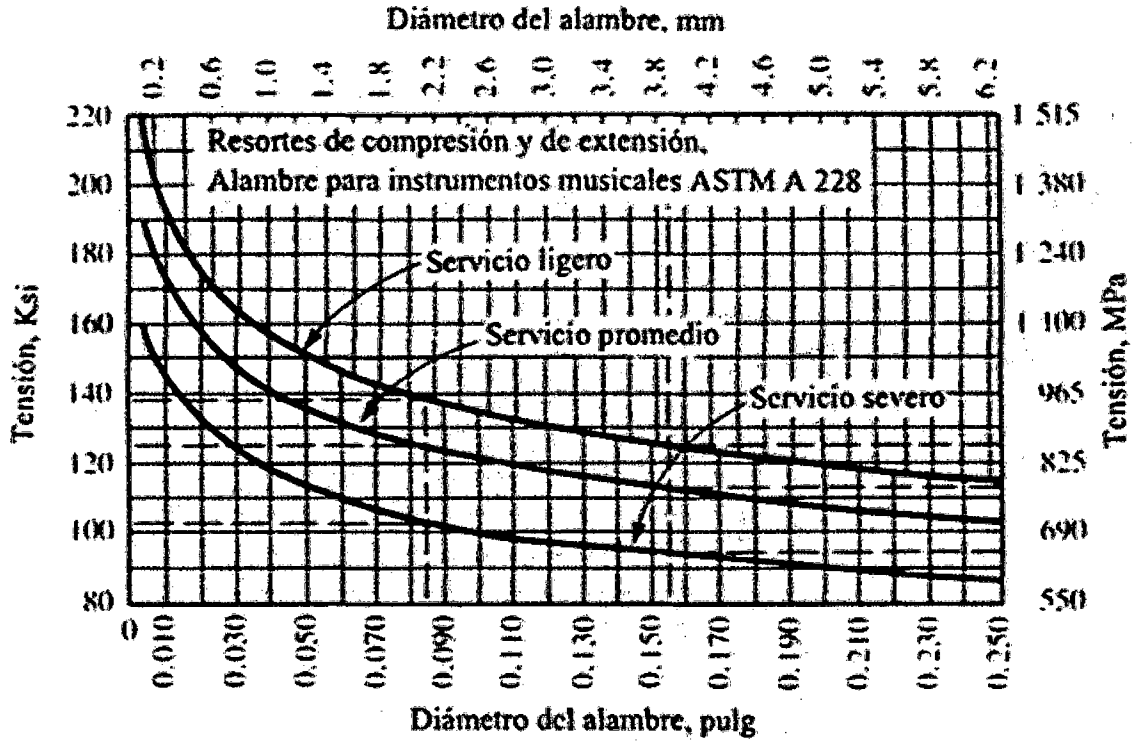


Figura II-25. Tensiones de diseño, alambre para instrumentos musicales ASTM A228.

Para tensiónes de diseño, de alambre de acero ASTM A229, templado en aceite, usar la Figura II-26.

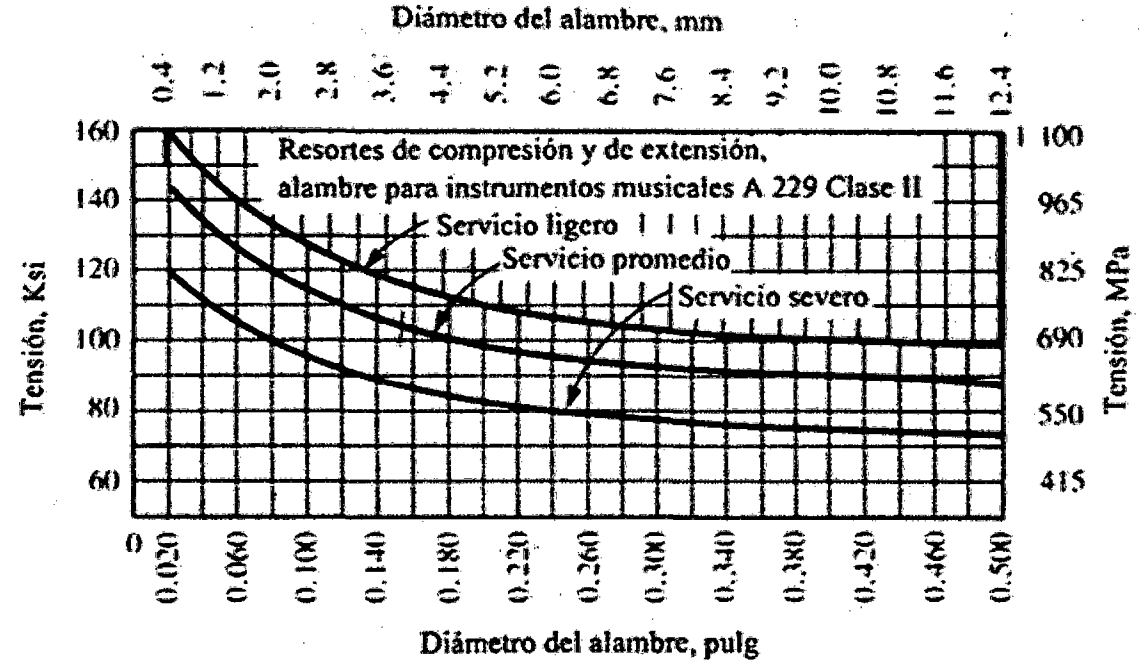


Figura II-26. Tensiones de diseño, alambre de acero ASTM A229.

Para tensiónes de diseño, de alambre de acero ASTM A231, con aleación de cromo y vanadio, calidad de resorte para válvulas, usar la Figura II-27.

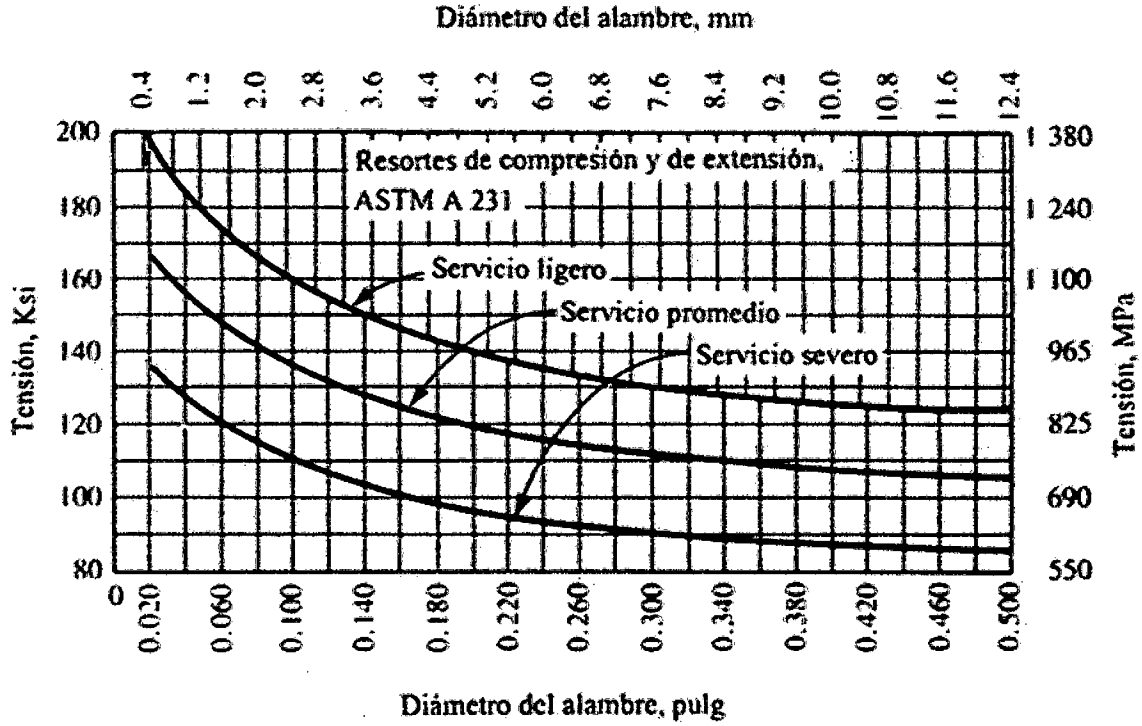


Figura II-27. Tensiones de diseño, alambre de acero ASTM A231

Para tensiones de diseño, de alambre de acero ASTM A401, con aleación de cromo y silicio, templado en aceite, usar la Figura II-10.

Para tensiones de diseño, de alambre de acero inoxidable resistente a la corrosión ASTM A313, templado en aceite, usar la Figura II-29. Use la misma Figura para alambre de acero inoxidable tipo 302. Para el tipo 304 multiplíquese por 0.95. Para el tipo 316 multiplíquese por 0.85.

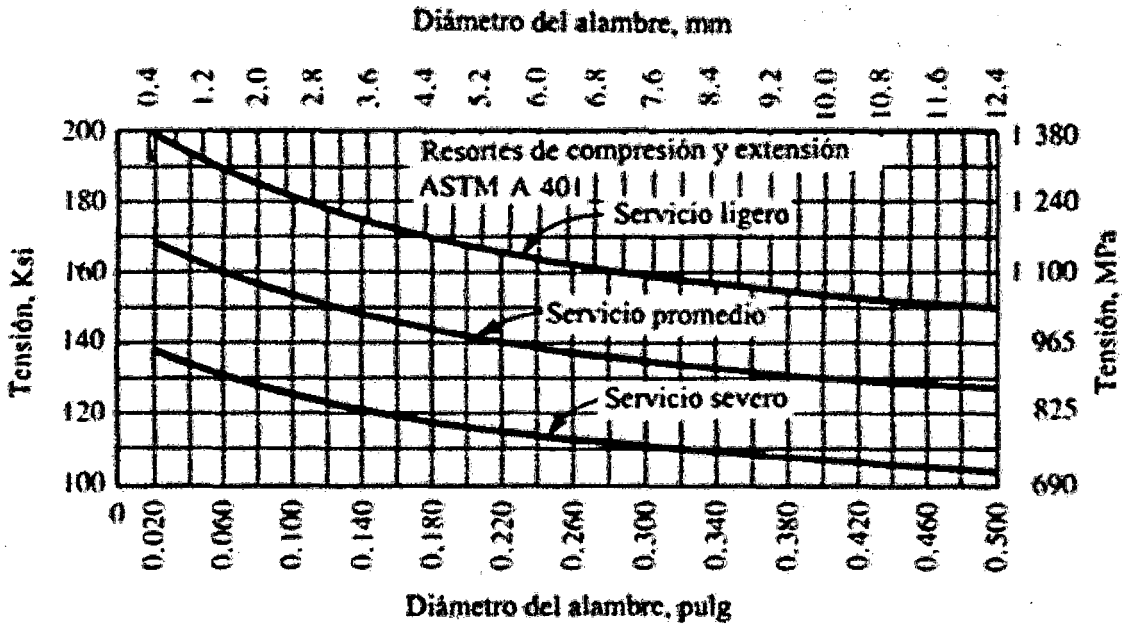


Figura II-28. Tensiones de diseño, alambre de acero ASTM A401

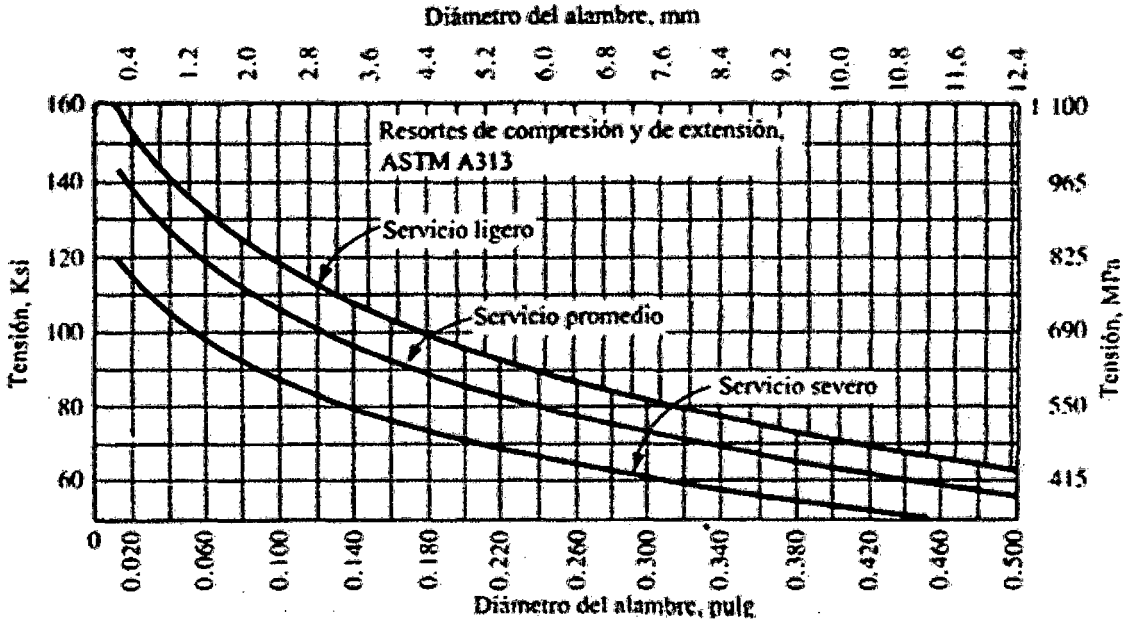


Figura II-29. Tensiones de diseño, alambre de acero ASTM A313.

2.3.10.12. Tipos de cargas y tensiones o esfuerzos permisibles

La tensión permisible de un resorte depende del tipo de carga, del material de fabricación y del diámetro del alambre.

Generalmente la carga se clasifica en:

- Servicio Ligero: cargas estáticas hasta 10 000 ciclos de carga con una tasa baja de carga (sin impacto).
- Servicio promedio: en el caso de diseño típicas para maquinaria; tasa de carga moderada y hasta 1 000 000 de ciclos.
- Servicio severo: ciclaje rápido para más de un millón de ciclos; posibilidad de carga por impacto o choque; los resortes para válvulas de motores son un ejemplo.

2.3.10.13. Tensiones y Deflexión para Resortes Helicoidales de Compresión

Cuando un resorte se comprime bajo una carga axial, el esfuerzo que se desarrolla en el alambre es *tensión por esfuerzo de corte por torsión*, representado por la ecuación (2-10).

$$\tau = \frac{Tc}{J} \qquad (2\text{-}10)$$

La ecuación resultante para el diseño se atribuye a Wahl. Así, la tensión máxima por esfuerzo de corte, que se presentará en la superficie interna del alambre se muestra en la ecuación (2-11).

$$\tau = \frac{8\mathrm{KFD_m}}{\pi D_w^3} = \frac{8KFC}{\pi D_w^2} \qquad (2\text{-}11)$$

Se puede observar en esta ecuación el impacto del diámetro del alambre en el rendimiento del resorte.

El factor de Wahl, *K*, en la ecuación 4-12 es el término al que se le atribuye la curvatura del alambre y la tensión por esfuerzo de corte directo. *K* está relacionada con *C*:

$$K = \frac{4C-1}{4C-4} + \frac{0.615}{C} \qquad (2\text{-}12)$$

Mott sugiere un valor mínimo de *C*=5. El valor de *K* se incrementa con rapidez para *C*<5.

2.3.10.14. Deflexión

La manera principal en que se carga el alambre de un resorte helicoidal es por torsión, la deflexión se calcula a partir de la fórmula de ángulo de combadura en la ecuación (2-13).

$$\theta = \frac{TL}{GJ} \qquad (2\text{-}13)$$

donde θ es el ángulo de combadura en radianes, T es el torque aplicado, L es la longitud del alambre, G es el módulo de elasticidad del material ante esfuerzo de corte y J es el momento de inercia polar del alambre.

La deflexión lineal, *f*, a partir de las variables típicas de diseño del resorte se muestra en la ecuación (2-14).

$$f = \frac{8FD_m^3 N_a}{GD_w^4} = \frac{8FC^3 N_a}{GD_w} \qquad (2\text{-}14)$$

La Tabla II-4 presenta los valores de G para materiales típicos con que se fabrican resortes. Estos datos son valores promedios. Se pueden presentar variaciones pequeñas en el tamaño del alambre y el tratamiento.

Tabla II-4. Modulo de elasticidad de alambre para resorte en corte (G) y Tensión (E).

Material y número ASTM	Módulo de elasticidad, G		Modulo de tensión, E	
	(psi)	(GPa)	(psi)	(GPa)
Acero duro extruido: A227	11.5×10^6	79.3	28.6×10^6	197
Alambre para instrumentos musicales: A228	11.85×10^6	81.7	29.0×10^6	200
Templado en aceite: A229	11.2×10^6	77.2	28.5×10^6	196
Cromo y vanadio: A231	11.2×10^6	77.2	28.5×10^6	196
Cromo y silicio: A401	11.2×10^6	77.2	29.5×10^6	203
Aceros inoxidables: A313 Tipos 302, 304, 316 Tipo 17-7 PH	 10.0×10^6 10.5×10^6	 69.0 72.4	 28.0×10^6 29.5×10^6	 193 203
Latón para resortes: B134	5.0×10^6	34.5	15.0×10^6	103
Bronce con fósforo: B159	6.0×10^6	41.4	15.0×10^6	103
Cobre con berilio: B197	7.0×10^6	48.3	17.0×10^6	117
Monel y K-Monel	9.5×10^6	65.5	26.0×10^6	179
Iconel e Iconel X	10.5×10^6	72.4	31.0×10^6	214

Fuente: Mott, R. L. Diseño de Elementos de Máquinas. México: Prentice Hall; 1995. Apéndice 20 (Pág. A-48)

2.3.10.15. *Pandeo*

La tendencia a pandeo de un resorte aumenta a medida que éste es más alto y delgado. La Figura II-30 muestra gráficas de la razón o relación crítica de deflexión hasta la longitud libre contra la razón de longitud libre contra la razón de longitud libre respecto al diámetro medio del resorte. En la figura se describen tres condiciones de sujeción distintas en los extremos. En este gráfico f_o representa la deflexión de operación. En la Figura II-30, si la relación real de f_o/L_f es mayor que la razón crítica, el resorte se pandeará al someterse a deflexión de operación.

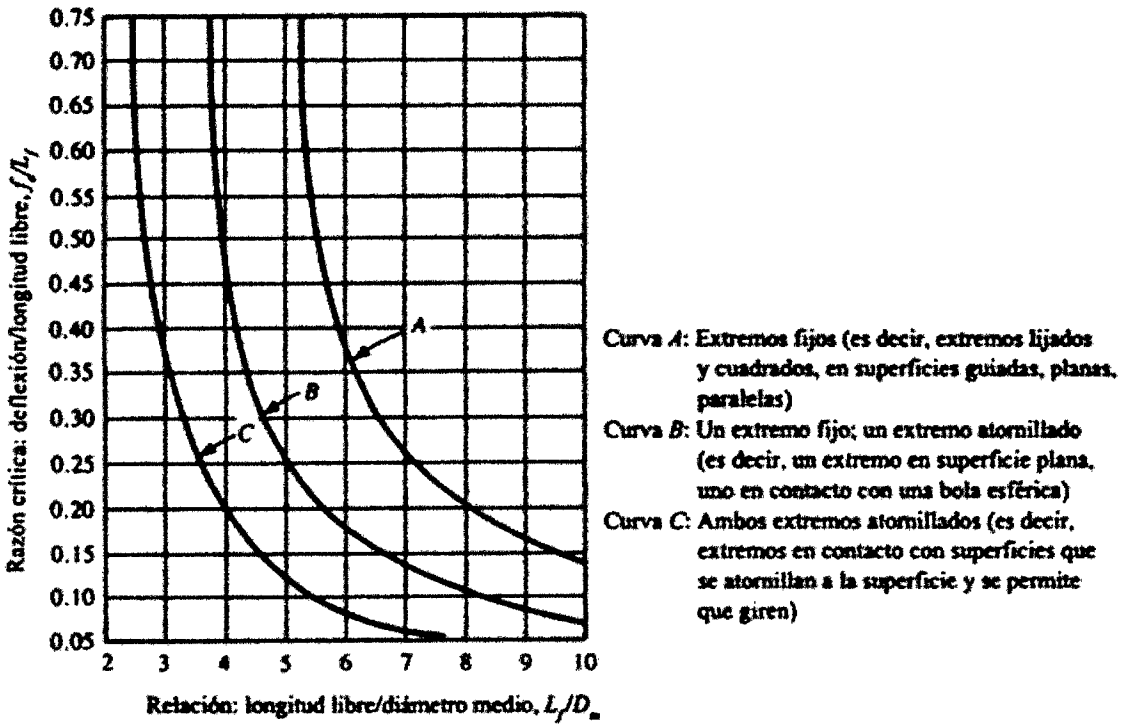


Figura II-30. Criterio para pandeo de resortes.

2.3.11. *ANÁLISIS DE RESORTES HELICOIDALES DE COMPRESIÓN*

El objetivo fundamental es el de analizar la geometría y las características asociadas al rendimiento de un resorte. En este análisis es necesario conocer el material de fabricación del resorte.

Para el análisis es deseable conocer los siguientes datos de entrada:

- Material del resorte.

- Tipo de servicio en base al número de ciclos que se espera (ligero, promedio o severo).
- Longitud Libre
- Diámetro exterior
- Diámetro del alambre
- Tipo de extremos
- Número total de bobinas
- Carga normal de operación

A partir de los datos anteriores se puede calcular:

a. Número de calibre para el alambre, diámetro medio, diámetro interior, índice de resorte, y factor de Wahl.

   El calibre se puede encontrar en la tabla II-2.

   Diámetro medio (ecuación 2-15):

$$D_m = OD - D_w \qquad (2\text{-}15)$$

   Diámetro interior (ecuación 2-16):

$$ID = D_m - D_w \qquad (2\text{-}16)$$

   Índice de resorte (ecuación 2-17):

$$C = D_m / D_w \qquad (2\text{-}17)$$

   Factor de Wahl (ecuación 2-18):

$$K = (4C-1)/(4C-4) + 0.615/C \qquad (2\text{-}18)$$

b. El esfuerzo o tensión que se espera con la carga de operación.

   Se calcula con la ecuación 2-11 con Fo:

$$\tau_o = \frac{8KF_oC}{\pi D_w^2} \qquad (2\text{-}19)$$

c. La deflexión del resorte bajo la acción de la carga de operación.

   Se calcula con la ecuación 2-14 con F_o:

$$f_o = \frac{8F_oC^3N_a}{GD_w} \qquad (2\text{-}20)$$

N_a depende del tipo de extremo del resorte. G se obtiene de la tabla IV-4. El valor f_o es la deflexión de longitud sin comprensión a la longitud de operación.

d. Longitud de operación, longitud comprimido y razón de resorte.

Longitud de operación:

$$L_o = L_f - f_o \qquad (2\text{-}21)$$

Longitud comprimido:

$$L_s = D_wN \qquad (2\text{-}22)$$

Razón de resorte:

$$k = \frac{\Delta F}{\Delta L} = \frac{F_o}{L_f - L_o} = \frac{F_o}{f_o} \qquad (2\text{-}23)$$

e. Fuerza del resorte cuando se encuentra en longitud comprimido y el esfuerzo correspondiente a longitud comprimido.

Fuerza en longitud comprimido:

$$F_s = k\left(L_f - L_s\right) \qquad (2\text{-}24)$$

Tensión o esfuerzo en longitud comprimido: Se puede calcular con la ecuación 2-4 utilizando F = Fs; o mejor aun considerando la proporcionalidad de esfuerzos y fuerzas se considera

$$\tau_s = \tau_o \frac{F_s}{F_o} \qquad (2\text{-}25)$$

f. Comparación de la tensión de diseño del material con la tensión en longitud de operación.

τ_d, Tensión de diseño, se calcula a partir de las gráficas de tensión de diseño, (figuras del II-24 al II-29) dependiendo del tipo del material del resorte, el diámetro del alambre y del tipo de servicio en base al número de

ciclos de carga que se espera. Si la tensión real de operación, τ_o, es menor que τ_d, resulta satisfactoria.

g. Comparación del esfuerzo o tensión máxima permisible con el esfuerzo en longitud comprimido

   τ_{max}, se sugiere determinarlo empleando la curva de servicio ligero en la gráfica de tensión de diseño (figuras del II-24 al II-29). La tensión máxima real que se espera se presenta en tensión comprimido, si la tensión comprimido es menor que la tensión máxima se considera el diseño como satisfactorio.

h. Verificación del resorte en cuanto al pandeo y márgenes de bobina

   Para verificar el resorte en cuanto a pandeo se utiliza la Figura II-30 según el tipo de sujeción del extremo del resorte. Si la relación real f_o/L_f es mayor que la relación crítica en el gráfico, el resorte se pandeará. La relación crítica en el gráfico depende de L_f/D_m.

   El margen para las bobinas, *cc*, se puede calcular como:

$$cc = \frac{L_o - L_s}{N_a} \qquad (2\text{-}26)$$

   Luego *cc* se compara con el margen mínimo que se sugiere ($D_w/10$).

i. Diámetro adecuado para un orificio en el cual se va a instalar el resorte.

   Se sugiere que el diámetro del orificio en el que se instalará el resorte cumpla con:

$$D_{orificio} > OD + \frac{D_w}{10} \qquad (2\text{-}27)$$

2.3.12. DISEÑO DE RESORTES HELICOIDALES DE COMPRESIÓN

Por lo general se considera el Diseño de Resortes de Compresión Helicoidal considerando los siguientes datos:

- Fuerza de instalado
- Longitud en instalado

- Fuerza de operación
- Longitud en operación.
- Cantidad de ciclos que se esperan en el resorte
- Temperatura a la que opera el resorte
- Diámetro del orificio donde se planea instalar el resorte

Se debe especificar para la propuesta de Diseño de Resortes de Compresión Helicoidal la siguiente información:

- Material adecuado para el resorte.
- Diámetro del alambre.
- Diámetro medio.
- Diámetro exterior.
- Diámetro interior
- Longitud libre
- Longitud comprimido
- Número de bobinas y tipo de condición en los extremos.

Se debe verificar la tensión en la carga máxima de operación y la condición en longitud comprimido.

2.3.12.1. Método 1

Este procedimiento se enfoca en la geometría del resorte, especificando el diámetro medio que satisface las restricciones en cuanto a espacio. Este proceso implica que el diseñador tenga tablas con los datos correspondientes a diámetros de alambre (tabla II-2), y tensiones de diseño para el material con que se va a fabricar el resorte (figuras del IV-24 al IV-29). Se considera hacer las estimaciones iníciales para las tensiones de diseño consultando las gráficas de las mismas contra diámetro de alambre para tomar una decisión lógica. Por lo general se realizarán varias pruebas.

Para el diseño del resorte consideramos los siguientes pasos:

i. Especifique el material y su Módulo de elasticidad ante esfuerzo de corte (G)

ii. Especifique para el resorte:

Fuerza en longitud de operación (F_o) (la fuerza máxima que observa el resorte en operación normal)

Longitud de operación (L_o)

Fuerza en longitud de instalado (F_i)

Longitud de instalado (L_i)

iii. Calcule la razón de resorte

$$k = \frac{F_o - F_i}{L_i - L_o} \qquad (2\text{-}28)$$

iv. Calcule la Longitud Libre

$$L_f = L_i + \frac{F_i}{k} \qquad (2\text{-}29)$$

v. Especifique un estimado inicial para el diámetro medio (D_m) considerando que debe ser menor que el orificio de instalación

vi. Especifique un estimado inicial para la tensión de diseño (τ_d).

Este estimado inicial se basa en la resistencia del material, consultando las gráficas para las tensiones de diseño de los materiales seleccionados y de acuerdo con su servicio.

vii. Calcule el Diámetro del alambre D_w para la prueba, considerando el Factor de Wahl K = 1.2

$$D_w = \sqrt[3]{\frac{8KF_oD_m}{\pi\tau_d}} \qquad (2\text{-}30)$$

D_w se determina despejando la ecuación 2-14. Si se grafica el índice de resorte contra el factor de Wahl (Figura II-31) se puede observar que el valor promedio aproximado de K es 1.2.

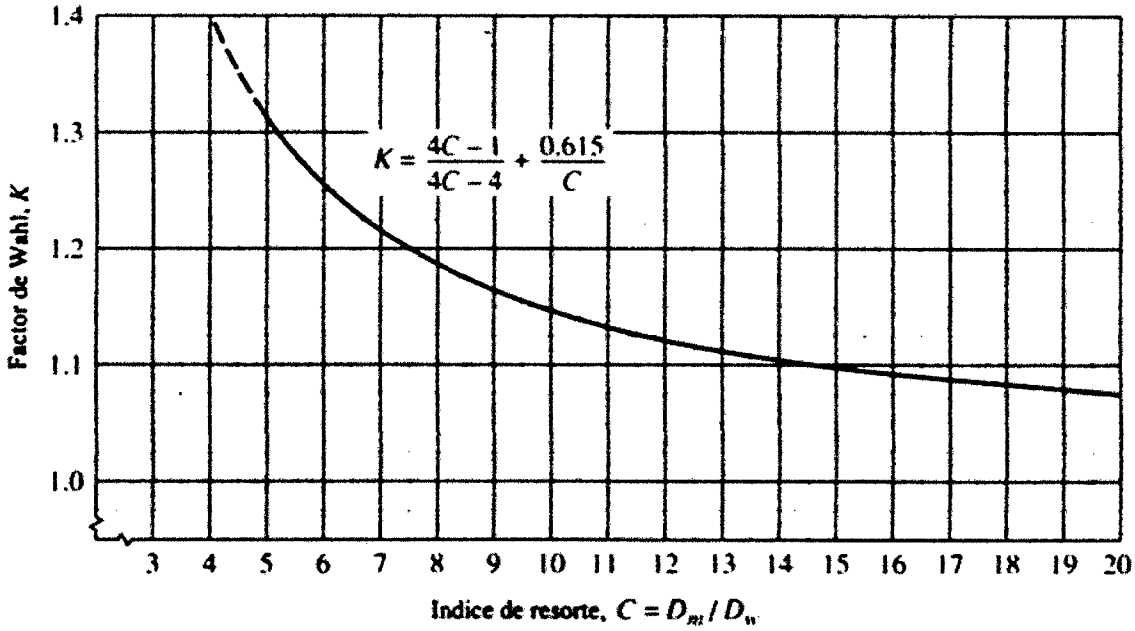


Figura II-31. C contra K para alambre redondo.

Con este valor podríamos simplificar el cálculo aproximado de Dw, obteniendo

$$D_w = \sqrt[3]{\frac{(8)(1.2)F_o D_m}{\pi \tau_d}} = \sqrt[3]{\frac{(3.06)F_o D_m}{\tau_d}} \qquad (2\text{-}31)$$

viii. Seleccione un diámetro de alambre estándar (D_w) de las tablas, este diámetro será próximo al calculado en el paso anterior.

ix. Especifique la tensión de diseño (τ_d) y la tensión máxima permisible para el material con ese diámetro de alambre (τ_{max}).

La tensión de diseño (τ_d) para el material con que se va a fabricar el resorte se obtiene de las figuras del II-24 al IV-29 según el servicio que presta (según el número de ciclos esperado).

La curva de servicio ligero se usará como estimación del esfuerzo o tensión máxima permisible (τ_{max}), porque se acerca mucho a la resistencia a punto cedente.

x. Calcule los valores reales del índice del resorte (*C*) y el factor de Wahl (*K*)

$$C = \frac{D_m}{D_w} \qquad (2\text{-}32)$$

$$K = \frac{4C-1}{4C-4} + \frac{0.615}{C} \qquad (2\text{-}33)$$

Si $C > 5$ el índice del resorte es aceptable. En caso contrario se considera que el índice de resorte es muy bajo, vuelva al paso 8 o al paso 5.

xi. Calcule la tensión real que se espera debido a la fuerza de operación, F_o, a partir de la ecuación 4-11.

$$\tau_o = \frac{8KF_oD_m}{\pi D_w^3} \qquad (2\text{-}34)$$

Si $\tau_o < \tau_d$ El diseño es seguro.

xii. Calcule el Número de bobinas activas, N_a, que se necesitan para darle la deflexión al resorte.

En base a la ecuación 4-14 y considerando que $F/f = k$, podemos despejar N_a. Obtenemos:

$$N_a = \frac{fGD_w}{8FC^3} = \frac{GD_w}{8kC^3} \qquad (2\text{-}35)$$

xiii. Calcule la longitud comprimido, L_s.

$$L_s = D_wN \qquad (2\text{-}36)$$

El número de bobinas, N, depende del tipo de extremo del resorte:

- Resortes con extremos lijados y a escuadra: $N = N_a + 2$
- Resortes con sólo extremos a escuadra: $N = N_a + 2$
- Resortes con bobinas en bruto: $N = N_a$
- Resortes con bobinas en bruto con extremos lijados: $N = N_a + 1$

Si $L_s < L_o$ se sigue calculando. En caso contrario, se presentó una situación imposible y vuelva al Paso 5 o al Paso 8.

xiv. Calcule la fuerza en el resorte en longitud comprimido, F_s, como el producto de la razón del resorte por la deflexión en longitud de comprimido $(L_f - L_s)$

$$F_s = k(L_f - L_s) \qquad (2\text{-}37)$$

xv. Calcule el esfuerzo o tensión en el resorte en longitud comprimido, τ_s.

La tensión o esfuerzo en el resorte es directamente proporcional a la fuerza, podemos considerar para la tensión en longitud comprimido:

$$\tau_s = \tau_o \frac{F_s}{F_o} \qquad (2\text{-}38)$$

Si $\tau_s < \tau_{max}$, el resorte no presentará cedencia. En caso contrario vuelva al Paso 5 o al Paso 8.

xvi. Verifique el resorte en cuanto al pandeo

Para verificar el resorte en cuanto a pandeo se utiliza la Figura II-30 según el tipo de sujeción en el extremo del resorte. Si la relación real f_o/L_f es mayor que la Razón Crítica en el gráfico, el resorte se pandeará. La Razón Crítica en el gráfico depende de L_f/D_m.

xvii. Verifique el margen para las bobinas, cc:

$$cc = \frac{L_o - L_s}{N_a} \qquad (2\text{-}39)$$

Si $cc > \frac{D_w}{10}$, el margen se considera aceptable.

xviii. Calcule el Diámetro Exterior (*OD*)

$$OD = D_m + D_w \qquad (2\text{-}40)$$

Si el resorte se instala en un orificio, para evitar fricción se debe cumplir que:

$$D_{orificio} > OD + \frac{D_w}{10} \qquad (2\text{-}41)$$

xix. Calcule el Diámetro Interior

$$ID = D_m - D_w \qquad (2\text{-}42)$$

Si el resorte se instala en una varilla, para evitar fricción se debe cumplir que:

$$D_{varilla} < ID - \frac{D_w}{10} \qquad (2\text{-}43)$$

2.3.12.2. Método 2

Este procedimiento proporciona un margen extenso para que el diseñador realice diseños que satisfagan las necesidades básicas en cuanto a fuerza, longitud y tensión o esfuerzo; sin embargo la geometría del resorte no se controla de manera estricta; es un resultado del procedimiento, más que un factor que deba especificarse al principio, como en el método 1.

Para el diseño del resorte consideramos los siguientes pasos:

i. Especifique el material y su Módulo de elasticidad ante esfuerzo de corte (*G*).

ii. Especifique el tipo de extremo que tendrá el resorte

iii. Especifique para el resorte:

Fuerza en longitud de operación (F_o) (la fuerza máxima que observa el resorte en operación normal)

Longitud de operación (L_o)

Fuerza en longitud de instalado (F_i)

Longitud de instalado (L_i)

iv. Calcule la razón de resorte

$$k = \frac{F_o - F_i}{L_i - L_o} \qquad (2\text{-}44)$$

v. Calcula la Longitud Libre

$$L_f = L_i + \frac{F_i}{k} \qquad (2\text{-}45)$$

vi. Determine la deflexión en Longitud de Operación

$$f_o = L_f - L_o \qquad (2\text{-}46)$$

vii. Ingrese un estimado inicial del diámetro del alambre (D_w)

viii. Ingrese el tipo de servicio del resorte.

Puede ser:

- Servicio Ligero
- Servicio Promedio
- Servicio Severo

ix. Calcule la tensión de diseño, τ_d, en función al diámetro de alambre (D_w) y del material para servicio ligero, promedio o severo. (Use las figuras del II-24 al II-29).

x. Calcule la tensión máxima permisible, τ_{max}, en función al diámetro de alambre (D_w) y del material (Use las figuras del II-24 al II-29). La curva de servicio ligero se usará como estimación del esfuerzo o tensión máxima permisible (τ_{max}), porque se acerca mucho a la resistencia a punto cedente.

xi. Calcule el nuevo diámetro del alambre (D_w) de prueba

Para calcular el nuevo diámetro se toma en cuenta la ecuación 2-11, con $F = F_o$ y $\tau = \tau_d$, y despejando D_w, tenemos:

$$D_w = \sqrt{\frac{8KF_oC}{\pi\tau_d}} \qquad (2\text{-}47)$$

De la Figura II-31 se puede observar que un estimado promedio para el Factor de Wahl, K, es de 1.2. Un estimado aceptable para el índice de resorte, C, sería de 7 (considerando que su valor mínimo debe ser igual a 5). De esto podemos obtener la fórmula que me permita obtener el nuevo diámetro del alambre (D_w) de forma razonable:

$$D_w = \sqrt{\frac{21.4F_o}{\tau_d}} \qquad (2\text{-}48)$$

xii. Busque y seleccione en la lista de tamaños estándar de alambres (Figura II-30) para encontrar el tamaño más cercano al obtenido en el paso anterior. Este será el nuevo diámetro del alambre (Dw)

xiii. Calcule nuevamente la tensión de diseño (τ_d) y la tensión máxima permisible (τ_{max}) con un procedimiento igual al realizado en los pasos 9 y 10.

xiv. Calcule el número máximo permisible de bobinas activas ($(N_a)_{max}$) para el resorte.

Para calcular este valor se considera que la longitud comprimido debe ser menor que la longitud de operación. La longitud de comprimido se calcula con $L_s = D_w N$, donde N depende del tipo de extremo del resorte.

- Resortes con extremos lijados y a escuadra:

$$N = N_a + 2 \qquad (2\text{-}49)$$
$$L_s = D_w(N_a + 2) \qquad (2\text{-}50)$$

- Resortes con sólo extremos a escuadra:

$$N = N_a + 2 \qquad (2\text{-}51)$$
$$L_s = D_w(N_a + 2) \qquad (2\text{-}52)$$

- Resortes con bobinas en bruto:

$$N = N_a \qquad (2\text{-}53)$$
$$L_s = D_w(N_a) \qquad (2\text{-}54)$$

- Resortes con bobinas en bruto con extremos lijados:

$$N = N_a + 1 \qquad (2\text{-}55)$$

$$L_s = D_w(N_a + 1) \qquad (2\text{-}56)$$

Considerando $L_s = L_o$ como un límite para encontrar el número máximo permisible de de bobinas activas ($(N_a)_{max}$)

- Resortes con extremos lijados y a escuadra:

$$(N_a)_{\max} = \frac{L_o - 2D_w}{D_w} \qquad (2\text{-}57)$$

- Resortes con sólo extremos a escuadra:

$$(N_a)_{\max} = \frac{L_o - 2D_w}{D_w} \qquad (2\text{-}58)$$

- Resortes con bobinas en bruto:

$$(N_a)_{\max} = \frac{L_o}{D_w} \qquad (2\text{-}59)$$

- Resortes con bobinas en bruto con extremos lijados:

$$(N_a)_{\max} = \frac{L_o - D_w}{D_w} \qquad (2\text{-}60)$$

xv. Seleccione el número de bobinas (N_a)

Se selecciona un valor menor al máximo calculado en el paso anterior. Cuanto menor sea el valor de N_a, habrá más espacio entre las bobinas adyacentes pero las tensiones serán más altas.

xvi. Calcule el índice de resorte (C)

En base a la ecuación 2-14 y considerando que $F/f = k$, podemos despejar C. Obtenemos:

$$N_a = \frac{fGD_w}{8FC^3} = \frac{GD_w}{8kC^3} \qquad (2\text{-}61)$$

Despejando el índice de resorte (C):

$$C = \left[\frac{GD_w}{8kN_a}\right]^{\frac{1}{3}} \qquad (2\text{-}62)$$

Si C > 5 el índice del resorte es aceptable y se puede seguir con el calculo

xvii. Se calcula el diámetro medio

$$D_m = C \times D_w \qquad (2\text{-}63)$$

xviii. Calcule el Diámetro Exterior (*OD*)

$$OD = D_m + D_w \qquad (2\text{-}64)$$

Si el resorte se instala en un orificio, para evitar fricción se debe cumplir que:

$$D_{orificio} > OD + \frac{D_w}{10} \qquad (2\text{-}65)$$

Se calcula el diámetro del orificio que se sugiere para que entre el resorte.

$$D_{orificio} = OD + \frac{D_w}{2} \qquad (2\text{-}66)$$

xix. Calcule el Diámetro Interior

$$ID = D_m - D_w \qquad (2\text{-}67)$$

Si el resorte se instala en una varilla, para evitar fricción se debe cumplir que:

$$D_{varilla} < ID - \frac{D_w}{10} \qquad (2\text{-}68)$$

xx. Verifique el resorte en cuanto al pandeo

Para verificar el resorte en cuanto a pandeo se utiliza la Figura II-30 según el tipo de sujeción en los extremos. Si la relación real f_o/L_f es mayor que la Razón crítica en el gráfico, el resorte se pandeará. La Razón Crítica en el gráfico depende de L_f/D_m.

Hasta este punto se debe estar conforme con la geometría del resorte para continuar con el análisis de tensión.

xxi. Calcule el factor de Wahl (K)

$$K = \frac{4C-1}{4C-4} + \frac{0.615}{C} \qquad (2\text{-}69)$$

xxii. Calcule la tensión real que se espera debido a la fuerza de operación, F_o, a partir de la ecuación 4-4.

$$\tau_o = \frac{8KCF_o}{\pi D_w^2} \qquad (2\text{-}70)$$

Si $\tau_o < \tau_d$ El diseño es seguro.

xxiii. Calcule la longitud comprimido, L_s.

$$L_s = D_w N \qquad (2\text{-}71)$$

El número de bobinas, N, depende del tipo de extremo del resorte:

- Resortes con extremos lijados y a escuadra: $N = N_a + 2$
- Resortes con sólo extremos a escuadra: $N = N_a + 2$
- Resortes con bobinas en bruto: $N = N_a$
- Resortes con bobinas en bruto con extremos lijados: $N = N_a + 1$

Si $L_s < L_o$ se sigue calculando. En caso contrario, se presentó una situación imposible.

xxiv. Calcule la fuerza en el resorte en longitud comprimido, F_s, como el producto de la razón del resorte por la deflexión en longitud de comprimido $(L_f - L_s)$

$$F_s = k(L_f - L_s) \qquad (2\text{-}72)$$

xxv. Calcule el esfuerzo o tensión en el resorte en longitud comprimido, τ_s.

La tensión o esfuerzo en el resorte es directamente proporcional a la fuerza, podemos considerar para la tensión en longitud comprimido:

$$\tau_s = \tau_o \frac{F_s}{F_o} \qquad (2\text{-}73)$$

Si $\tau_s < \tau_{max}$, el resorte no presentará cedencia.

xxvi. Verifique el margen para las bobinas, *cc*:

$$cc = \frac{L_o - L_s}{N_a} \qquad (2\text{-}74)$$

Si $cc > \frac{D_w}{10}$, el margen se considera aceptable.

Si no es aceptable se puede disminuir el número de bobinas o aumentar el diámetro del alambre.

2.3.13. REPRESENTACIÓN DE NÚMEROS EN JAVA

Los números se almacenan en las variables. Una variable representa un trozo de la memoria del computador. La memoria está formada por una gran cantidad de bytes y cada byte está constituido por 8 bits. Un bit puede almacenar un 1 o un 0.

2.3.13.1. Enteros

Una variable entera (int) está formada por 4 bytes, es decir 32 bits. Estos 32 bits representan el valor almacenado por esa variable en binario. Por ejemplo:

00000000	00000000	00000000	00000101
1er. byte	2do. byte	3er. byte	4to. byte

Figura II-32. Entero en binario.

El valor representado por esta variable es:

1*2^2 + 0*2^1 + 1*2^0

En donde x^y se usa acá como una abreviación de x elevado a y.

En general, una variable entera x está formada por 32 bits que denotaremos x31, x30, ... , x2, x1 y x0. El valor numérico representado por la variable x está dado por el siguiente cálculo:

- Si x31 es 0, el valor es positivo y se calcula como:

  x31*2^31 + x30*2^30 + ... + x2*2^2 +x1*2^1+ x0*2^0

- Si x31 es 1, el valor es negativo y se calcula construyendo una nueva palabra y, tal que:

  yi= 1 si xi==0

  0 si xi==1

  valor(x) = - (valor(y) + 1)

  Se dice que y es el complemento de x. Este tipo de representación se llama Reprsentación de Complemento a 2.

Ejemplos:

- valor(000...001001) = 1*2^3+1*2^0= 9
- valor(111...111010) = - (valor(000...000101)+1) = - (5+1) = -6

Una variable entera (int) siempre utiliza 32 bits, aún cuando el número sea pequeño. Por otra parte, no es capaz de almacenar números demasiado grandes (en valor absoluto).

Valores máximos y mínimos

- Máximo= valor(011...111111)= 2^31-1
- Mínimo= valor(100...000000)= -2^31

Por lo tanto, con un int se pueden almacenar números de 9 dígitos aproximadamente.

Observación: usualmente se usa el tipo int para almacenar enteros, pero también existen otros tipos que también almacenan enteros pero en menos o más bits. La Tabla V-1 muestra los tipos enteros presentes en Java.

Tabla II-5. Tipos de Enteros en Java.

tipo	número de bits	rango representado
int	32	[-2^31,2^31-1]
short	16	[-2^15,2^15-1]

byte	8	[-2^7,2^7-1]
long	64	[-2^63,2^63-1]

Fuente:Elaboración propia con datos de dominio público

2.3.13.2. Reales

Representación en punto flotante

Un número real se almacena en una variable especificando 3 componentes: el signo (+ o -), la mantisa y el exponente. Por ejemplo, el número 11 se representa como:

- el signo: +
- mantisa: 10110000...
- exponente: 4

La mantisa es una secuencia de bits que siempre comienza en 1. El exponente indica en donde colocar el punto que separa la parte entera de la fraccionaria. Un valor 0 indica que el punto se coloca justo antes del primer bit. Un valor 4 indica que se coloca después del cuarto bit. Un valor -3 indica que hay que imaginar que el número va precedido por 0.000 y luego viene la mantisa.

¿Qué número es el siguiente?

- el signo: -
- mantisa: 110010000...
- exponente: 7

-1100100.00... , que es -(4+32+64)= -100

Otros ejemplos:

signo	mantisa	exponente	en binario	en decimal
+	1000...	100	1 seguido de 99 ceros	2^99

+	1010...	0	0.1010	0.5+0.125 (0.625)
+	1000...	-3	0.0001	2^(-4) (0.0625)

En una variable de tipo double se destina 1 bit para el signo, 11 bits para el exponente y 52 bits para la mantisa.

Valores máximos y mínimos

- Máximo en valor absoluto: 2^1023
- Valor más pequeño: 2^(-1024)

La ventaja de esta representación es que se alcanza un rango de representación mucho más grande de lo que sería posible con 64 bits.

Precisión y error de representación

La desventaja es que los números se almacenan con una precisión limitada. Esto significa que un número que requiera 100 bits de mantisa será aproximado a uno que solo ocupa 52, introduciéndose entonces un error de representación.

El error que se comente se puede visualizar mejor en base 10. El error que comete es como cuando el siguiente número:

12345678901234567890123456789012345678901234567890

se aproxima al siguiente valor:

1234567890123456*10^25

Es decir, solo se reprentan 15 dígitos de los 40 iniciales. El error absoluto que se comete es:

67890123456789012345678901234567890

que parece enorme, pero lo que importa es el error relativo que es ~10^(-15) que es bajísimo y para la mayoría de las aplicaciones numéricas es insignificante. Pero es un error que conviene tener presente.

Además del tipo double existe el tipo float como se muestra en la tabla V-2.

Tabla II-6. Tipos en punto flotante en Java.

Tipo	tamaño en bits	mantisa en bits	exponente en bits	rango representado	precisión en dígitos
double	64	52	11	~ [-10^300,10^300]	~ 15
float	32	24	7	~ [-10^33,10^33]	~ 7

Fuente:Elaboración propia con datos de dominio público

Conceptos esenciales:

- El rango de representación de los enteros es limitado. Si el resultado de una operación aritmética (como la multiplicación) excede el rango de representación se producirá un error en la magnitud del valor almacenado. Lamentablemente, Java no alerta cuando se comete este tipo de errores.
- El rango de representación de los reales es casi ilimitado, pero los números se almacenan con una precisión limitada (15 dígitos para una variable de tipo double). Esto hace que al realizar multiplicaciones y divisiones, los valores que se obtienen poseen un error de precisión, que no es relevante para la mayoría de las aplicaciones numéricas.

El estándar IEEE para la aritmética en coma flotante

El IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) ha creado un estándar la presentación de números en coma flotante. Este estándar especifica como deben representarse los números en coma flotante con simple precisión (32 bits) o doble precisión (64 bits), y también cómo deben realizarse las operaciones aritméticas con ellos.

Simple Precisión

El estándar IEEE-754 para la representación en simple precisión de números en coma flotante exige una cadena de 32 bits. El primer bit es el bit de signo (S), los siguientes 8 son los bits del exponente (E) y los restantes 23 son la mantisa (M):

```
S EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM
0 1      8 9                     31
```

El valor V representado por esta cadena puede ser determinado como sigue:

- Si E=255 y M es no nulo, entonces V=NaN ("Not a number")
- Si E=255 y M es cero y S es 1, entonces V=-Infinito
- Si E=255 y M es cero y S es 0, entonces V=Infinito
- Si 0<E<255 entonces V=(-1)^S * 2^(E-127) * (1.M)

  donde "1.M" se emplea para representar el número binario creado por la anteposición a M de un 1 y un punto binario.
- Si E=0 y M es no nulo, entonces V=(-1)^S * 2^(-126) * (0.M)

  Estos son valores "sin normalizar".
- Si E=0 y M es cero y S es 1, entonces V=-0
- Si E=0 y M es cero y S es 0, entonces V=0

En particular,

```
0 00000000 00000000000000000000000 = 0
1 00000000 00000000000000000000000 = -0

0 11111111 00000000000000000000000 = Infinito
1 11111111 00000000000000000000000 = -Infinito

0 11111111 00000100000000000000000 = NaN
1 11111111 00100010001001010101010 = NaN

0 10000000 00000000000000000000000 = +1 * 2^(128-127) * 1.0 = 2
0 10000001 10100000000000000000000 = +1 * 2^(129-127) * 1.101 = 6.5
1 10000001 10100000000000000000000 = -1 * 2^(129-127) * 1.101 = -6.5
0 00000001 00000000000000000000000 = +1 * 2^(1-127) * 1.0 = 2^(-126)
0 00000000 10000000000000000000000 = +1 * 2^(-126) * 0.1 = 2^(-127)
0 00000000 00000000000000000000001 = +1 * 2^(-126) *
                                     0.00000000000000000000001 =
                                     2^(-149) (valor positivo más pequeño)
```

Doble precisión

El estándar IEEE-754 para la representación en doble precisión de números en coma flotante exige una cadena de 64 bits. El primer bit es el bit de signo (S), los siguientes 11 son los bits del exponente (E) y los restantes 52 son la mantisa (M):

```
S EEEEEEEEEEE MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
0 1          11 12                                                 63
```

El valor V representado por esta cadena puede ser determinado como sigue:

- Si E=2047 y M es no nulo, entonces V=NaN ("Not a number")
- Si E=2047 y M es cero y S es 1, entonces V=-Infinito
- Si E=2047 y M es cero y S es 0, entonces V=Infinito
- Si 0<E<2047 entonces V=(-1)**S * 2 ** (E-1023) * (1.M). donde "1.M" se emplea para representar el número binario creado por la anteposición a M de un 1 y un punto binario.
- Si E=0 y M es no nulo, entonces V=(-1)**S * 2 ** (-1022) * (0.M). Estos son valores "sin normalizar".
- Si E=0 y M es cero y S es 1, entonces V=-0

- Si E=0 y M es cero y S es 0, entonces V=0

- Si E=2047 y M es no nulo, entonces V=NaN ("Not a number")
- Si E=2047 y M es cero y S es 1, entonces V=-Infinito
- Si E=2047 y M es cero y S es 0, entonces V=Infinito
- Si 0<E<2047 entonces V=(-1)**S * 2 ** (E-1023) * (1.M). donde "1.M" se emplea para representar el número binario creado por la anteposición a M de un 1 y un punto binario.
- Si E=0 y M es no nulo, entonces V=(-1)**S * 2 ** (-1022) * (0.M). Estos son valores "sin normalizar".
- Si E=0 y M es cero y S es 1, entonces V=-0

- Si E=0 y M es cero y S es 0, entonces V=0

CAPÍTULO III

COMPONETE AGJAVA APLICADO AL ALGORITMO GENÉTICO PARA LA OPTIMIZACIÓN DEL DISEÑO DE RESORTES HELICOIDALES DE COMPRESIÓN

3.1. CONSIDERACIONES DE IMPLEMENTACIÓN

En un problema de diseño de ingeniería que incluye el diseño de componentes mecánicos, la meta es ya sea minimizar o maximizar un objetivo diseño y simultáneamente satisfacer un número de restricciones que se representan como ecuaciones e inecuaciones. En el problema de diseño de resortes helicoidales de compresión el objetivo es minimizar el volumen y satisfacer las condiciones referentes al diseño de un resorte. Las variables en el diseño de resortes helicoidales de compresión involucra la combinación de variables reales y discretas.

Los algoritmos genéticos trabajarán eficientemente si tenemos ciertas clases útiles para:

- Generar números aleatorios.
- Manejar los números a nivel de bits para la representación de enteros y de acuerdo al estándar IEEE 754 para los números en punto flotante.
- Crear listas ordenadas.

Las clases anteriores servirán de soporte para implementar las clases necesarias para un algoritmo genético, estas nuevas clases sirven para:

- Crear un cromosoma y manipularlos.

- Representar un operador genético y manipular un cromosoma., además de implementar la variedad existente de operadores genéticos.
- Crear una población de cromosomas y manipularlos.
- Interpretar los cromosomas a través de vistas.

Es necesaria también una clase para representar un resorte helicoidal de compresión.

Con todas estas clases se puede implementar de manera fácil un algoritmo para optimizar el diseño de resortes helicoidales de compresión basada en algoritmos genéticos. Sin problemas variantes de este programa pueden ser implementados fácilmente para otros problemas de diseño.

Se emplearon las técnicas de la Programación Orientada a Objetos para crear cada una de las clases. Para la codificación del algoritmo se usó el lenguaje de Programación Java por su portabilidad y su disponibilidad.

3.2. APLICACIÓN DE METODOLOGÍA DE DESARROLLO BASADO EN COMPONENTES

Esta parte representa la descripción de cada uno de los artefactos generados en cada una de las actividades propuestas en esta metodología y se muestran solamente los artefactos necesarios para el buen entendimiento del Componente AGJava y del Algoritmo Genético para la optimización del Diseño de Resortes Helicoidales de Compresión.

3.2.1. FASE DE INCEPCIÓN

En esta fase se ejecutaron las siguientes actividades:

- Descripción del Negocio
- Identificar Procesos de Negocio
- Clasificar y Ranquear Procesos de Negocio
- Definir la Arquitectura Preliminar

- Refinar el Proceso de Negocio
- Realizar el Envisioning (Visión) del Sistema

3.2.2. FASE DE ELABORACIÓN

Durante la fase de elaboración del sistema se ejecutaron las siguientes actividades:

- Detectar Actores y Casos de Uso
- Realizar el Modelo Conceptual del Negocio
- Refinar Casos de Uso
- Clasificar y Ranquear Casos de Uso
- Refinar Caso de Uso Crítico
- Identificar Componentes
    - Identificar Interfaces y Operaciones del Sistema
    - Definir el Modelo de Tipos del Negocio
    - Definir Interfaces del Negocio
    - Identificar Interfaces de Sistemas Existentes
    - Crear Especificación y Arquitectura Inicial
- Interacción de Componentes
    - Refinar Interfaces y Operaciones de Sistema
    - Descubrir Operaciones de Interfaces de Negocio
    - Definir Políticas de Manejo de Integridad Referencial
    - Refinar la Arquitectura

Se clasificaron las clases como clases borde, control y entidad, como se muestran en las figuras III-1 a III -3

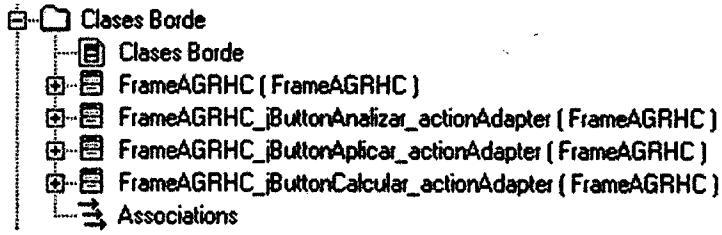


Figura III-1. Clases Borde

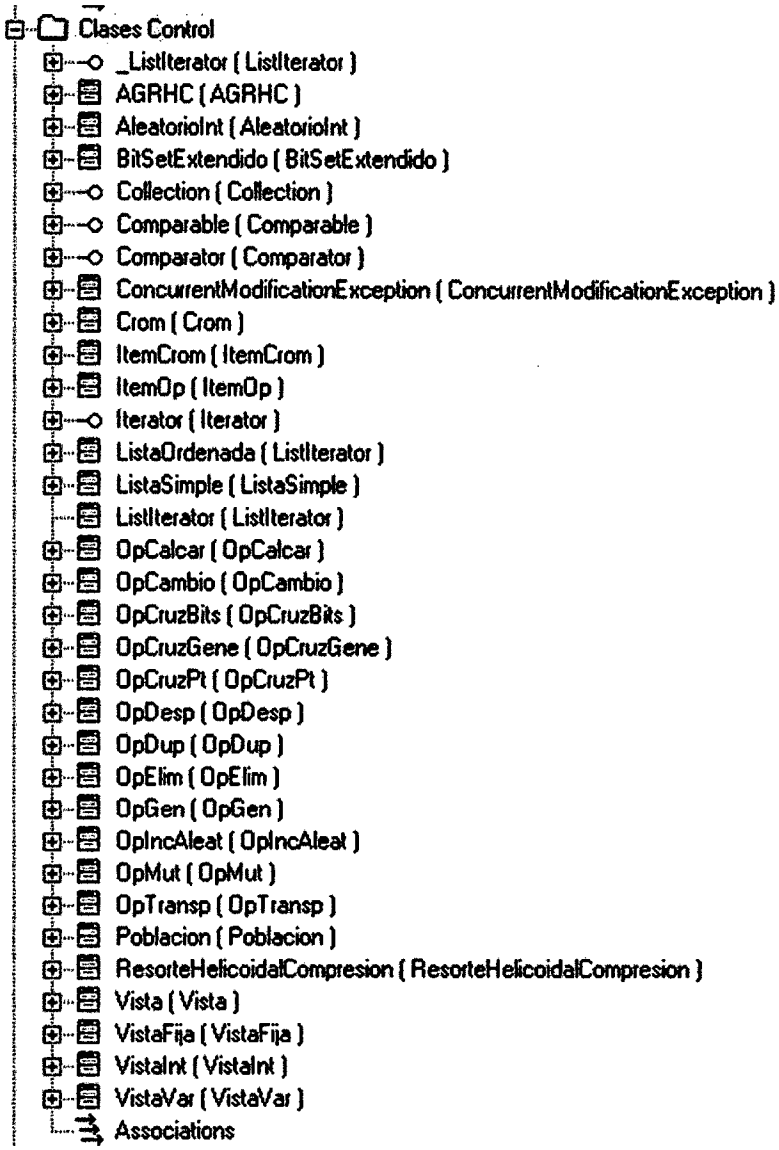


Figura III-2. Clases Control

Figura III-3. Clases Entidad

Durante esta etapa se definieron las relaciones entre las clases de análisis que se muestra en la Figura III-4

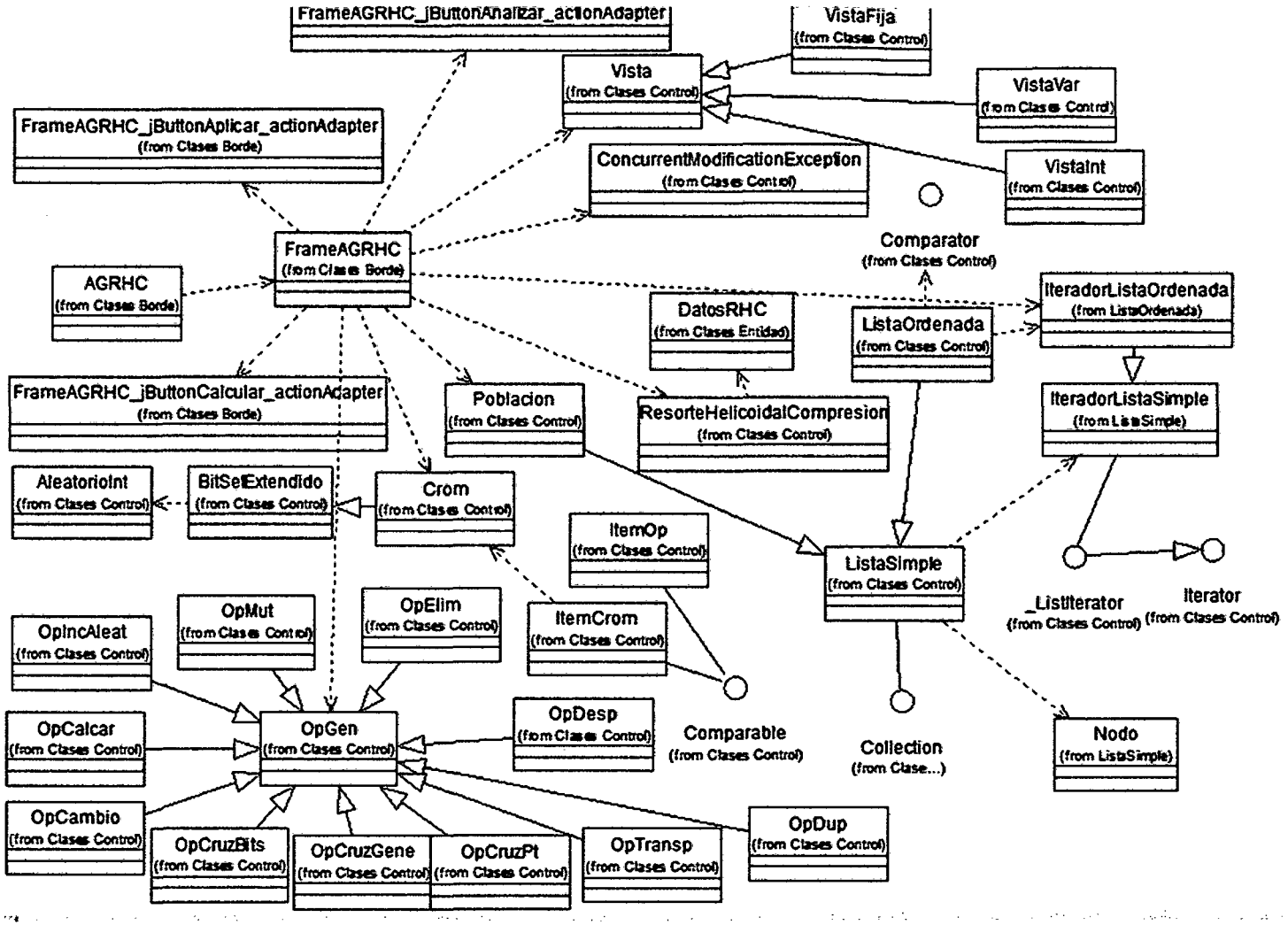


Figura III-4. Clases de Análisis.

Las clases se organizan en tres paquetes. (Figura III-5).

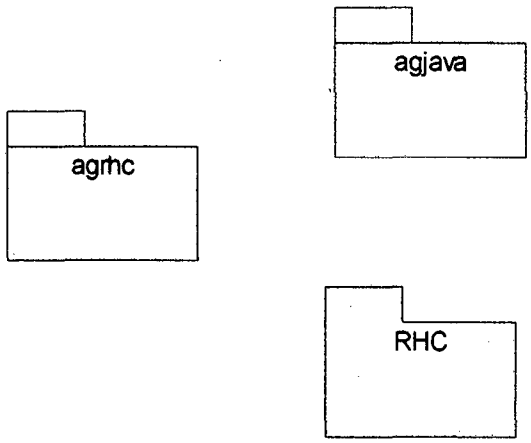


Figura III-5. Organización en paquetes

En la figura III-6 y III-7 se muestra la pertenencia de cada una de las clases a los paquetes. El paquete agjava (Figura III – 7) se convertirá en el componente agjava.rar y será un componente reusable y extensible para quien quiera implementar algoritmos genéticos independientemente del ambito de solución.

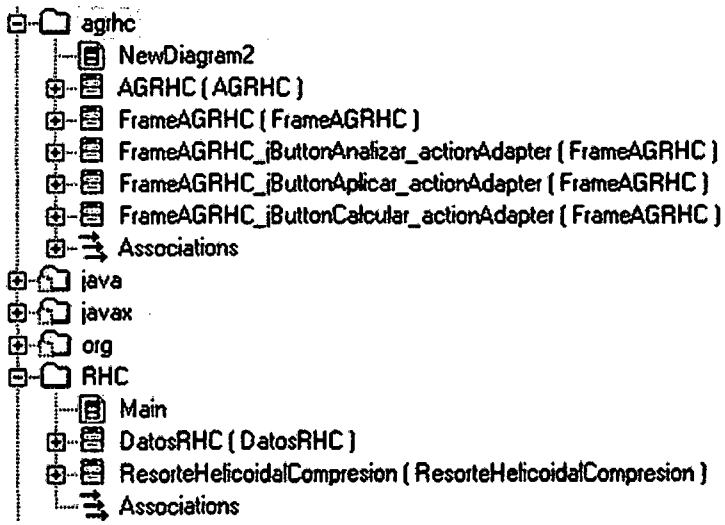


Figura III-6. Clases de los paquetes agrhc y rhc

El componente agjava constará de dos paquetes, el **paquete agjava.ag** comprende todas las clases relacionadas a posibilitar la implementación de

los algortimos genéticos mientras que el **paquete agjava.util** tiene clases utilitarias que sirven de apoyo al paquete agjava.ag.

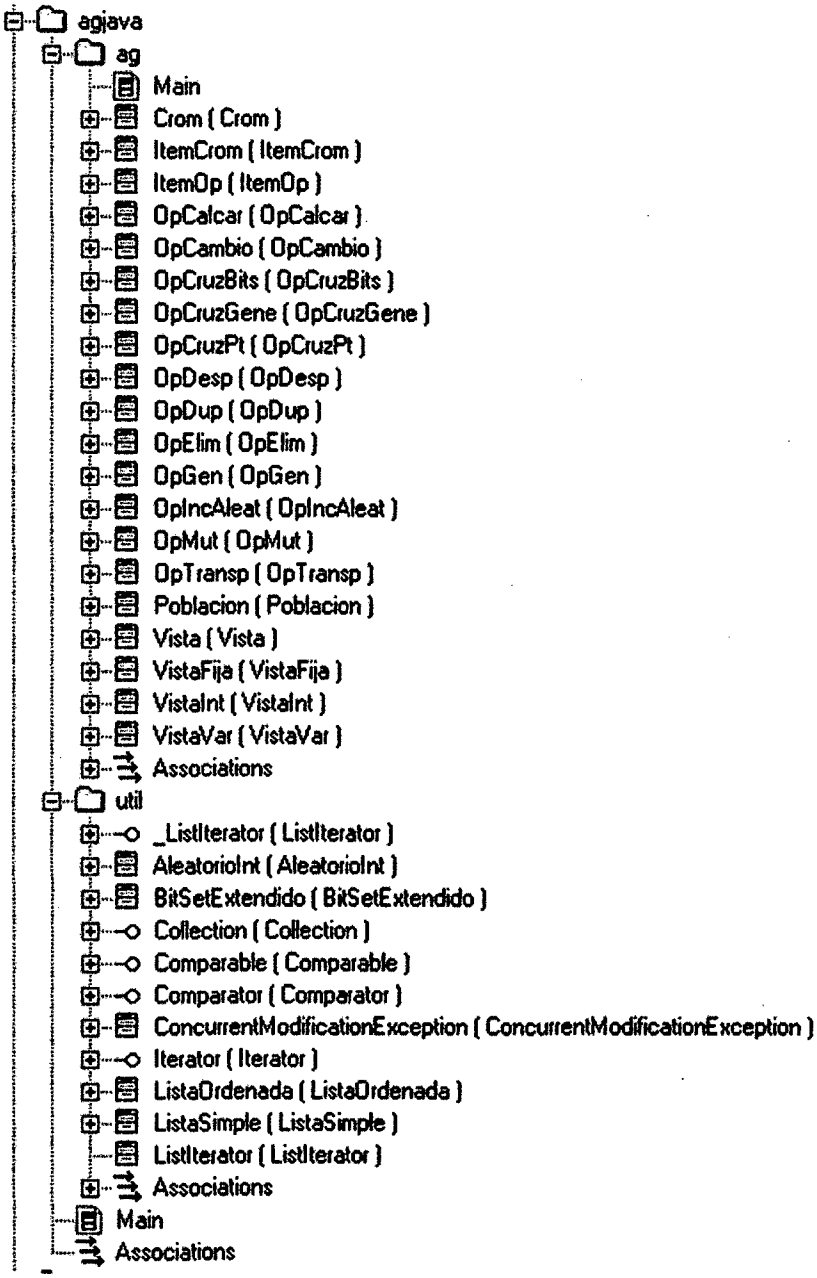
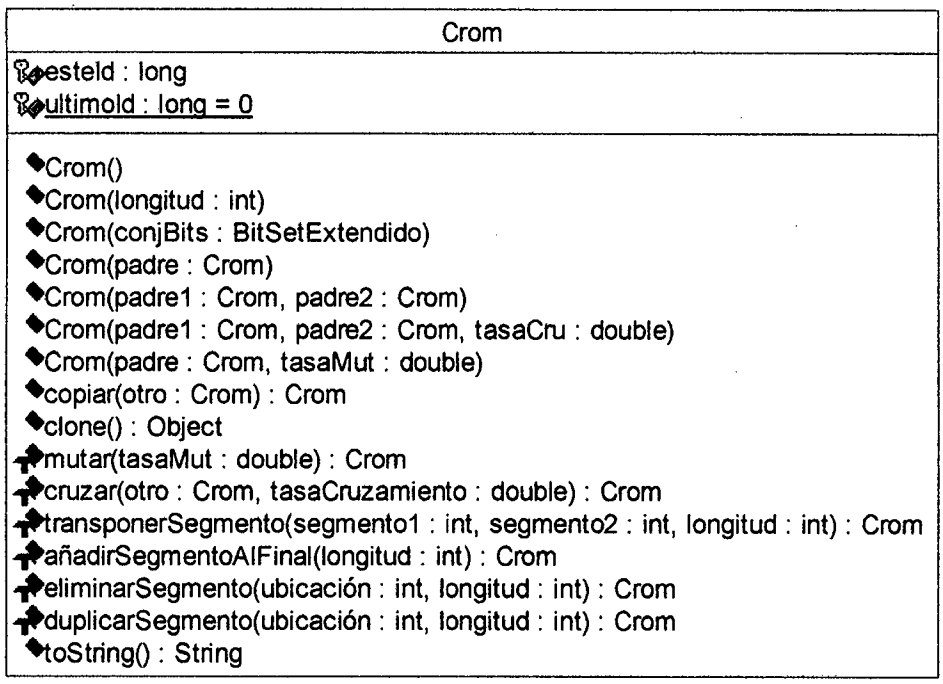
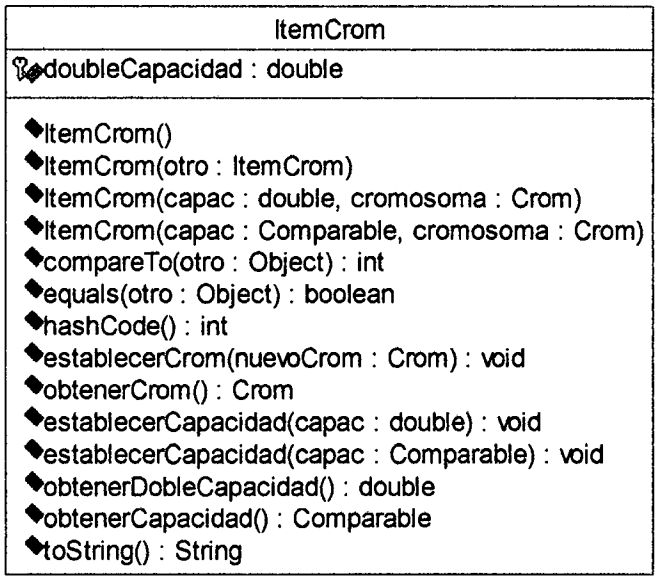
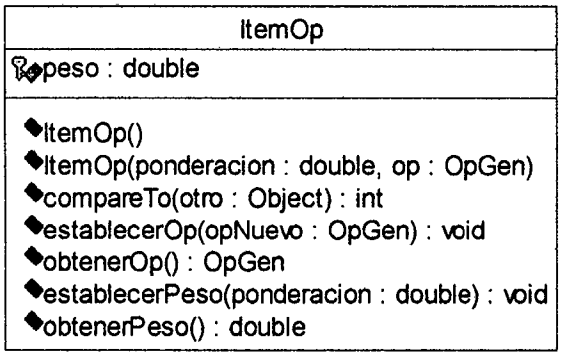


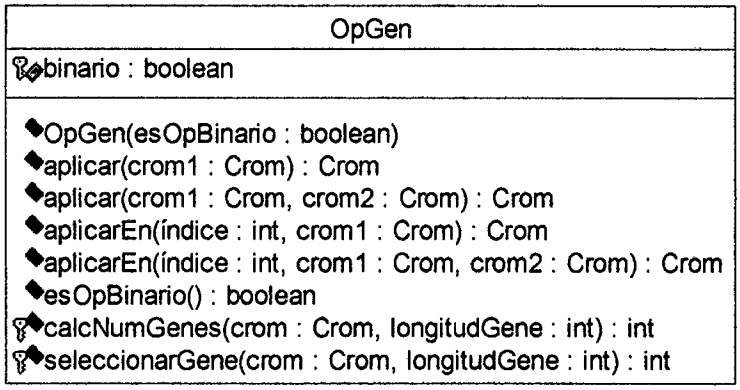
Figura III-7. Clases del componente agjava

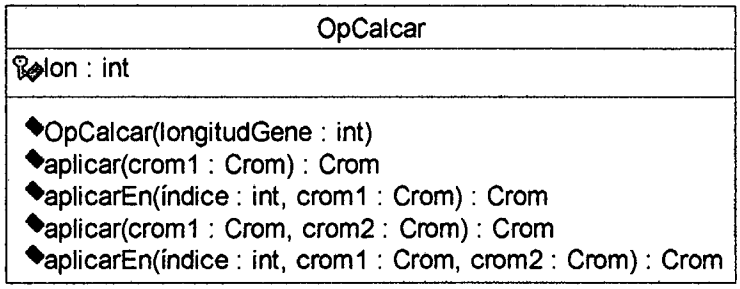
Clases de Diseño del paquete agjava.ag:

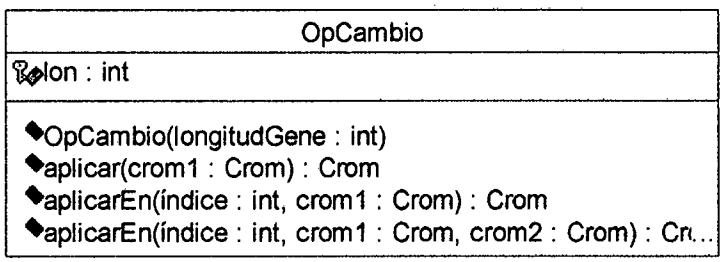


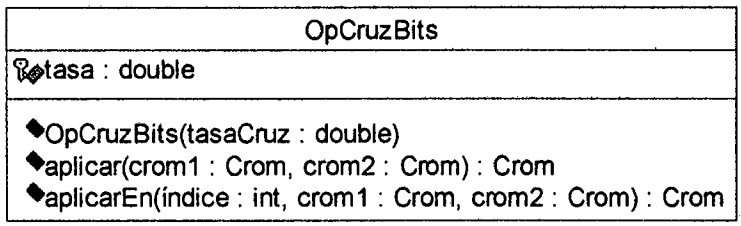


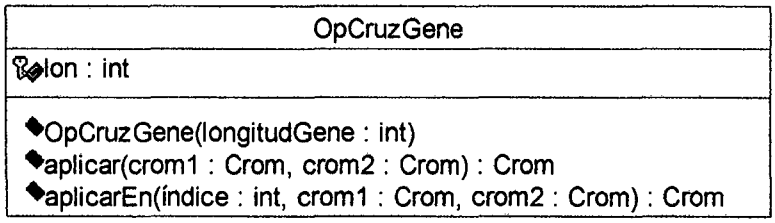

ItemOp
peso : double
ItemOp()
ItemOp(ponderacion : double, op : OpGen)
compareTo(otro : Object) : int
establecerOp(opNuevo : OpGen) : void
obtenerOp() : OpGen
establecerPeso(ponderacion : double) : void
obtenerPeso() : double

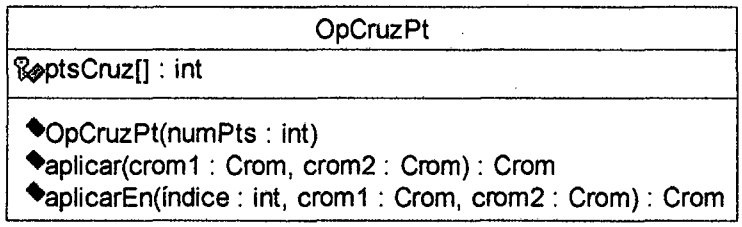

OpGen
binario : boolean
OpGen(esOpBinario : boolean)
aplicar(crom1 : Crom) : Crom
aplicar(crom1 : Crom, crom2 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom, crom2 : Crom) : Crom
esOpBinario() : boolean
calcNumGenes(crom : Crom, longitudGene : int) : int
seleccionarGene(crom : Crom, longitudGene : int) : int

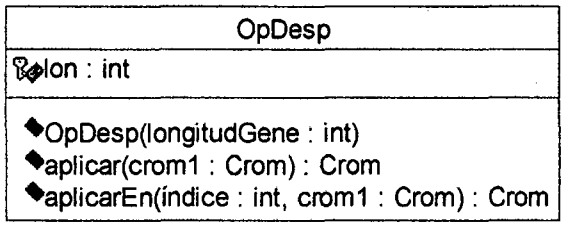

OpCalcar
lon : int
OpCalcar(longitudGene : int)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom
aplicar(crom1 : Crom, crom2 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom, crom2 : Crom) : Crom

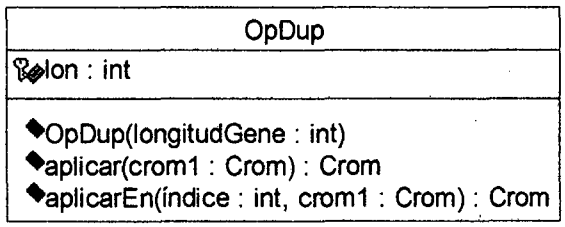

OpCambio
lon : int
OpCambio(longitudGene : int)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom, crom2 : Crom) : Cr...

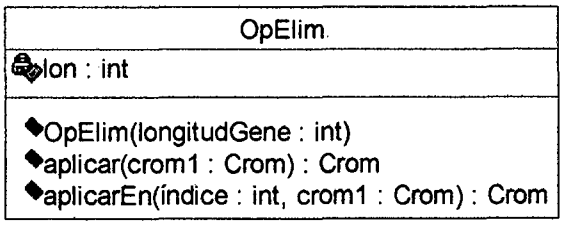

OpCruzBits
tasa : double
OpCruzBits(tasaCruz : double)
aplicar(crom1 : Crom, crom2 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom, crom2 : Crom) : Crom

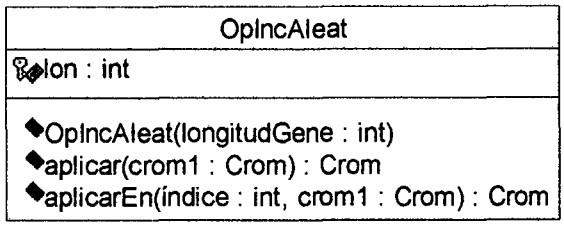

OpCruzGene
lon : int
OpCruzGene(longitudGene : int)
aplicar(crom1 : Crom, crom2 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom, crom2 : Crom) : Crom

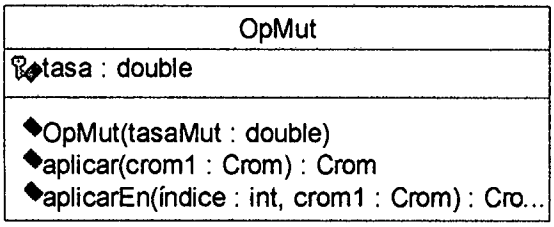

OpCruzPt
ptsCruz[] : int
OpCruzPt(numPts : int)
aplicar(crom1 : Crom, crom2 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom, crom2 : Crom) : Crom

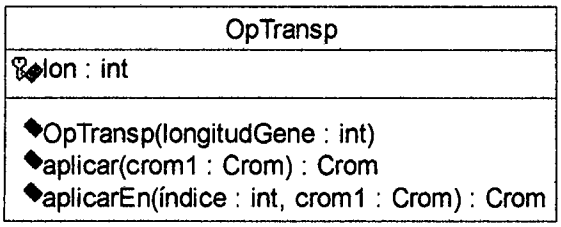

OpDesp
lon : int
OpDesp(longitudGene : int)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom

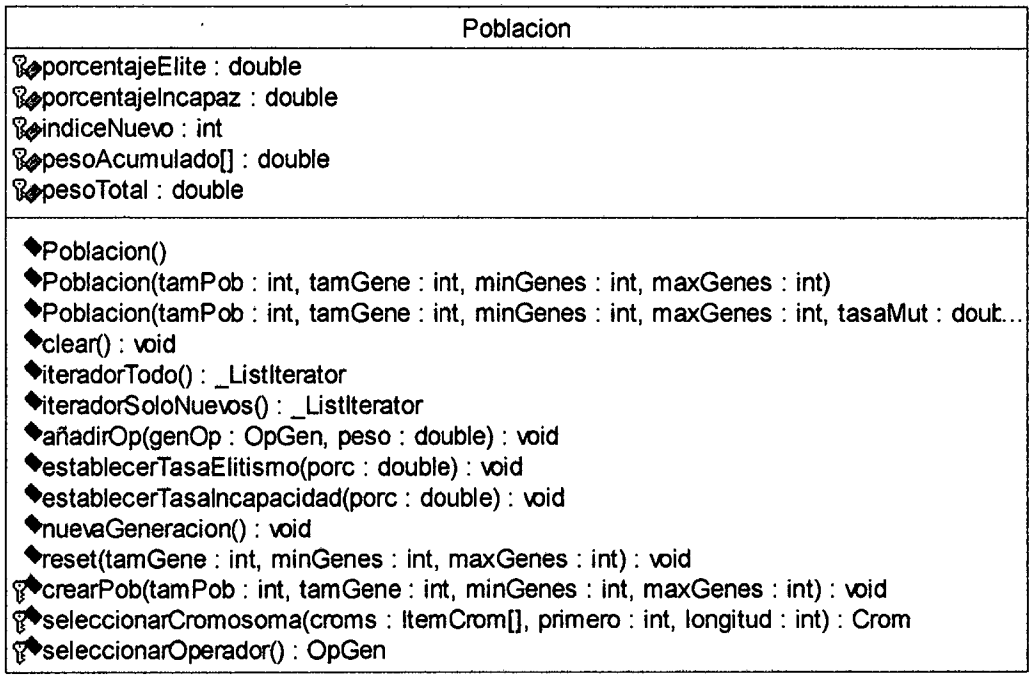

OpDup
lon : int
OpDup(longitudGene : int)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom

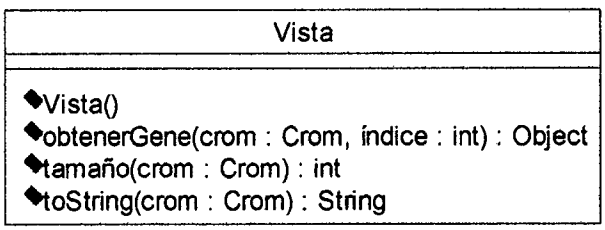

OpElim
lon : int
OpElim(longitudGene : int)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom

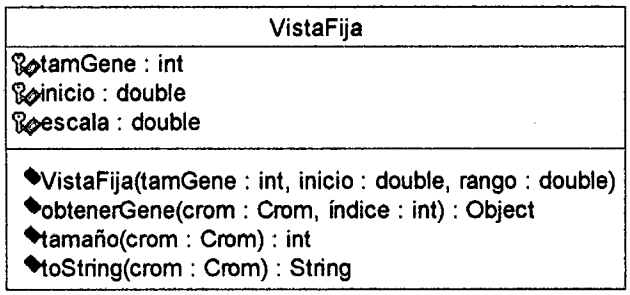

OpIncAleat
lon : int
OpIncAleat(longitudGene : int)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom

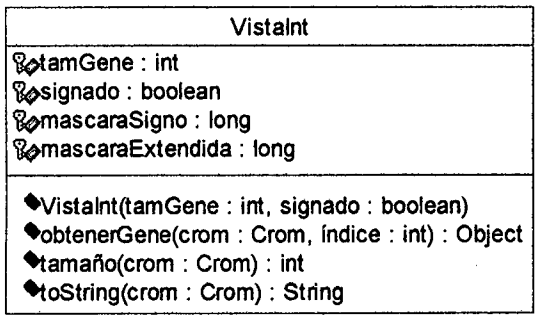

OpMut
tasa : double
OpMut(tasaMut : double)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Cro...


OpTransp
lon : int
OpTransp(longitudGene : int)
aplicar(crom1 : Crom) : Crom
aplicarEn(índice : int, crom1 : Crom) : Crom

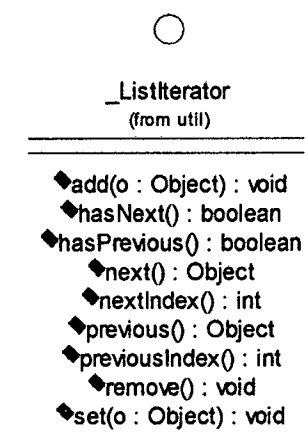

Poblacion
porcentajeElite : double
porcentajeIncapaz : double
indiceNuevo : int
pesoAcumulado[] : double
pesoTotal : double
Poblacion()
Poblacion(tamPob : int, tamGene : int, minGenes : int, maxGenes : int)
Poblacion(tamPob : int, tamGene : int, minGenes : int, maxGenes : int, tasaMut : doub...
clear() : void
iteradorTodo() : _ListIterator
iteradorSoloNuevos() : _ListIterator
añadirOp(genOp : OpGen, peso : double) : void
establecerTasaElitismo(porc : double) : void
establecerTasaIncapacidad(porc : double) : void
nuevaGeneracion() : void
reset(tamGene : int, minGenes : int, maxGenes : int) : void
crearPob(tamPob : int, tamGene : int, minGenes : int, maxGenes : int) : void
seleccionarCromosoma(croms : ItemCrom[], primero : int, longitud : int) : Crom
seleccionarOperador() : OpGen

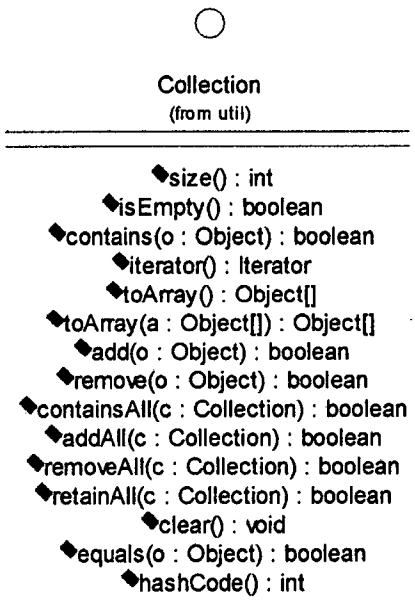

Vista
Vista()
obtenerGene(crom : Crom, índice : int) : Object
tamaño(crom : Crom) : int
toString(crom : Crom) : String

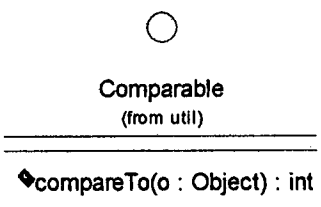


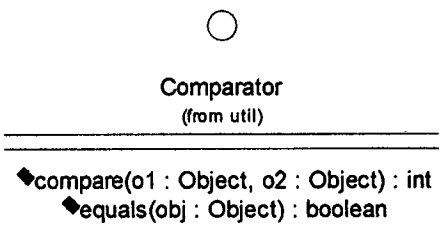


Interfaces del paquete agjava.util

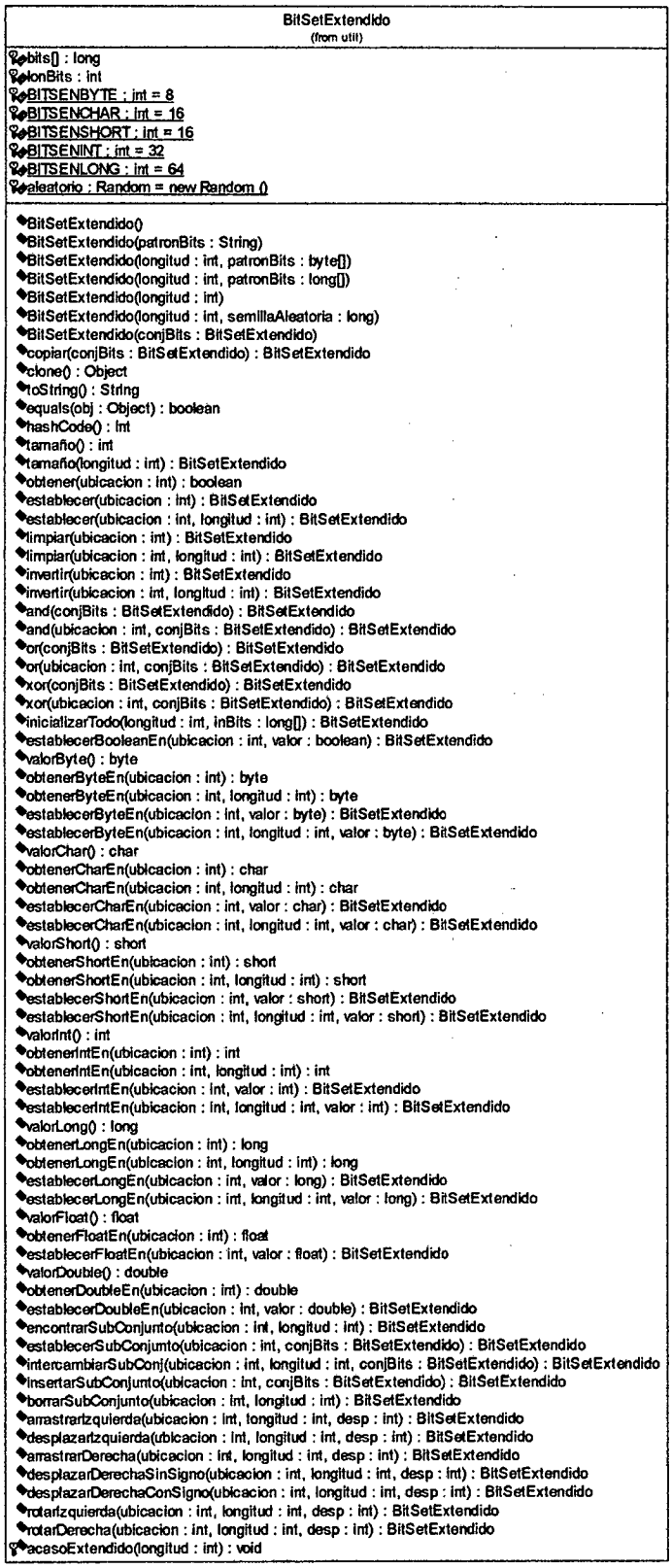


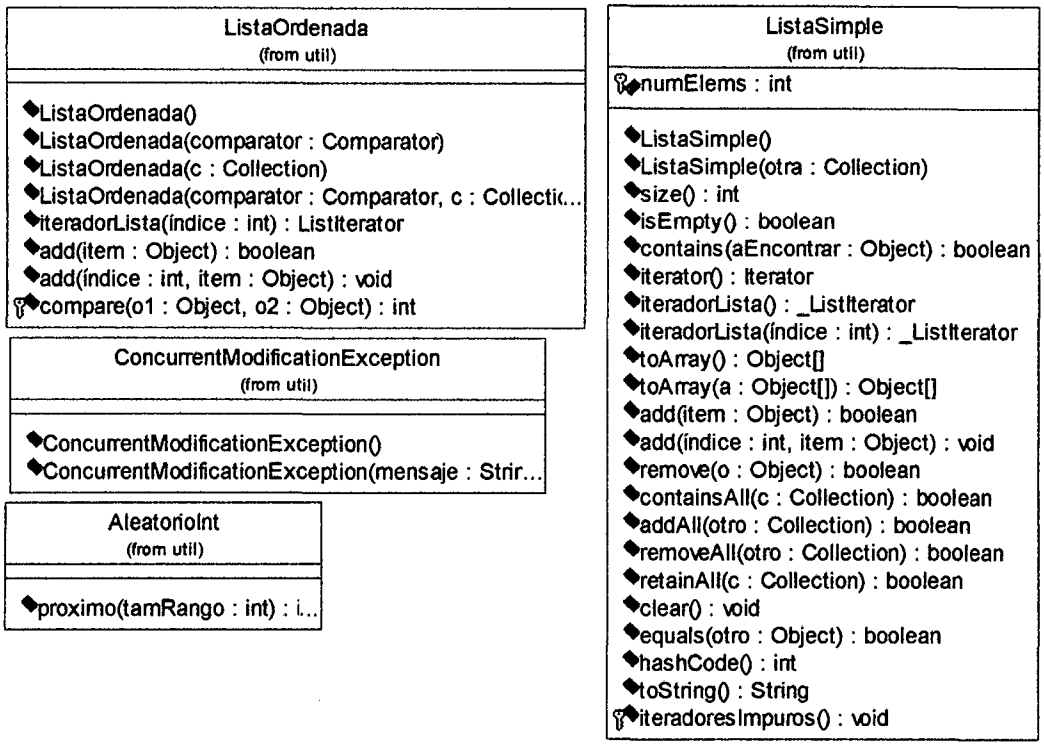




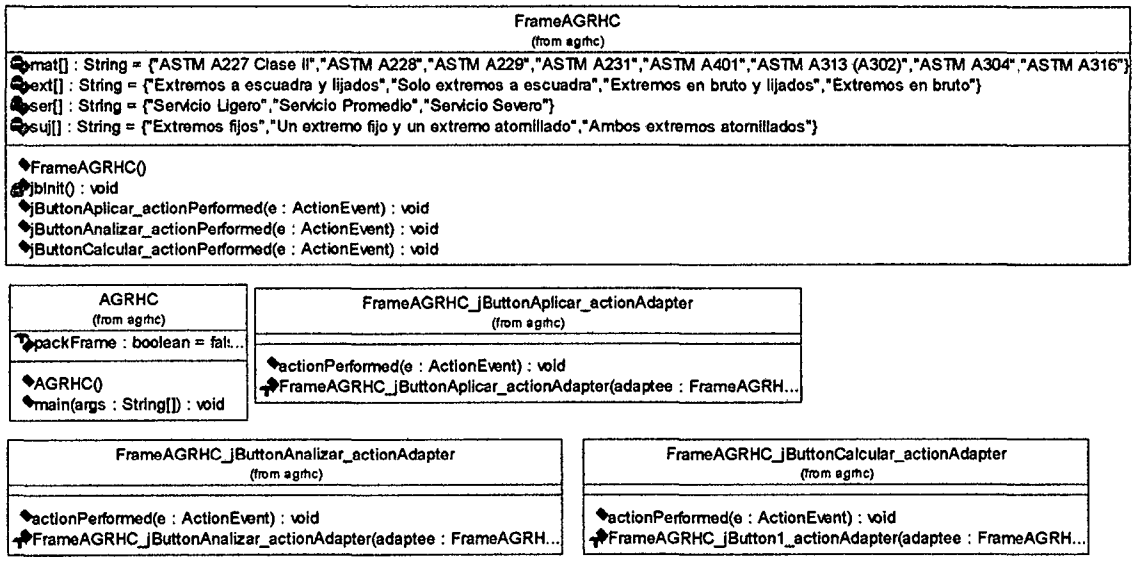


Clases de Diseño del paquete agjava.util:





Clases de Diseño del paquete agrhc:



Clases de Diseño del paquete RHC

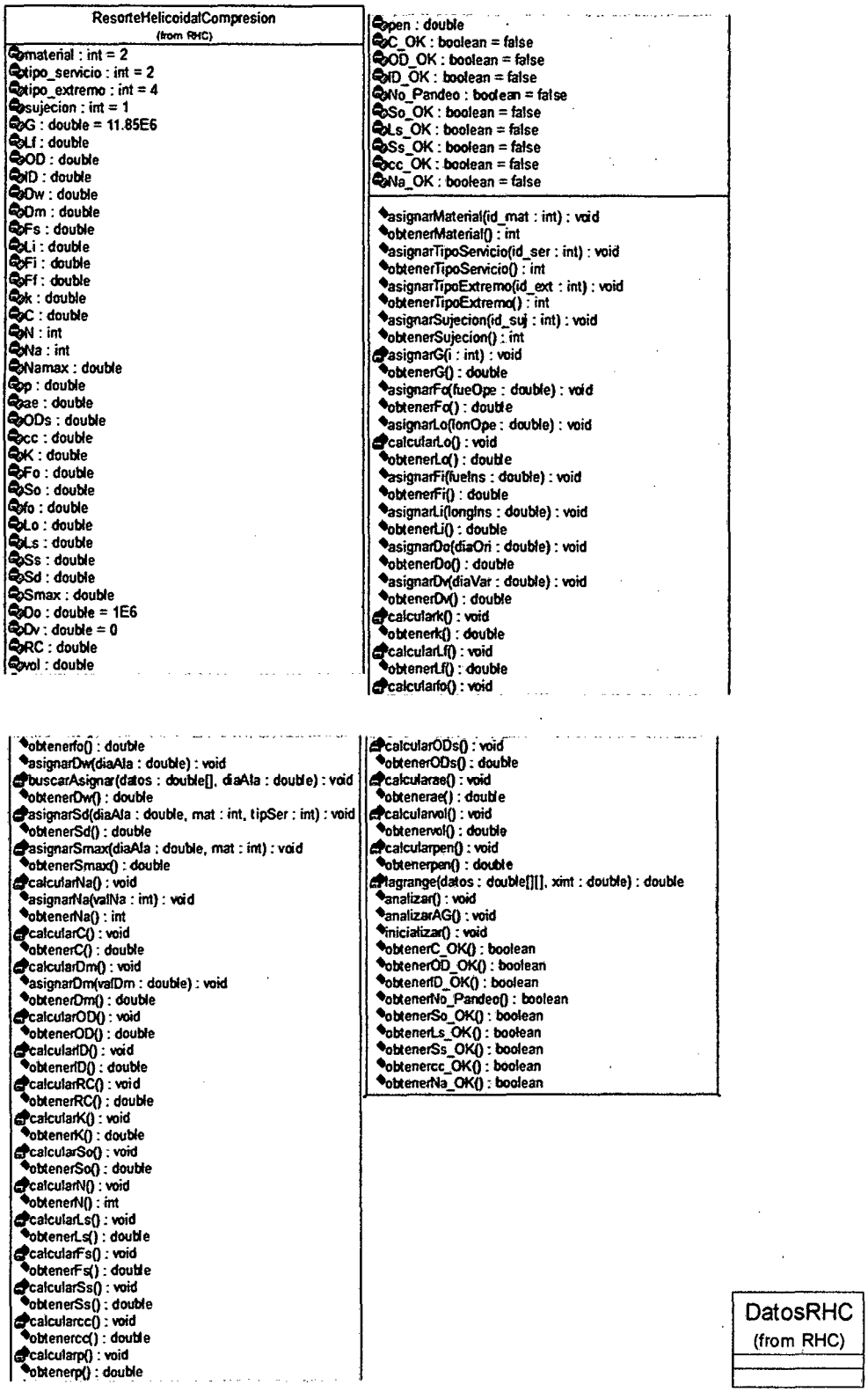


Diagrama de Componentes Core donde se puede observar la dependencia de los componentes para lograr la funcionalidad completa de la aplicación que permita optimizar el diseño de resortes helicoidales de compresión:

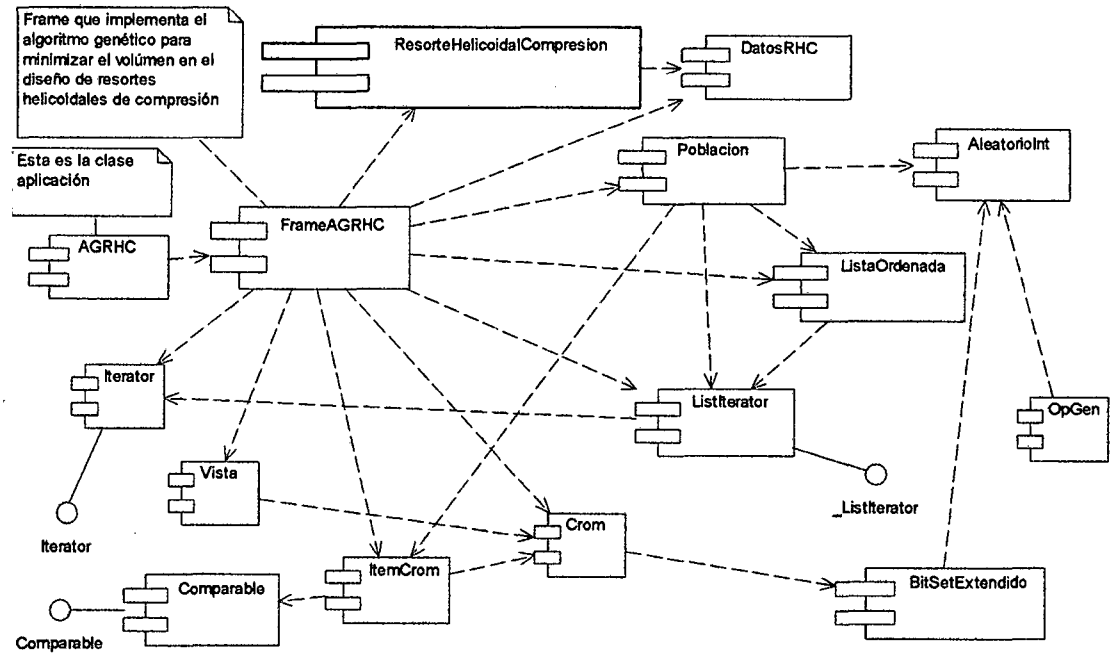


Diagrama de componentes con los paquetes creados y su dependencia:

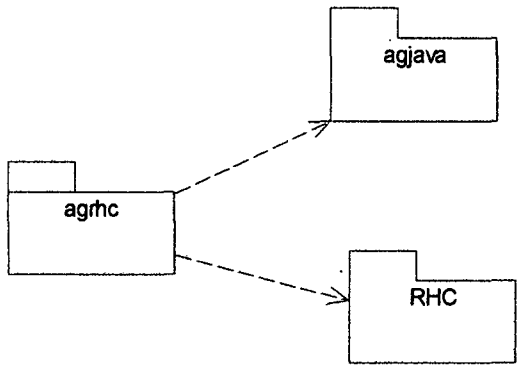


Diagrama de Componentes del paquete agrhc:

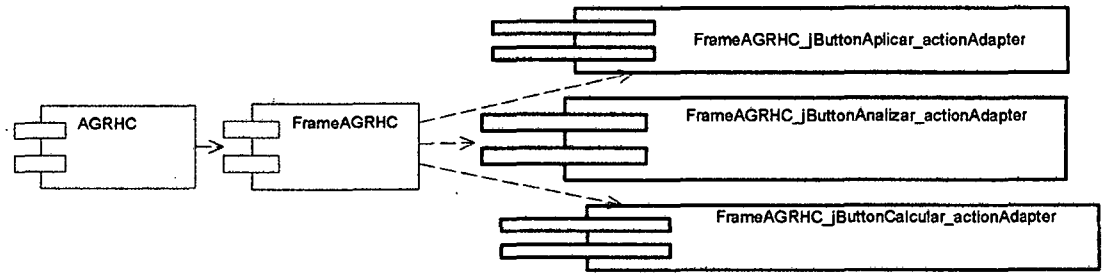
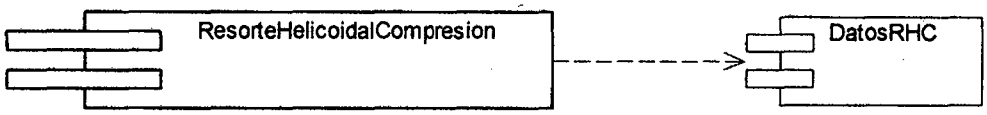


Diagrama de componentes del paquete rhc



El componente agjava es el componente principal de esta tesis. Podemos ver el diagrama de componentes del paquete agjava que contiene dos paquetes, util y ag:

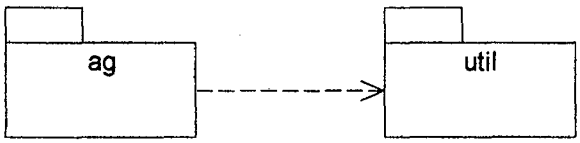


Diagrama de componentes del paquete útil:

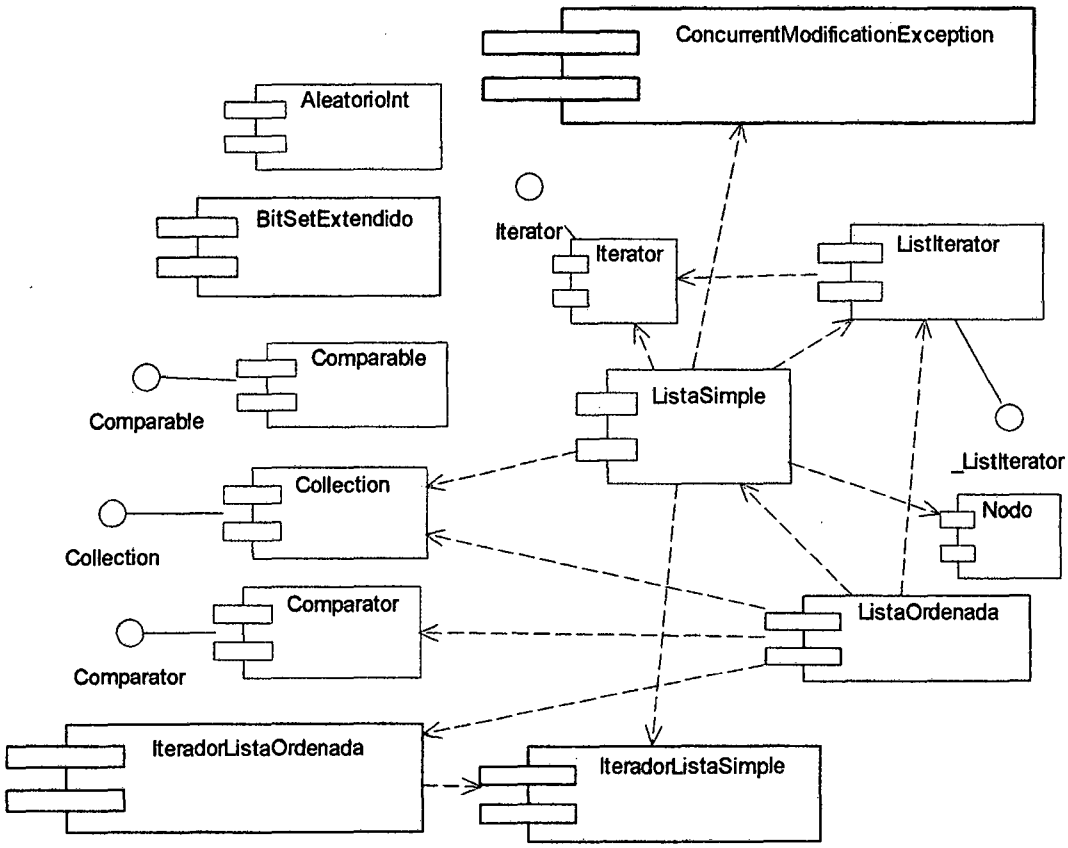
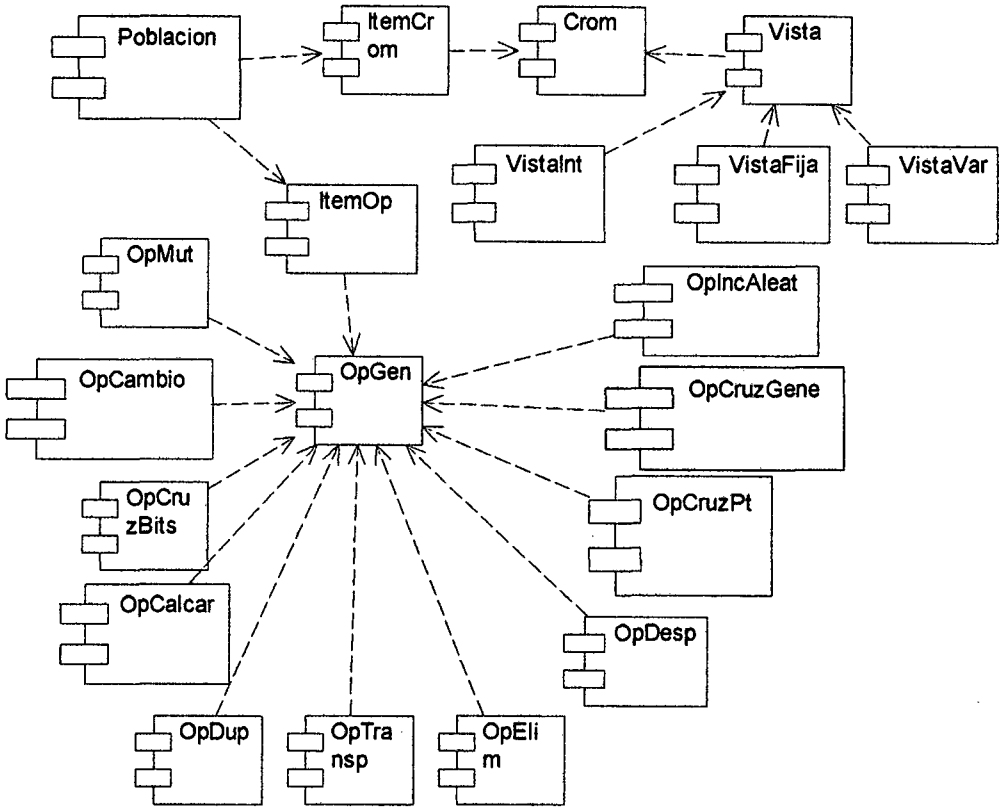


Diagrama de componentes del paquete ag:

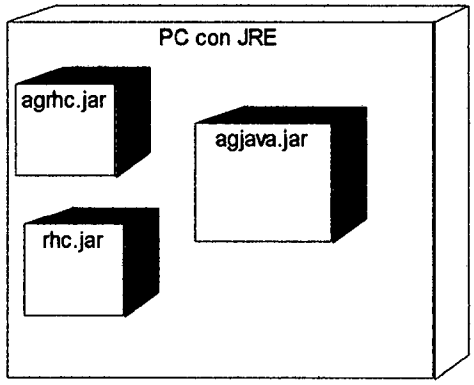


3.2.3. FASE DE CONSTRUCCIÓN

En la fase de construcción se desarrollan las siguientes actividades:

- Definir Modelos de Información para Interfaces
- Especificar pre y pos condiciones
- Especificar Restricciones de Componentes e Interfaces

El siguiente diagrama de despliegue nos muestra el entorno sencillo requerido para la aplicación:



Para la especificación y documentación del componente agjava se utilizó la herramienta javadoc, así logramos especificar para el componente agjava. El documento navegable contiene información de los paquetes del componente y para cada clase muestra un resumen con los campos, constructores y métodos. Además si uno quiere se puede obtener información detallada de campos, constructores y métodos. También se muestra información acerca de la relación de la clase con otras clases (Ver Figura III-8). El detalle de la documentación navegable para cada una de las clases se adjunta en el CD que acompaña la tesis junto con el componente agjava.jar, además del código completo.

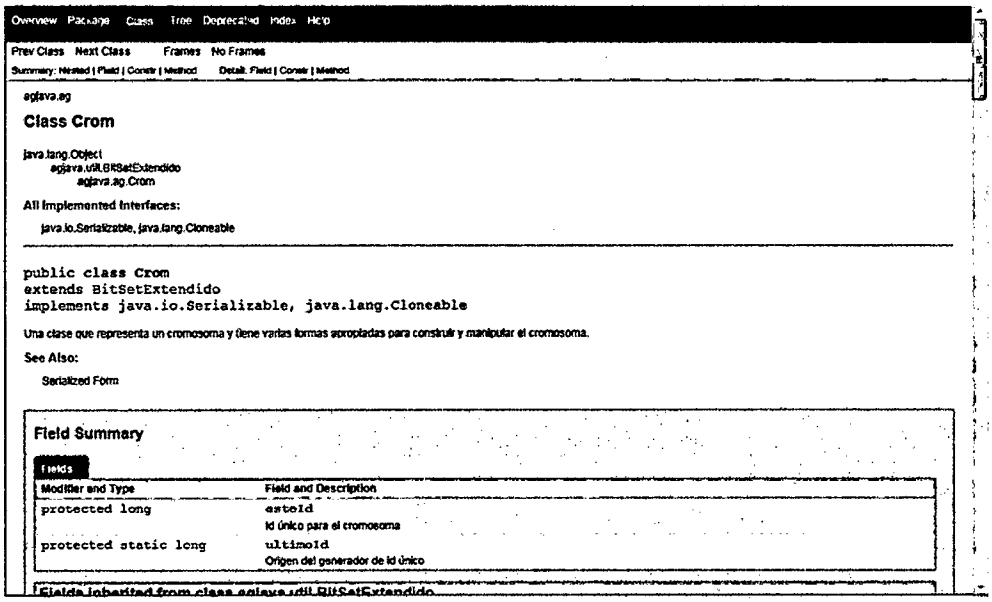


Figura III-8. Especificación de Interfaces del Componente agjava

Esta fase está fuertemente ligada a la implementación, los detalles de implementación se exponen ampliamente en la sección que sigue.

3.3. IMPLEMENTACIÓN DE LA FUNCIONALIDAD DEL COMPONENTE AGJAVA

3.3.1. PAQUETE AGJAVA.UTIL

El código correspondiente para la implementación de cada una de estas clases utilitarias se encuentra en el Anexo A.

Clase AleatorioInt

Para generar números aleatorios enteros se ha creado la clase AleatorioInt que es usado por algunas clases que implementan operadores genéticos y también al crear una nueva población por la clase Población. Por ejemplo, para generar números aleatorios enteros entre 0 y 500 podemos hacer:

```
AleatorioInt num = new AleatorioInt();
for(int i=1; i<=5; i++)
System.out.println(num.proximo(500));

Resultado:
417
313
406
462
209
Press any key to continue...
```

Clase BitSetExtendido

Una de las clases más importantes es la clase BitSetExtendido que permitirá crear un cromosoma, siendo esto esencial en lo que se refiere a los algoritmos genéticos, con esta clase se pretende manejar los números en su representación de bits de acuerdo a la representación de los números en Java, veamos a continuación como se puede usar esta clase.

Los bits con esta clase se almacenan en un array de bits.

```
BitSetExtendido cB01 = new BitSetExtendido("01000000000000000000000000000000");
BitSetExtendido cB02 = new BitSetExtendido("00000000000000000000000000000000");
BitSetExtendido cB03 = new BitSetExtendido("10000000000000000000000000000000");
BitSetExtendido cB04 = new BitSetExtendido("01111111100000000000000000000000");
BitSetExtendido cB05 = new BitSetExtendido("11111111100000000000000000000000");
```

```
BitSetExtendido cB06 = new BitSetExtendido("01111111100000100000000000000000");
BitSetExtendido cB07 = new BitSetExtendido("11111111100000100000000000000000");
BitSetExtendido cB08 = new BitSetExtendido("01000000110100000000000000000000");
BitSetExtendido cB09 = new BitSetExtendido("11000000110100000000000000000000");
BitSetExtendido cB10 = new BitSetExtendido("00000000100000000000000000000000");
BitSetExtendido cB11 = new BitSetExtendido("00000000100000000000000000000000");
BitSetExtendido cB12 = new BitSetExtendido("00000000000000000000000000000001");

System.out.println( "valorFloat("+ cB01+")="+cB01.valorFloat());
System.out.println( "valorFloat("+ cB02+")="+cB02.valorFloat());
System.out.println( "valorFloat("+ cB03+")="+cB03.valorFloat());
System.out.println( "valorFloat("+ cB04+")="+cB04.valorFloat());
System.out.println( "valorFloat("+ cB05+")="+cB05.valorFloat());
System.out.println( "valorFloat("+ cB06+")="+cB06.valorFloat());
System.out.println( "valorFloat("+ cB07+")="+cB07.valorFloat());
System.out.println( "valorFloat("+ cB08+")="+cB08.valorFloat());
System.out.println( "valorFloat("+ cB09+")="+cB09.valorFloat());
System.out.println( "valorFloat("+ cB10+")="+cB10.valorFloat());
System.out.println( "valorFloat("+ cB11+")="+cB11.valorFloat());
System.out.println( "valorFloat("+ cB12+")="+cB12.valorFloat());
```

Resultado:

```
valorFloat(01000000000000000000000000000000)=2.0
valorFloat(00000000000000000000000000000000)=0.0
valorFloat(10000000000000000000000000000000)=-0.0
valorFloat(01111111100000000000000000000000)=Infinity
valorFloat(11111111100000000000000000000000)=-Infinity
valorFloat(01111111100000100000000000000000)=NaN
valorFloat(11111111100000100000000000000000)=NaN
valorFloat(01000000110100000000000000000000)=6.5
valorFloat(11000000110100000000000000000000)=-6.5
valorFloat(00000001000000000000000000000000)=1.17549435E-38
valorFloat(00000000100000000000000000000000)=5.877472E-39
valorFloat(00000000000000000000000000000001)=1.4E-45
```

Otro ejemplo de la creación de un conjunto de bits de 66 caracteres:

```
BitSetExtendido conjBits = new
BitSetExtendido("01000000000000000000000000000000000000000000000000000000000001111
");
System.out.println( conjBits);
```

Resultado:

```
01000000000000000000000000000000000000000000000000000000000001111
Press any key to continue...
```

En el caso de ingresar caracteres diferentes a '0' se reconocen como '1':

```
BitSetExtendido conjBits = new BitSetExtendido("suerte");
System.out.println( conjBits);
```

Resultado:

```
111111
```

Se puede crear un conjunto de bits a partir de un array de tipo byte:

```
byte a[]= {0,1,1,1,1,1,1,1};
BitSetExtendido conjBits = new BitSetExtendido(64,a);
System.out.println( conjBits);
```

Resultado:

```
0000000100000001000000010000000100000001000000010000000100000000
```

Se puede crear un conjunto de bits a partir de un array de tipo long:

```
long b[]= {127,1};
BitSetExtendido conjBits = new BitSetExtendido(128,b);
System.out.println( conjBits);
```

Resultado:

```
10000000000000000000000000000000000000000000000000000000000000010000000000
00000000000000000000000000000000000000000001111111
Press any key to continue...
```

Se puede crear otro conjunto de bits a partir de un array de tipo long truncando los bits que sean necesarios:

```
long b[]= {127,1};
BitSetExtendido conjBits = new BitSetExtendido(20,b);
System.out.println( conjBits);
```

Resultado:

```
00000000000001111111
Press any key to continue...
```

Para crear un conjunto de bits con elementos igual a cero:

```
BitSetExtendido conjBits = new BitSetExtendido(65);
System.out.println( conjBits);
```

Resultado:

```
00000000000000000000000000000000000000000000000000000000000000000
Press any key to continue...
```

Para crear un conjunto de bits como una copia de otro conjunto de bits:

```
BitSetExtendido conjBits = new BitSetExtendido(60,5);
System.out.println( conjBits);

BitSetExtendido conjBits1 = new BitSetExtendido(conjBits);
System.out.println( conjBits1);
```

Resultado:

```
10110000001101011001101111110010110100110101110010110101100 0
10110000001101011001101111110010110100110101110010110101100 0
Press any key to continue...
```

Se puede copiar el contenido de un conjunto de bits a otro.

```
BitSetExtendido conjBits = new BitSetExtendido(60,5);
System.out.println( conjBits);

BitSetExtendido conjBits1 = new BitSetExtendido();
conjBits1.copiar(conjBits);
System.out.println( conjBits1);
```

Resultado:

```
10110000001101011001101111110010110100110101110010110101100 0
10110000001101011001101111110010110100110101110010110101100 0
Press any key to continue...
```

El método clone() crea y devuelve una copia del conjunto de bits:

```
BitSetExtendido conjBits = new BitSetExtendido(59,0);
System.out.println( conjBits);

BitSetExtendido conjBits1 = (BitSetExtendido) conjBits.clone();
System.out.println( conjBits1);
```

Resultado:

```
01111110101001101110000100010010101101111001111000111011010
01111110101001101110000100010010101101111001111000111011010
Press any key to continue...
```

Con el método equals() comprobamos si dos objetos son los mismos:

```
BitSetExtendido conjBits1 = new BitSetExtendido("0010");

BitSetExtendido conjBits2;

conjBits2 = conjBits1;

if(conjBits2.equals(conjBits1))
      System.out.println("Es el mismo objeto");
else
      System.out.println("No es el mismo objeto");
```

Resultado:

```
Es el mismo objeto
Press any key to continue...
```

Con el método hash() se genera el código hash para el conjunto de bits:

```
BitSetExtendido conjBits = new BitSetExtendido("0001");
System.out.println(conjBits.hashCode());

Resultado:

5
Press any key to continue...
```

El método tamaño() determina el tamaño del conjunto de bits:

```
BitSetExtendido conjBits = new BitSetExtendido("1010");
System.out.println(conjBits.tamaño());
```

Resultado:

```
4
Press any key to continue...
```

Con el método tamaño() se puede variar el tamaño del conjunto de bits:

```
BitSetExtendido conjBits = new BitSetExtendido("1010");
conjBits.tamaño(8);
System.out.println(conjBits);
```

Resultado:

```
00001010
Press any key to continue...
```

Para obtener el valor de un bit en particular se usa el método obtener(), obtener() devuelve true si el valor del bit es 1:

```
BitSetExtendido conjBits = new BitSetExtendido("11000");
      if(conjBits.obtener(4)==true)
            System.out.println("1");
      else
            System.out.println("0");
```

Resultado:

```
1
Press any key to continue...
```

Para establecer el valor a "1" de un bit en particular se usa el método establecer():

```
BitSetExtendido conjBits = new BitSetExtendido("11000");
conjBits.establecer(0);
System.out.println(conjBits);
```

Resultado:

```
11001
Press any key to continue...
```

Para establecer el valor a "1" de un rango de bits en particular:

```
BitSetExtendido conjBits = new BitSetExtendido("11000001");
conjBits.establecer(3,2);
System.out.println( conjBits);
```

Resultado:

```
11011001
Press any key to continue...
```

Para establecer el valor a "0" de un bit en particular se usa el método limpiar():

```
BitSetExtendido conjBits = new BitSetExtendido("111111111");
conjBits.limpiar(3);
System.out.println( conjBits);
```

Resultado:

```
111110111
Press any key to continue...
```

Para establecer el valor a "0" de un rango de bits en particular:

```
BitSetExtendido conjBits = new BitSetExtendido("111111111");
```

```
conjBits.limpiar(3,4);
System.out.println( conjBits);
```

Resultado:

```
110000111
Press any key to continue...
```

El método invertir() invierte el valor de un bit en particular o de un conjunto de bits en particular:

```
BitSetExtendido conjBits = new BitSetExtendido("101010101010");
conjBits.invertir(4);
System.out.println( conjBits);
```

Resultado:

```
101010111010
Press any key to continue...
```

Otra forma de utilizar el método invertir

```
BitSetExtendido conjBits = new BitSetExtendido("101010101010");
conjBits.invertir(4,5);
System.out.println( conjBits);
```

Resultado:

```
101101011010
Press any key to continue...
```

El método and() Realiza la operación and con otro conjunto de bits:

```
BitSetExtendido conjBits1 = new BitSetExtendido(     "1001100");
BitSetExtendido conjBits2 = new BitSetExtendido("101000101010");
conjBits1.and(conjBits2);
System.out.println( conjBits1);
```

Resultado:

```
000000001000

Press any key to continue...
```

Otra forma de utilizar el método and:

```
BitSetExtendido conjBits1 = new BitSetExtendido("101001001100");
BitSetExtendido conjBits2 = new BitSetExtendido(     "1011010");
conjBits1.and(conjBits2);
System.out.println( conjBits1);
```

Resultado:

```
000001001000
Press any key to continue...
```

Otra prueba con el método and:

```
BitSetExtendido conjBits1 = new BitSetExtendido("   1001100");
BitSetExtendido conjBits2 = new BitSetExtendido("101000101010");
conjBits1.and(3,conjBits2);
System.out.println( conjBits1);
```

Resultado:

```
000000001000100
Press any key to continue...
```

De forma similar al método and() se usa el método or() para realizar la operación or inclusiva:

```
BitSetExtendido conjBits1 = new BitSetExtendido(     "1001100");
BitSetExtendido conjBits2 = new BitSetExtendido("101000101010");
conjBits1.or(conjBits2);
System.out.println( conjBits1);
```

Resultado:

```
101001101110
Press any key to continue...
```

Otra prueba más del método or:

```
BitSetExtendido conjBits1 = new BitSetExtendido("1001100");
BitSetExtendido conjBits2 = new BitSetExtendido("101000101010");
conjBits2.or(conjBits1);
System.out.println( conjBits2);
```

Resultado:

```
101001101110
Press any key to continue...
```

Otra forma más de usar el método or:

```
BitSetExtendido conjBits1 = new BitSetExtendido("1001100");
BitSetExtendido conjBits2 = new BitSetExtendido("101000101010");
conjBits1.or(4,conjBits2);
System.out.println(conjBits1);
```

Resultado:

```
1010001011101100
Press any key to continue...
```

El método xor() se emplea para la operación or exclusiva:

```
BitSetExtendido conjBits1 = new BitSetExtendido("1001100");
BitSetExtendido conjBits2 = new BitSetExtendido("101000101010");
conjBits1.xor(conjBits2);
System.out.println( conjBits1);
```

Resultado:

```
101001100110
Press any key to continue...

BitSetExtendido conjBits1 = new BitSetExtendido("1001100");
BitSetExtendido conjBits2 = new BitSetExtendido("101000101010");
conjBits2.xor(5,conjBits1);
System.out.println(conjBits2);
```

Resultado:

```
001110101010
Press any key to continue...
```

El método inicializarTodo() permite inicializar un conjunto de bits basado en un array de tipo long:

```
BitSetExtendido conjBits1 = new BitSetExtendido("1001100");
long a[] = {4,5};
conjBits1.inicializarTodo(70,a);
System.out.println(conjBits1);
```

Resultado:

```
00010100000000000000000000000000000000000000000000000000000000100
Press any key to continue...
```

Para establecer el valor de un bit en particular utilizando un bolean se usa el método establecerBooleanEn():

```
BitSetExtendido conjBits = new BitSetExtendido("1001100");
conjBits.establecerBooleanEn(0,true);
conjBits.establecerBooleanEn(6,false);
System.out.println(conjBits);
```

Resultado:

```
0001101
Press any key to continue...
```

En el lenguaje de programación Java los enteros positivos tienen una representación binaria simple, mientras que los números enteros negativos tienen una representación usando la notación binaria con complemento a 2.

El método valorByte() devuelve el valor byte equivalente del conjunto de bits para representar valores enteros entre -128 y +127:

```
BitSetExtendido conjBits = new BitSetExtendido("1001100");
System.out.println(conjBits.valorByte());
System.out.println(conjBits);
```

Resultado:

```
76
01001100
Press any key to continue...
```

```
BitSetExtendido conjBits = new BitSetExtendido("1001100");
System.out.println(conjBits.obtenerByteEn(2));
```

Resultado:

```
19
Press any key to continue...
```

```
BitSetExtendido conjBits = new BitSetExtendido("1001100");
System.out.println(conjBits.obtenerByteEn(2,3));
```

Resultado:

```
3
Press any key to continue...
```

El método establecerByteEn() permite la representación de un byte (ocho bits) en la ubicación dada en el conjunto de bits:

```
BitSetExtendido conjBits = new BitSetExtendido("1001100");
conjBits.establecerByteEn(5,(byte)3);
System.out.println(conjBits);
```

Resultado:

```
0000001101100
Press any key to continue...
```

```
BitSetExtendido conjBits = new BitSetExtendido("100000001001100");
conjBits.establecerByteEn(5,6,(byte)(-8));
System.out.println(conjBits);
```

Resultado:

```
100011100001100
Press any key to continue...
```

El tipo char (16 bits) permite declarar valores enteros en el rango \u0000 a \uFFFF en Unicode (0 a 65535) para este tipo tenemos los métodos valorChar(), obtenerCharEn(), establecerCharEn():

```
BitSetExtendido conjBits = new BitSetExtendido("1100000001001100");
System.out.println((int)conjBits.valorChar());
System.out.println((int)conjBits.obtenerCharEn(14));
System.out.println((int)conjBits.obtenerCharEn(3,5));
System.out.println();
conjBits.establecerCharEn(3,(char)2);
System.out.println(conjBits);
BitSetExtendido conjBits2 = new BitSetExtendido("1100000001001100");
conjBits2.establecerCharEn(3,5,(char)2);
System.out.println(conjBits2);
```

```
Resultado:

49228
3
9

00000000000000000000000010100
1100000000010100
Press any key to continue...
```

El tipo short (16 bits) permite declarar valores enteros en el rango -32768 y +32767 para este tipo tenemos los métodos valorShort(), obtenerShortEn(), establecerShortEn():

```
BitSetExtendido conjBits = new BitSetExtendido("1100000001001100");
System.out.println(conjBits.valorShort());
System.out.println(conjBits.obtenerShortEn(14));
System.out.println(conjBits.obtenerShortEn(3,5));
System.out.println();
conjBits.establecerShortEn(3,(short)2);
System.out.println(conjBits);
BitSetExtendido conjBits2 = new BitSetExtendido("1100000001001100");
conjBits2.establecerShortEn(3,5,(short)2);
System.out.println(conjBits2);
```

Resultado:

```
-16308
3
9

00000000000000000000000010100
1100000000010100
Press any key to continue...
```

El tipo int (32 bits) permite declarar valores enteros en el rango -2147483648 y +2147483647 para este tipo tenemos los métodos valorInt(), obtenerIntEn(), establecerIntEn():

```
BitSetExtendido conjBits = new BitSetExtendido("1100000001001100");
System.out.println(conjBits.valorInt());
System.out.println(conjBits.obtenerIntEn(14));
System.out.println(conjBits.obtenerIntEn(3,5));
System.out.println();
conjBits.establecerIntEn(3,2);
System.out.println(conjBits);
BitSetExtendido conjBits2 = new BitSetExtendido("1100000001001100");
conjBits2.establecerIntEn(3,5,2);
System.out.println(conjBits2);
```

Resultado:

```
49228
3
9

00000000000000000000000000000000000010100
1100000000010100
Press any key to continue...
```

El tipo long (64 bits) permite declarar valores enteros en el rango -9223372036854775808 y +9223372036854775808 para este tipo tenemos los métodos valorLong(), obtenerLongEn(), establecerLongEn():

```
BitSetExtendido conjBits = new BitSetExtendido("1100000001001100");
System.out.println(conjBits.valorLong());
System.out.println(conjBits.obtenerLongEn(14));
System.out.println(conjBits.obtenerLongEn(3,5));
System.out.println();
conjBits.establecerLongEn(3,2);
System.out.println(conjBits);
BitSetExtendido conjBits2 = new BitSetExtendido("1100000001001100");
conjBits2.establecerLongEn(3,5,2);
```

```
System.out.println(conjBits2);

Resultado:

49228
3
9

00000000000000000000000000000000000000000000000000000000000000000000000000000101
00
1100000000010100
Press any key to continue...
```

En el caso de lo números en punto flotante, estos se representan usando el estándar IEEE 754.

Para el tipo float (32 bits) se tienen los métodos valorFloat(), obtener-FloatEn() y establecerFloatEn():

```
BitSetExtendido conjBits = new BitSetExtendido("1100000001001100");
System.out.println(conjBits.valorFloat());
System.out.println(conjBits.obtenerFloatEn(14));
conjBits.establecerFloatEn(3,2.0F);
System.out.println(conjBits);
System.out.println(conjBits.valorFloat());
```

Resultado:

```
6.8983E-41
4.2E-45
00000000000010000000000000000000000000000000100
5.6E-45
Press any key to continue...
```

Para el tipo double (64 bits) se tienen los métodos valorDouble(), obtener-DoubleEn() y establecerDoubleEn():

```
BitSetExtendido conjBits = new BitSetExtendido("1100000001001100");
System.out.println(conjBits.valorDouble());
System.out.println(conjBits.obtenerDoubleEn(14));
conjBits.establecerDoubleEn(3,2.0);
System.out.println(conjBits);
System.out.println(conjBits.valorDouble());

Resultado:

2.4322E-319
1.5E-323
00000000000010000000000000000000000000000000000000000000000000000000000000000001
00
2.0E-323
Press any key to continue...
```

El método encontrarSubConjunto() permite crear un subconjunto a partir de un conjunto de bits:

```
BitSetExtendido conjBits1 = new BitSetExtendido("00101110000");
System.out.println(conjBits1);
```

```
BitSetExtendido conjBits2;
conjBits2 = conjBits1.encontrarSubConjunto(3,5);
System.out.println(conjBits2);
```

Resultado:

```
00101110000
01110
Press any key to continue...
```

El método establecerSubConjunto() permite establecer un subconjunto en otro conjunto de bits:

```
BitSetExtendido conjBits1 = new BitSetExtendido("000000");
BitSetExtendido conjBits2 = new BitSetExtendido("111");
BitSetExtendido conjBits3;
conjBits3=conjBits1.establecerSubConjunto(2,conjBits2);
System.out.println(conjBits1);
System.out.println(conjBits2);
System.out.println(conjBits3);
```

Resultado:

```
011100
111
011100
Press any key to continue...
```

El método intercambiarSubConj() permite intercambiar bits entre conjuntos de bits:

```
BitSetExtendido conjBits1 = new BitSetExtendido("000000");
BitSetExtendido conjBits2 = new BitSetExtendido("111111");
BitSetExtendido conjBits3;
conjBits3=conjBits1.intercambiarSubConj(2,2,conjBits2);
System.out.println(conjBits1);
System.out.println(conjBits2);
System.out.println(conjBits3);
```

Resultado:

```
001100
110011
001100
Press any key to continue...
```

El método insertarSubConjunto() permite insertar un subconjunto en la posición dada del conjunto de bits:

```
BitSetExtendido conjBits1 = new BitSetExtendido("000000");
BitSetExtendido conjBits2 = new BitSetExtendido("111111");
BitSetExtendido conjBits3;
conjBits3=conjBits1.insertarSubConjunto(2,conjBits2);
System.out.println(conjBits1);
System.out.println(conjBits2);
System.out.println(conjBits3);
```

Resultado:

```
11111100
```

```
111111
11111100
Press any key to continue...
```

Si la posición es negativa se inserta al final del conjunto de bits:

```
BitSetExtendido conjBits1 = new BitSetExtendido("000000");
BitSetExtendido conjBits2 = new BitSetExtendido("111111");
BitSetExtendido conjBits3;
conjBits3=conjBits1.insertarSubConjunto(-2,conjBits2);
System.out.println(conjBits1);
System.out.println(conjBits2);
System.out.println(conjBits3);
```

Resultado:

```
111111000000
111111
111111000000
Press any key to continue...
```

Para borrar un subconjunto de bits se tiene el método borrarSubConjunto():

```
BitSetExtendido conjBits1 = new BitSetExtendido("0000001111111");
conjBits1.borrarSubConjunto(3,6);
System.out.println(conjBits1);
```

Resultado:

```
0000111
Press any key to continue...
```

Para arrastrar a la izquierda un subconjunto de bits se tiene el método arrastrarIzquierda():

```
BitSetExtendido conjBits1 = new BitSetExtendido("01100101");
conjBits1.arrastrarIzquierda(2,4,2);
System.out.println(conjBits1);
```

Resultado:

```
01010101
Press any key to continue...
```

Para desplazar a la izquierda un subconjunto de bits se tiene el método desplazarIzquierda():

```
BitSetExtendido conjBits1 = new BitSetExtendido("01100101");
conjBits1.desplazarIzquierda(2,4,2);
System.out.println(conjBits1);
```

Resultado:

```
01010001
Press any key to continue...
```

Para arrastrar y desplazar a la derecha un subconjunto de bits se tienen los métodos arrastrarDerecha (), desplazarDerechaSinSigno(), desplazarDerechaConSigno():

```
BitSetExtendido conjBits1 = new BitSetExtendido("01100101");
conjBits1.arrastrarDerecha(2,4,2);
System.out.println(conjBits1);
BitSetExtendido conjBits2 = new BitSetExtendido("01100101");
conjBits2.desplazarDerechaSinSigno(2,4,2);
System.out.println(conjBits2);
BitSetExtendido conjBits3 = new BitSetExtendido("01100101");
conjBits3.desplazarDerechaConSigno(2,4,2);
System.out.println(conjBits3);
```

Resultado

```
01101001
01001001
01111001
Press any key to continue...
```

Para rotar a la derecha y a la izquierda un subconjunto de bits tenemos el método rotarDerecha() y rotarIzquierda() respectivamente:

```
BitSetExtendido conjBits1 = new BitSetExtendido("01100101");
conjBits1.rotarIzquierda(2,4,1);
System.out.println(conjBits1);
BitSetExtendido conjBits2 = new BitSetExtendido("01100101");
conjBits2.rotarDerecha(2,4,1);
System.out.println(conjBits2);
```

Resultado:

```
01001101
01110001
Press any key to continue...
```

Clase ListaSimple

Permite crear la estructura de datos denominada Lista Simple. Esta clase sirve de base para crear la clase Poblacion y la clase ListaOrdenada que contendrá la población del algoritmo genético.

Se crean dos clases internas en la clase ListaSimple, una de ellas es

```
protected static class Nodo
{
    public Object item;
    public Nodo prox;
    public Nodo prev;
}
```

Un miembro estático es un miembro de la clase, una clase anidada estática no puede referirse directamente a un miembro del objeto, sólo puede hacerlo a través de un objeto de su clase.

La otra clase anidada es IteradorListaSimple para crear un iterador sobre esta lista. Esta añade toda la funcionalidad para poder trabajar con un objeto de la clase ListIterator.

Para crear una lista simple sin ningún elemento y muestra el tamaño de la lista:

```
ListaSimple lista1 = new ListaSimple();
System.out.println(lista1.size());
```

Resultado:

```
0
Press any key to continue...
```

Se puede crear una lista a partir de un objeto Collection

```
String cadena = new String("Hola");
Boolean booleano = new Boolean(false);
Integer entero = new Integer(10);

ListaSimple lista1 = new ListaSimple();
lista1.add(cadena);
lista1.add(booleano);
lista1.add(entero);

ListaSimple lista2 = new ListaSimple(lista1);
System.out.println(lista2.toString());
```

Resultado:

```
(Hola,false,10)
Press any key to continue...
```

Algunos métodos que sirven para añadir elementos a la lista (add), imprimir los elementos de la lista (toString), devolver el tamaño de la lista (size), saber si la lista esta vacía (isEmpty), saber si la lista contiene el objeto que se indica (contains).

```
String cadena = new String("Hola");
Boolean booleano = new Boolean(false);
Integer entero = new Integer(10);

ListaSimple lista1 = new ListaSimple();
lista1.add(cadena);
lista1.add(booleano);
lista1.add(entero);

ListaSimple lista2 = new ListaSimple(lista1);
System.out.println("Elementos de lista: " + lista2.toString());
System.out.println("Tamaño de lista: "+lista2.size());
System.out.println("Lista vacía: "+lista2.isEmpty());
System.out.println("Objeto booleano en lista: "
      +lista2.contains(booleano));
```

Resultado:

```
Elementos de lista: (Hola,false,10)
Tamaño de lista: 3
Lista vacía: false
Objeto booleano en lista: true
Press any key to continue...
```

Para imprimir los elementos de una lista se puede usar un Iterador, la función iteradorLista() devuelve un iterador sobre esta lista:

```
ListIterator miIterador = lista1.iteradorLista();
while (miIterador.hasNext())
      System.out.print(miIterador.next()+" ");
System.out.println();
```

```
Resultado:

Hola false 10
Press any key to continue...
```

Para establecer un iterador a partir del segundo elemento de la lista:

```
ListIterator miIterador = lista1.iteradorLista(1);
while (miIterador.hasNext())
      System.out.print(miIterador.next()+" ");
System.out.println();
```

Resultado:

```
false 10
Press any key to continue...
```

Para saber si lista1 contiene todos los elementos de la Colección lista2:

```
String cadena = new String("Hola");
Boolean booleano = new Boolean(false);
Integer entero = new Integer(10);

ListaSimple lista1 = new ListaSimple();
lista1.add(cadena);
lista1.add(booleano);
lista1.add(entero);

ListaSimple lista2 = new ListaSimple(lista1);

System.out.println(lista1.containsAll(lista2));
```

Resultado:

```
true
Press any key to continue...
```

Para convertir la lista en un Array de tipo Object:

```
Object miArray[] = lista1.toArray();
for(int i=0; i<miArray.length;i++)
      System.out.print(miArray[i] + " ");
System.out.println();
```

Resultado:

```
Hola false 10
Press any key to continue...
```

Para almacenar los elementos en un array específico:

```
Object[] miArray = new Object[3];
lista1.toArray(miArray);
      for(int i=0; i<miArray.length;i++)
            System.out.print(miArray[i] + " ");
      System.out.println();
```

Resultado:

```
Hola false 10
Press any key to continue...
```

Para insertar un elemento en un índice dado:

```
Double doble = new Double(45.89);
lista1.add(1,doble);

System.out.println(lista1.toString());
```

Resultado:

```
(Hola,45.89,false,10)
Press any key to continue...
```

Para remover un objeto de la lista:

```
Double doble = new Double(45.89);
lista1.add(1,doble);
System.out.println(lista1.toString());

lista1.remove(booleano);
System.out.println(lista1.toString());
```

```
Resultado:
(Hola,45.89,false,10)
(Hola,45.89,10)
Press any key to continue...
```

Para añadir los elementos de una colección a la lista:

```
String cadena = new String("Hola");
Boolean booleano = new Boolean(false);
Integer entero = new Integer(10);

ListaSimple lista1 = new ListaSimple();
lista1.add(cadena);
lista1.add(booleano);
lista1.add(entero);
System.out.println("Lista 1: "+ lista1.toString());

String cadena2 = new String("Adios");
Boolean booleano2 = new Boolean(true);
Integer entero2 = new Integer(20);

ListaSimple lista2 = new ListaSimple();
lista2.add(cadena2);
lista2.add(booleano2);
lista2.add(entero2);
System.out.println("Lista 2: "+ lista2.toString());

lista1.addAll(lista2);
```

```
System.out.println("Nueva Lista 1: "+ lista1.toString());
```

Resultado:

```
Lista 1: (Hola,false,10)
Lista 2: (Adios,true,20)
Nueva Lista 1: (Hola,false,10,Adios,true,20)
Press any key to continue...
```

Para remover de una lista los elementos que coinciden con la selección:

```
String cadena = new String("Hola");
Boolean booleano = new Boolean(false);
Integer entero = new Integer(10);

ListaSimple lista1 = new ListaSimple();
lista1.add(cadena);
lista1.add(booleano);
lista1.add(entero);
System.out.println("Lista 1: "+ lista1.toString());

String cadena2 = new String("Adios");
Boolean booleano2 = new Boolean(true);
Integer entero2 = new Integer(20);

ListaSimple lista2 = new ListaSimple();
lista2.add(cadena2);
lista2.add(booleano2);
lista2.add(entero2);
System.out.println("Lista 2: "+ lista2.toString());

lista1.addAll(lista2);
System.out.println("Nueva Lista 1: "+ lista1.toString());


System.out.println("Se remueven los elementos que coinciden " +
        "con la Lista 2");
lista1.removeAll(lista2);
System.out.println("Nueva Lista 1: "+ lista1.toString());
```

```
Resultado:

Lista 1: (Hola,false,10)
Lista 2: (Adios,true,20)
Nueva Lista 1: (Hola,false,10,Adios,true,20)
Se remueven los elementos que coinciden con la Lista 2
Nueva Lista 1: (Hola,false,10)
Press any key to continue...
```

Para retener sólo los elementos en esta lista que están contenidos en la colección especificada:

```
String cadena = new String("Hola");
Boolean booleano = new Boolean(false);
Integer entero = new Integer(10);

ListaSimple lista1 = new ListaSimple();
lista1.add(cadena);
lista1.add(booleano);
lista1.add(entero);
System.out.println("Lista 1: "+ lista1.toString());
```

```
        String cadena2 = new String("Adios");
        Boolean booleano2 = new Boolean(true);
        Integer entero2 = new Integer(20);

        ListaSimple lista2 = new ListaSimple();
        lista2.add(cadena2);
        lista2.add(booleano2);
        lista2.add(entero2);
        System.out.println("Lista 2: "+ lista2.toString());

        lista1.addAll(lista2);
        System.out.println("Nueva Lista 1: "+ lista1.toString());


        System.out.println("Se retienen los elementos que coinciden " +
                "con la Lista 2");
        lista1.retainAll(lista2);
        System.out.println("Nueva Lista 1: "+ lista1.toString());
```

```
Resultado:

Lista 1: (Hola,false,10)
Lista 2: (Adios,true,20)
Nueva Lista 1: (Hola,false,10,Adios,true,20)
Se retienen los elementos que coinciden con la Lista 2
Nueva Lista 1: (Adios,true,20)
Press any key to continue...
```

Para comparar una lista con una colección para saber si son iguales:

```
        String cadena = new String("Hola");
        Boolean booleano = new Boolean(false);
        Integer entero = new Integer(10);

        ListaSimple lista1 = new ListaSimple();
        lista1.add(cadena);
        lista1.add(booleano);
        lista1.add(entero);
        System.out.println("Lista 1: "+ lista1.toString());

        ListaSimple lista2 = new ListaSimple(lista1);
        System.out.println("Lista 2: "+ lista2.toString());

        System.out.println("lista1.equals(lista2):"+
      lista1.equals(lista2));
```

```
Resultado:

Lista 1: (Hola,false,10)
Lista 2: (Hola,false,10)
lista1.equals(lista2): true
Press any key to continue...
```

Para generar el código hash de una lista:

```
        String cadena = new String("Hola");
        Boolean booleano = new Boolean(false);
        Integer entero = new Integer(10);

        ListaSimple lista1 = new ListaSimple();
        lista1.add(cadena);
        lista1.add(booleano);
        lista1.add(entero);
```

```
String cadena2 = new String("Adios");
Boolean booleano2 = new Boolean(true);
Integer entero2 = new Integer(20);

ListaSimple lista2 = new ListaSimple();
lista2.add(cadena2);
lista2.add(booleano2);
lista2.add(entero2);


System.out.println("Codigo hash Lista 1: "+ lista1.hashCode());
System.out.println("Codigo hash Lista 2: "+ lista2.hashCode());
```

Resultado:

```
Codigo hash Lista 1: 2255872
Codigo hash Lista 2: 63111218
Press any key to continue...
```

Para remover cada uno de los elementos de la lista:

```
String cadena = new String("Hola");
Boolean booleano = new Boolean(false);
Integer entero = new Integer(10);

ListaSimple lista1 = new ListaSimple();
lista1.add(cadena);
lista1.add(booleano);
lista1.add(entero);
System.out.println("Lista 1: "+ lista1.toString());
lista1.clear();
System.out.println("Nueva Lista 1: "+ lista1.toString())
```

Resultado:

```
Lista 1: (Hola,false,10)
Nueva Lista 1: ()
Press any key to continue...
```

Clase ListaOrdenada

Extiende la clase ListaSimple, se usa en la clase Poblacion y para ordenar los resultados en el conjunto de soluciones final. ListaSimple sirve para manejar los elementos en orden ascendente (parcial) basado en el resultado de llamar a compareTo en el objeto o al usar un Comparator provisto en la construcción.

```
Poblacion diseños = new Poblacion( 3, 2, 4, 4);
ListaOrdenada resultados = new ListaOrdenada( diseños );
System.out.println(resultados.toString());
```

```
Resultado:
([0.0,[1:10010111]],[0.0,[2:00011010]],[0.0,[3:00111010]])
Press any key to continue...
```

Para crear un iterador de lista:

```
Poblacion diseños = new Poblacion( 3, 2, 4, 4);

ListaOrdenada resultados = new ListaOrdenada( diseños );
System.out.println(resultados.toString());

for (ListIterator iter =resultados.iteradorLista();
            iter.hasNext(); )

    System.out.println( iter.next() );
```

Resultado:

```
([0.0,[1:10001000]],[0.0,[2:11011000]],[0.0,[3:01010011]])
[0.0,[1:10001000]]
[0.0,[2:11011000]]
[0.0,[3:01010011]]
Press any key to continue...
```

3.3.2. CLASES DEL PAQUETE AGJAVA.AG

El código correspondiente para la implementación de cada una de estas clases del algoritmo genético se encuentra en el Anexo B.

Clase Crom

La clase Crom es una extensión de la clase BitSetExtendido por lo tanto también tiene los métodos de BitSetExtendido. Representa un cromosoma en un algoritmo genético y que se utilizará directamente en la aplicación que se quiera desarrollar. Lo usa la clase Poblacion, ItemCrom, las clases que determinan los operadores genéticos y las clases que lo interpretan a través de vistas.

El constructor por defecto Crom():

```
Crom cromosoma1 = new Crom();
System.out.println(cromosoma1);
System.out.println(cromosoma1.esteId);
System.out.println(cromosoma1.ultimoId);

Crom cromosoma2 = new Crom();
System.out.println(cromosoma2);
System.out.println(cromosoma2.esteId);
System.out.println(cromosoma2.ultimoId);
```

Resultado:

```
[0:]
0
0
[0:]
0
0
Press any key to continue...
```

Para crear cromosomas aleatoriamente con una longitud dada:

```
Crom cromosoma1 = new Crom(5);
System.out.println(cromosoma1);
System.out.println(cromosoma1.esteId);
System.out.println(cromosoma1.ultimoId);

Crom cromosoma2 = new Crom(5);
System.out.println(cromosoma2);
System.out.println(cromosoma1.esteId);
System.out.println(cromosoma1.ultimoId);
```

Resultado:

```
[1:00010]
1
1
[2:01001]
1
2
Press any key to continue...
```

Se pueden crear cromosomas a partir de una instancia de la clase BitSetExtendido:

```
BitSetExtendido conjBits1 = new BitSetExtendido("1110001");
Crom cromosoma1 = new Crom(conjBits1);
System.out.println(cromosoma1);

BitSetExtendido conjBits2 = new BitSetExtendido(
                                                "0111100000001");
Crom cromosoma2 = new Crom(conjBits2);
System.out.println(cromosoma2);
```

Resultado:

```
[1:1110001]
[2:0111100000001]
Press any key to continue...
```

Para crear un cromosoma exactamente igual al padre:

```
BitSetExtendido conjBits1 = new BitSetExtendido("1110001");
Crom cromosoma1 = new Crom(conjBits1);
System.out.println(cromosoma1);

Crom cromosoma2 = new Crom(cromosoma1);
System.out.println(cromosoma2);
```

Resultado:

```
[1:1110001]
[2:1110001]
Press any key to continue...
```

Se puede crear un cromosoma a partir de otros dos cromosomas con una tasa de cruzamiento por defecto de 0.5 (la probabilidad de escoger un bit de cada cromosoma es de 0.50):

```
BitSetExtendido conjBits1 = new BitSetExtendido("00000000");
Crom cromosoma1 = new Crom(conjBits1);
System.out.println(cromosoma1);

BitSetExtendido conjBits2 = new BitSetExtendido("11111");
Crom cromosoma2 = new Crom(conjBits2);
System.out.println(cromosoma2);

Crom cromosoma3 = new Crom(cromosoma1,cromosoma2);
System.out.println(cromosoma3);
```

Resultado:

```
[1:00000000]
[2:11111]
[3:00001101]
Press any key to continue...
```

Se puede crear un cromosoma a partir de otros dos cromosomas con una tasa de cruzamiento entre 0 y 1:

```
BitSetExtendido conjBits1 = new BitSetExtendido("0000000000");
Crom cromosoma1 = new Crom(conjBits1);
System.out.println(cromosoma1);

BitSetExtendido conjBits2 = new BitSetExtendido("11111");
Crom cromosoma2 = new Crom(conjBits2);
System.out.println(cromosoma2);

Crom cromosoma3 = new Crom(cromosoma1,cromosoma2,0.7);
System.out.println(cromosoma3);
```

Resultado:

```
[1:0000000000]
[2:11111]
[3:0000011011]
Press any key to continue...
```

Un cromosoma puede mutar para dar origen a otro cromosoma, se puede especificar una tasa de mutación:

```
BitSetExtendido conjBits1 = new BitSetExtendido("0000000000");
Crom cromosoma1 = new Crom(conjBits1);
Crom cromosoma2 = new Crom(cromosoma1,0.8);
System.out.println(cromosoma2);
```

Resultado:

```
[2:1110101101]
Press any key to continue...
```

Un cromosoma puede copiar el contenido de otro cromosoma y hacerlo suyo:

```
BitSetExtendido conjBits1 = new BitSetExtendido("0000000000");
Crom cromosoma1 = new Crom(conjBits1);
System.out.println(cromosoma1);

BitSetExtendido conjBits2 = new BitSetExtendido("11111");
```

```
Crom cromosoma2 = new Crom(conjBits2);
System.out.println(cromosoma2);

cromosoma1.copiar(cromosoma2);
System.out.println(cromosoma1);
```

Resultado:

```
[1:0000000000]
[2:11111]
[1:11111]
Press any key to continue...
```

Se puede clonar un cromosoma y crear un nuevo cromosoma:

```
Crom cromosoma1 = new Crom(8);
System.out.println(cromosoma1);

Crom cromosoma2 = (Crom)cromosoma1.clone();
System.out.println(cromosoma2);
```

Resultado:

```
[1:01011100]
[2:01011100]
Press any key to continue...
```

Un cromosoma puede mutar con una tasa de mutación dada:

```
Crom cromosoma1 = new Crom(8);
System.out.println(cromosoma1);

cromosoma1.mutar(0.9);
System.out.println(cromosoma1);
```

Resultado:

```
[1:10011111]
[1:11101000]
Press any key to continue...
```

Para trasponer conjuntos de bits de un cromosoma:

```
Crom cromosoma1 = new Crom(8);
System.out.println(cromosoma1);

cromosoma1.transponerSegmento(2,5,2);
System.out.println(cromosoma1);
```

Resultado:

```
[1:01000101]
[1:00101001]
Press any key to continue...
```

Se puede añadir un segmento aleatorio de bits al final del cromosoma:

```
Crom cromosoma1 = new Crom(8);
System.out.println(cromosoma1);

cromosoma1.añadirSegmentoAlFinal(5);
System.out.println(cromosoma1);
```

Resultado:

```
[1:00100010]
[1:1001100100010]
Press any key to continue...
```

Para eliminar un conjunto de bits de un cromosoma:

```
Crom cromosoma1 = new Crom(8);
System.out.println(cromosoma1);

cromosoma1.eliminarSegmento(3,2);
System.out.println(cromosoma1);
```

Resultado:

```
[1:11111000]
[1:111000]
Press any key to continue...
```

En el caso de duplicar un segmento de bits después del original del cromosoma:

```
Crom cromosoma1 = new Crom(8);
System.out.println(cromosoma1);

cromosoma1.duplicarSegmento(2,4);
System.out.println(cromosoma1);
```

Resultado:

```
[1:11000110]
[1:110000110]
Press any key to continue...
```

Clase ItemCrom

La clase ItemCrom sirve para manejar un cromosoma, es decir un objeto de la clase Crom y asignarle una capacidad que se usará en la población y en la aplicación que se basará en algoritmos genéticos.

El constructor ItemCrom() permite crear un objeto vacío:

```
ItemCrom item1 = new ItemCrom();
```

Se puede crear un objeto de la clase ItemCrom a partir de un cromosoma y se le puede asignar una capacidad y se puede hacer una copia profunda a partir de otro objeto de la clase ItemCrom:

```
Crom crom1 = new Crom(64);
System.out.println(crom1);

double capac1 = crom1.obtenerDoubleEn(0);

ItemCrom item1 = new ItemCrom(capac1,crom1);
ItemCrom item2 = new ItemCrom(item1);
```

```
System.out.println(item1);
System.out.println(item2);
```

Resultado

```
[1:10101011010111111010010010100110110110100000110111100100111001011
[-9.041936355733492E-100,[1:10101011010111111010010010100110110110100000110111100
010011100101]]
[-9.041936355733492E-100,[2:10101011010111111010010010100110110110100000110111100
010011100101]]
Press any key to continue...
```

compareTo() es un método que permite comparar 2 objetos ItemCrom que devuelve un -1 si el objeto ItemCrom que lo llama es menor, 0 si son iguales y 1 si es mayor.

```
Crom crom1 = new Crom(8);
Crom crom2 = new Crom(8);
System.out.println(crom1);
System.out.println(crom2);

double capac1 = crom1.obtenerDoubleEn(0);
double capac2 = crom2.obtenerDoubleEn(0);

ItemCrom item1 = new ItemCrom(capac1,crom1);
ItemCrom item2 = new ItemCrom(capac2,crom2);

System.out.println(item1);
System.out.println(item2);

System.out.println(item1.compareTo(item2));
```

Resultado:

```
[1:11001000]
[2:11100001]
[9.9E-322,[1:00000000000000000000000000000000000000000000000000000110010000]]
[1.11E-321,[2:00000000000000000000000000000000000000000000000000000111000001]]

-1
Press any key to continue...
```

En el caso de comparar si dos objetos representan lo mismo se puede usar el método equals

```
Crom crom1 = new Crom(8);
double capac1 = crom1.obtenerDoubleEn(0);

ItemCrom item1 = new ItemCrom(capac1,crom1);
ItemCrom item2 = item1;

System.out.println(item1.equals(item2));
```

Resultado:

```
true
Press any key to continue...
```

Si se le quiere asignar otro objeto de la clase Crom a un objeto de la clase ItemCrom:

```
Crom crom1 = new Crom(8);
Crom crom2 = new Crom(8);
System.out.println(crom1);
System.out.println(crom2);

double capac1 = crom1.obtenerDoubleEn(0);

ItemCrom item1 = new ItemCrom(capac1,crom1);
System.out.println(item1);
item1.establecerCrom(crom2);
System.out.println(item1);
```

Resultado:

```
[1:01101101]
[2:00101100]
[5.4E-322,[1:00000000000000000000000000000000000000000000000000000000011101101]]
[5.4E-322,[2:00101100]]
Press any key to continue...
```

Se puede obtener el cromosoma y capacidad de un objeto ItemCrom con los métodos obtenerDobleCapacidad y obtenerCrom.

```
Crom crom1 = new Crom(8);
double capac1 = crom1.obtenerDoubleEn(0);
ItemCrom item1 = new ItemCrom(capac1,crom1);
System.out.println(item1.obtenerDobleCapacidad());
System.out.println(item1.obtenerCrom());
```

Resultado:

```
5.93E-322
[1:00000000000000000000000000000000000000000000000000000000001111000]
Press any key to continue...
```

Clase OpGen y sus clases derivadas

La clase OpGen representa un operador genético y sirve de base para crear los otros operadores genéticos específicos (sus sub clases), está clase representará a los diversos operadores genéticos que pueda tener una población y será manejado con la clase ItemOp. La clase OpGen permite crear un operador genético para aplicarlo a los cromosomas.

Para calcar un cromosoma podemos usar la clase OpCalcar donde se especifica el tamaño del gene:

```
Crom crom1 = new Crom(12);
Crom crom2 = new Crom(12);
System.out.println(crom1);
System.out.println(crom2);
OpCalcar oper1 = new OpCalcar(3);
```

```
oper1.aplicar(crom1,crom2);
System.out.println(crom1);
System.out.println(crom2);
```

Resultado:

```
[1:000110101001]
[2:010110001010]
[1:000110101010]
[2:010110001010]
Press any key to continue...
```

OpCambio es una clase que representa un operador genético unario que al azar suma o resta 1 del valor de un gene aleatorio en un cromosoma y hace lo opuesto al próximo gene en el cromosoma.

```
Crom crom1 = new Crom(12);
Crom crom2 = new Crom(12);

System.out.println(crom1);
System.out.println(crom2);

OpCambio op = new OpCambio(6);
op.aplicar(crom1);
System.out.println(crom1);

op.aplicarEn(2,crom2);
System.out.println(crom2);
```

Resultado:

```
[1:101000010000]
[2:001101000101]
[1:101001001111]
[2:001110000100]

Press any key to continue...
```

OpCruzBits es una clase que representa un operador genético binario que realiza un cruzamiento aleatorio a nivel de bits entre dos cromosomas con una tasa de cruzamiento dada:

```
Crom crom1 = new Crom(12);
Crom crom2 = new Crom(12);

System.out.println(crom1);
System.out.println(crom2);

OpCruzBits oper = new OpCruzBits(50.0);

oper.aplicar(crom1,crom2);

System.out.println(crom1);
System.out.println(crom2);

Crom crom3 = new Crom(12);
Crom crom4 = new Crom(12);
```

```
System.out.println(crom3);
System.out.println(crom4);

oper.aplicarEn(4,crom3,crom4);

System.out.println(crom3);
System.out.println(crom4);
```

Resultado:

```
[1:100011111000]
[2:101010110100]
[1:101010110100]
[2:101010110100]
[3:110010100010]
[4:000110000000]
[3:000110000000]
[4:000110000000]
Press any key to continue...
```

OpCruzGene es una clase que representa un operador genético binario que realiza un cruzamiento de gene simple entre dos cromosomas.

```
Crom crom1 = new Crom(16);
Crom crom2 = new Crom(16);

System.out.println(crom1);
System.out.println(crom2);

OpCruzGene oper = new OpCruzGene(8);

oper.aplicar(crom1,crom2);

System.out.println(crom1);
System.out.println(crom2);

Crom crom3 = new Crom(16);
Crom crom4 = new Crom(16);

System.out.println(crom3);
System.out.println(crom4);

oper.aplicarEn(4,crom3,crom4);

System.out.println(crom3);
System.out.println(crom4);
```

Resultado:

```
[1:1010010011111010]
[2:1010110110000101]
[1:1010010010000101]
[2:1010110111111010]
[3:0011011100111011]
[4:0100001110010010]
[3:0011011110010010]
[4:0100001100111011]
Press any key to continue...
```

OpCruzPt es una clase que representa un operador genético binario que realiza un cruzamiento de n puntos entre dos cromosomas. Los límites del gene no se respetan.

```
Crom crom1 = new Crom(18);
Crom crom2 = new Crom(18);

System.out.println(crom1);
System.out.println(crom2);

OpCruzPt oper = new OpCruzPt(3);

oper.aplicar(crom1,crom2);

System.out.println(crom1);
System.out.println(crom2);
```

Resultado:

```
[1:001101001011001001]
[2:001000010010001011]
[1:001000001011001011]
[2:001101010010001001]
Press any key to continue...
```

OpDes es una clase que representa un operador genético unario que aleatoriamente suma o resta 1 del valor de un gene seleccionado en un cromosoma, se especifica el tamaño del gene al crear el operador.

```
Crom crom1 = new Crom(16);
System.out.println(crom1);

OpDesp oper = new OpDesp(8);

oper.aplicar(crom1);
System.out.println(crom1);

Crom crom2 = new Crom(16);
System.out.println(crom2);

oper.aplicarEn(6,crom2);
System.out.println(crom2);
```

Resultado:

```
[1:0001011001010000]
[1:0001011101010000]
[2:1010101011111111]
[2:1010101011111110]
Press any key to continue...
```

OpDup es una clase que representa un operador genético unario que duplica un bit de un gene al azar.

```
Crom crom1 = new Crom(24);
Crom crom2 = new Crom(24);

OpDup oper = new OpDup(8);
```

```
System.out.println(crom1);
oper.aplicar(crom1);
System.out.println(crom1);

System.out.println(crom2);
oper.aplicarEn(5,crom2);
System.out.println(crom2);
```

Resultado:

```
[1:0101001101010000101011]
[1:0101001100101000010101011]
[2:0110110001010000000110000]
[2:01101100010100000000110000]
Press any key to continue...
```

OpElim es una clase que representa un operador genético unario que quita un único gene al azar, aunque siempre deja por lo menos uno.

```
Crom crom1 = new Crom(24);
Crom crom2 = new Crom(24);

OpElim oper = new OpElim(8);

System.out.println(crom1);
oper.aplicar(crom1);
System.out.println(crom1);

System.out.println(crom2);
oper.aplicarEn(3,crom2);
System.out.println(crom2);
```

Resultado:

```
[1:000101100100100000110001]
[1:0100100000110001]
[2:001111000111101010111111]
[2:0011110001111010]
Press any key to continue...
```

OpIncAleat es una clase que representa un operador genético unario que agrega un gene aleatoriamente al final del cromosoma.

```
Crom crom1 = new Crom(16);
Crom crom2 = new Crom(16);

OpIncAleat oper = new OpIncAleat(8);

System.out.println(crom1);
oper.aplicar(crom1);
System.out.println(crom1);

System.out.println(crom2);
oper.aplicarEn(3,crom2);
System.out.println(crom2);
```

Resultado:

```
[1:1111001101100011]
[1:000001011111001101100011]
```

```
[2:0011001000011010]
[2:1000001000110010000011010]
Press any key to continue...
```

OpMut es una clase que representa un operador genético unario que muta un cromosoma a una tasa dada.

```
Crom crom1 = new Crom(24);
Crom crom2 = new Crom(24);

OpMut oper = new OpMut(8);

System.out.println(crom1);
oper.aplicar(crom1);
System.out.println(crom1);

System.out.println(crom2);
oper.aplicarEn(3,crom2);
System.out.println(crom2);
```

Resultado:

```
[1:110011100100001101011110]
[1:101010110000000101011110]
[2:011110110101001111001110]
[2:001101100101001110000111]
Press any key to continue...
```

OpTransp es una clase que representa un operador genético unario que transpone segmentos de tamaño fijo (genes) alrededor del interior de un cromosoma.

```
Crom crom1 = new Crom(24);
Crom crom2 = new Crom(24);

OpTransp oper = new OpTransp(8);

System.out.println(crom1);
oper.aplicar(crom1);
System.out.println(crom1);

System.out.println(crom2);
oper.aplicarEn(1,crom2);
System.out.println(crom2);
```

```
[1:100110111100111010001000]
[1:110011101001101110001000]
[2:110110000111111001111111]
[2:011111110111111011011000]
Press any key to continue...
```

Clase ItemOp

ItemOp es una clase que permite asociar un operador genético y su peso (usados para predisponer la selección entre operadores que compiten). Se puede comparar con el método compareTo si dos objetos de la clase ItemOp son iguales, menor o mayor, el método establecerOp y establecerPeso per-

miten asociar un operador y peso, obtenerOp y obtenerPeso permiten obtener el operador y el peso.

```
Crom crom1 = new Crom(12);
Crom crom2 = new Crom(12);
System.out.println(crom1);
System.out.println(crom2);

OpCruzBits op1 = new OpCruzBits(0.70);
OpMut op2 = new OpMut(0.80);

ItemOp item1 = new ItemOp (0.4,op1);
ItemOp item2 = new ItemOp (1.5,op2);

System.out.println(item1.compareTo(item2));

item2.establecerOp(op1);
item2.establecerPeso(0.4);

System.out.println(item1.compareTo(item2));

item1.obtenerOp().aplicar(crom1,crom2);
System.out.println(crom1);
System.out.println(crom2);

System.out.println(item1.obtenerPeso());
```

Resultado:

```
[1:111110100000]
[2:101111111101]
-1
0
[1:101111111100]
[2:101111111101]
Press any key to continue...
```

Clase Vista y sus clases derivadas

La clase Vista sirve de base para las otras vistas (clases derivadas) que permiten una interpretación específica del cromosoma o parte de esta. Las subclases de vista se usan en la aplicación que se basa en algoritmos genéticos.

La vista por defecto (tal como se provee por esta clase base), simplemente trata al cromosoma como una cadena de bits de genes simples, cada uno codificando un boolean. Interpretaciones más útiles se pueden codificar por las subclases.

```
Crom crom1 = new Crom(10);
System.out.println(crom1);

Vista vista1 = new Vista();
```

```
System.out.println(vista1.obtenerGene(crom1,4));
System.out.println(vista1.tamaño(crom1));
System.out.println(vista1.toString(crom1));
```

Resultado:

```
[1:1101110111]
true
10
[1:1101110111]
Press any key to continue...
```

Las clases derivadas de Vista se usan para imponer cierta interpretación en un cromosoma. Sus clases derivadas redefinen los métodos.

La clase VistaFija es una vista que interpreta cromosomas como una cadena de genes de tamaño fijo, codificando los valores de punto flotante en un rango dado.

```
Crom crom1 = new Crom(90);
System.out.println(crom1);

//Crea una vista de un cromosoma de tamaño 10 con valores de tipo
//double entre 20 y 20+980=1000
VistaFija vista1 = new VistaFija(10,20,980);

//Obtiene el valor del gene en el índice 0 de crom. El resultado es
//un Double que soporta el valor codificado por ese gene interpretado
//para caer dentro del rango requerido.
System.out.println(vista1.obtenerGene(crom1,0));

//Devuelve una cadena que representa a los valores de los Double contenidos
//en los genes
System.out.println(vista1.toString(crom1));

//Devuelve el número de genes que contiene esta vista en crom1
System.out.println(vista1.tamaño(crom1));
```

Resultado:

```
[1:1111000101000111101011100111111011011111001001110011101000101000100100001010000001
01110010]
374.4477028347996
374.4477028347996,173.27468230694038,544.9657869012708,910.909090909091,169.4428
1524926686,724.1055718475073,908.0351906158359,136.87194525904204,944.4379276637
342
9
Press any key to continue...
```

La clase VistaInt es una vista que interpreta los cromosomas como una cadena de genes de tamaño fijo, codificando los valores numéricos enteros.

```
Crom crom1 = new Crom(12);
System.out.println(crom1);

//Crea una vista de un cromosoma de tamaño 4 con valores de tipo
//Long que no están signados (false)
VistaInt vista1 = new VistaInt(4,false);

//Obtiene el valor del gene en el índice 0 de crom. El resultado es
```

```
//un Long que soporta el valor codificado por ese gene
System.out.println(vista1.obtenerGene(crom1,0));

//Devuelve una cadena que representa a los valores de
//los Long contenidos en los genes
System.out.println(vista1.toString(crom1));

//Devuelve el número de genes que contiene esta vista en crom1
System.out.println(vista1.tamaño(crom1));

Resultado:

[1:001100011101]
13
13,1,3
3
Press any key to continue...
```

Una vista que interpreta un cromosoma como un cierto número de genes de tamaño variable, cada uno de los cuales codifica un valor Long (posiblemente truncado).

```
Crom crom1 = new Crom(12);
System.out.println(crom1);
//Se define los puntos límites para los genes
int[] a = {0,5,10};
//Se crea la vista en función a estos puntos límite
VistaVar vista1 = new VistaVar(a);
//Se obtiene el gene en el índice 0
System.out.println(vista1.obtenerGene(crom1,0));
//Se obtienen los valores Long de crom1 de acuerdo a la vista
System.out.println(vista1.toString(crom1));
//La cantidad de genes que se generan
System.out.println(vista1.tamaño(crom1));

Resultado:

[1:110001111100]
28
28,3
2
Press any key to continue...
```

Clase Poblacion

Esta clase representa una población de cromosomas, esta es una de las clases más importantes del algoritmo genético.

El siguiente código crea una población con 5 cromosomas con un tamaño de gene igual a 3, con un mínimo de genes igual a 4, un máximo de genes igual a 4 y una tasa de mutación de 0.5 (por defecto crea un operador de cruce de 2 puntos cuando se le coloca una tasa de mutación mayor a cero). Además creamos un iterador sobre todos los individuos de la población.

```
Poblacion miPob = new Poblacion(5,3,4,4,0.05);
```

```
Iterator miIter = miPob.iteradorTodo();

while(miIter.hasNext()){
       System.out.println(miIter.next());
}
```

Resultado:

```
[0.0,[1:0101101111000]]
[0.0,[2:1101001010111]]
[0.0,[3:1001001111110]]
[0.0,[4:1000100011101]]
[0.0,[5:0010101101100]]
Press any key to continue...
```

Para vaciar la población y dejarlas sin individuos se usa el método clear():

```
Poblacion miPob = new Poblacion(5,3,4,4,0.05);

Iterator miIter = miPob.iteradorTodo();

while(miIter.hasNext()){
       System.out.println(miIter.next());
}

miPob.clear();
miIter = miPob.iteradorTodo();
System.out.println("\nPoblacion:");
while(miIter.hasNext()){
       System.out.println(miIter.next());
}
```

Resultado:

```
[0.0,[1:0101101111000]]
[0.0,[2:1101001010111]]
[0.0,[3:1001001111110]]
[0.0,[4:1000100011101]]
[0.0,[5:0010101101100]]

Poblacion:
Press any key to continue...
```

Para establecer una tasa de elitismo (parte de la población que no cambia) tenemos el método establecerTasaElitismo(). El método establecerTasaIncapacidad() determina la proporción de la población incapacitado para reproducirse. Para crear una nueva generación lo hacemos con el método nuevaGeneracion(). Si se quiere establecer un iterador sobre los elementos nuevos usamos el método iteradorSoloNuevos().

```
Poblacion miPob = new Poblacion(10,3,4,4,0.05);

Iterator miIter = miPob.iteradorTodo();

while(miIter.hasNext()){
       System.out.println(miIter.next());
```

```
}

miPob.establecerTasaElitismo(0.50);
miPob.establecerTasaIncapacidad(0.20);
miPob.nuevaGeneracion();

miIter = miPob.iteradorSoloNuevos();

System.out.println("Nueva Poblacion:");
while(miIter.hasNext()){
      System.out.println(miIter.next());
}

miIter = miPob.iteradorTodo();

System.out.println("Toda la Poblacion:");
while(miIter.hasNext()){
      System.out.println(miIter.next());
}
```

Resultado:

```
[0.0,[1:110000000011]]
[0.0,[2:111011011001]]
[0.0,[3:110101101010]]
[0.0,[4:000001100000]]
[0.0,[5:010011010111]]
[0.0,[6:011111110011]]
[0.0,[7:010100010101]]
[0.0,[8:001000011010]]
[0.0,[9:011101001010]]
[0.0,[10:110011100010]]
Nueva Poblacion:
[0.0,[16:110101101010]]
[0.0,[17:011111010011]]
[0.0,[19:010101010010]]
[0.0,[21:110011101010]]
[0.0,[23:011101001010]]
Toda la Poblacion:
[0.0,[11:011111110011]]
[0.0,[12:010100010101]]
[0.0,[13:001000011010]]
[0.0,[14:011101001010]]
[0.0,[15:110011100010]]
[0.0,[16:110101101010]]
[0.0,[17:011111010011]]
[0.0,[19:010101010010]]
[0.0,[21:110011101010]]
[0.0,[23:011101001010]]
Press any key to continue...
```

Para añadir un Nuevo operador para la población usamos el método añadirOp() y le asignamos un peso de 0.20:

```
Poblacion miPob = new Poblacion(10,3,4,4,0.05);

miPob.establecerTasaElitismo(0.40);
miPob.establecerTasaIncapacidad(0.10);
OpGen op1 = new OpDesp(3);
miPob.añadirOp(op1,0.20);
```

```
miPob.nuevaGeneracion();

miIter = miPob.iteradorTodo();

System.out.println("Toda la Poblacion:");
while(miIter.hasNext()){
       System.out.println(miIter.next());
}
```

Para eliminar la población actual y crear una nueva población con tamaños de gene igual a 2, con un mínimo de genes igual a 4 y un máximo de genes igual a 4, usamos el método reset():

```
Poblacion miPob = new Poblacion(10,3,4,4,0.05);

Iterator miIter = miPob.iteradorTodo();

System.out.println("Toda la Poblacion:");
while(miIter.hasNext()){
       System.out.println(miIter.next());
}

miPob.reset(2,4,4);

miIter = miPob.iteradorTodo();

System.out.println("Toda la Poblacion:");
while(miIter.hasNext()){
       System.out.println(miIter.next());
}
```

Resultado:

```
Toda la Población:
[0.0,[1:110111000101]]
[0.0,[2:101100011101]]
[0.0,[3:100011110101]]
[0.0,[4:100110101010]]
[0.0,[5:001111110101]]
[0.0,[6:111110110000]]
[0.0,[7:000110000011]]
[0.0,[8:111000011001]]
[0.0,[9:000111011100]]
[0.0,[10:100001000101]]
Toda la Población:
[0.0,[11:00111101]]
[0.0,[12:00101101]]
[0.0,[13:10100110]]
[0.0,[14:11001010]]
[0.0,[15:11100000]]
[0.0,[16:10110111]]
[0.0,[17:11000001]]
[0.0,[18:00010101]]
[0.0,[19:10101111]]
[0.0,[20:10101110]]
Press any key to continue...
```

Los métodos seleccionarCromosoma() y seleccionarOperador() se utilizan por el método nuevaGeneracion() para seleccionar un cromosoma con quien reproducirse y seleccionar un operador genético para la reproducción.

3.4. CLASES PARA REPRESENTAR EL RESORTE HELICOIDAL DE COMPRESIÓN

Se han creado dos clases, una de las clases (ResorteHelicoidalCompresion) está específicamente la relacionada con el modelamiento de las características y propiedades de un resorte helicoidal de compresión.

La otra clase (DatosRHC) almacena los datos correspondientes a los datos de:

- Diámetros de alambres para resortes helicoidales de compresión tanto en pulgadas como en milímetros (según la Tabla II-2)
- Tensiones de diseño para resortes helicoidales de compresión por material y según tipo de servicio. (según las figuras del II-24 al II-29).
- Lf/Dm versus Razón crítica (según la Figura II-30).

El código completo para estás clases se encuentran en el Anexo C.
Se describirá a continuación como se implementó la clase ResorteHelicoidal de Compresión.

Clase ResorteHelicoidalCompresion

Se declaran variables que sirven para modelar en los resortes el tipo de material, tipo de servicio, tipo de extremo, tipo de sujeción en los extremos, y módulo de elasticidad ante esfuerzo de corte.

```
private int material = 2;
private int tipo_servicio = 2;
private int tipo_extremo = 4;
private int sujecion = 1;
private double G = 11.85E6;      //Módulo del alambre del resorte
```

Estas variables se inicializan por defecto para que el usuario de estos pueda utilizarlos si es que no tiene muy claro que valores utilizar, el valor de 2 para el tipo de material significa que utilizará un alambre ASTM A228, el tipo de

servicio es el servicio promedio (valor 2), los extremos están en bruto (valor 4), y la sujeción en los extremos es fija (valor 1) y el modulo de elasticidad ante esfuerzo de corte es de 11.85E06 psi (para el material ASTM A228).

Otras variables (Ver Capitulo II) que se utilizan en esta clase son las siguientes:

```
private double Lf;          //Longitud libre
private double OD;          //Diámetro exterior
private double ID;          //Diámetro interior
private double Dw;          //Diámetro del alambre
private double Dm;          //Diámetro medio
private double Fs;          //Fuerza en longitud comprimido
private double Li;          //Longitud instalado;
private double Fi;          //Fuerza en longitud instalado
private double Ff;          //Fuerza en longitud libre
private double k;           //Razón de resorte
private double C;           //Índice de resorte
private int N;              //Número total de bobinas
private int Na;             //Número de bobinas activas
private double Namax;       //Número máximo de bobinas activas
private double p;           //Espaciamiento
private double ae;          //Angulo de espaciamiento
private double ODs; //Diámetro exterior en longitud comprimido
private double cc;          //Margen de bobina
private double K;           //Factor de Wahl
private double Fo;          //Fuerza en longitud de operación
private double So;          //Esfuerzo o tensión en carga de operación
private double fo;          //Deflexión en longitud de operación
private double Lo;          //Longitud de operación
private double Ls;          //Longitud comprimido
private double Ss;          //Tensión o esfuerzo en longitud comprimido;
private double Sd;          //Tensión de diseño
private double Smax;        //Tensión máxima permisible
private double Do = 1E6;    //Diámetro del orificio de instalación
private double Dv = 0;      //Diámetro de la varilla de instalación
private double RC;          //Razón Crítica
private double vol; //Volúmen
private double pen; //Penalización
```

La variable pen que representa la penalización acumula 1E06 por cada restricción de diseño que no se cumple, está variable resulta útil al momento de hacer el modelo de optimización.

Además se consideran algunas variables booleanas (pueden tomar un valor verdadero o falso) para verificar si algunas variables violan las restricciones asociadas a estas, además las inicializamos con un valor false (falso). Así tenemos:

```
private boolean C_OK = false;
private boolean OD_OK = false;
private boolean ID_OK = false;
private boolean No_Pandeo = false;
```

```
private boolean So_OK = false;
private boolean Ls_OK = false;
private boolean Ss_OK = false;
private boolean cc_OK = false;
private boolean Na_OK = false;
```

Para asignar el tipo de Material consideramos los materiales de los cuales tenemos los datos completos como para poder realizar el diseño y le asignamos una identificación para su asignación:

- 1 Para ASTM A227 Clase II
- 2 Para ASTM A228
- 3 Para ASTM A229
- 4 Para ASTM A231
- 5 Para ASTM A401
- 6 Para ASTM A313 (A302)
- 7 Para ASTM A304
- 8 Para ASTM A316

Al asignar el material también asignamos el módulo de elasticidad ante esfuerzo de corte, además si no se asignará un valor valido para el tipo de material se asume que por defecto el material es ASTM A228. Para ello implementamos el siguiente procedimiento:

```
public void asignarMaterial(int id_mat){

        if (id_mat >= 1 && id_mat <= 8)
                material = id_mat;
        else
                material = 2;

        asignarG(material);

        if (Dw > 0) {
                asignarDw(Dw);
                asignarSd(Dw,material,tipo_servicio);
                asignarSmax(Dw,material);
        }

}
```

Para asignar el tipo de servicio del resorte creamos un método (Vea en el capítulo II.- Tipos de cargas y tensiones o esfuerzos permisibles) que le asigna:

- 1 Para Servicio Ligero
- 2 Para Servicio Promedio
- 3 Para Servicio Severo

El código del método es:

```
public void asignarTipoServicio(int id_ser){
        if (id_ser >= 1 && id_ser <= 3)
                tipo_servicio = id_ser;
        else
                tipo_servicio = 2;

        if (Dw > 0) {
                asignarDw(Dw);
                asignarSd(Dw,material,tipo_servicio);
                asignarSmax(Dw,material);
        }
}
```

Se puede observar que si no se asigna un tipo de servicio válido, por defecto se asume que el servicio es el promedio.

El tipo de extremo del resorte tiene asociado los valores siguientes:

- 1 para extremos a escuadra y lijados
- 2 para solo extremos a escuadra
- 3 para extremos en bruto y lijados
- 4 para extremos en bruto

Si no se le asigna un valor válido, por defecto se le asigna extremos en bruto (valor igual a 4). El código respectivo es:

```
public void asignarTipoExtremo(int id_ext){
        if (id_ext >= 1 && id_ext <= 4)
                tipo_extremo = id_ext;
        else
                tipo_extremo = 4;
}
```

Respecto al tipo de sujeción en los extremos del resorte se le asigna:

- 1 para extremos fijos

- 2 para un extremo fijo y un extremo atornillado
- 3 para ambos extremos atornillados

En el caso de no asignarle un valor válido, se asume por defecto que ambos extremos están fijos. El código es el que sigue:

```
public void asignarSujecion(int id_suj){
	if (id_suj >= 1 && id_suj <= 3)
		sujecion = id_suj;
	else
		sujecion = 1;
}
```

El Modulo de elasticidad en corte (G) del alambre para resortes se asigna dependiendo del tipo de material (Vea la Tabla II-4):

```
private void asignarG (int i){
	if (i==1)
		G = 11.5E6;
	else if (i==2)
		G = 11.85E6;
	else if (i==3)
		G = 11.2E6;
	else if (i==4)
		G = 11.2E6;
	else if (i==5)
		G = 11.2E6;
	else if (i==6)
		G = 10.0E6;
	else if (i==7)
		G = 11.2E6;
	else
		G = 11.2E6;
}
```

El método asignarFo() asigna la fuerza en longitud de operación (Fo = Fuerza en longitud de operación) y el método asignarLo() asigna la longitud de operación (Lo = longitud de operación). Si es necesario calcular la longitud de operación en función de la longitud libre y la deflexión tenemos el método:

```
private void calcularLo(){
	Lo = Lf - fo;
}
```

El método asignarFi() asigna la Fuerza en longitud instalado (Fi = Fuerza en longitud de instalado) y el método asignarLi() asigna la Longitud instalado (Li = Longitud de instalado).

El método asignarDo() asigna el diámetro del orificio donde se instalará el resorte (Do = Diámetro del orificio) y el método asignarDv() asigna el diámetro de la varilla donde se instalará el resorte (Dv = Diámetro de la varilla).

Para calcular la razón de resorte (k = razón de resorte) tomamos en cuenta la Ec. II-1:

```
private void calculark(){
      k = (Fo-Fi)/(Li-Lo);
}
```

De la Ec. II-1 también podemos calcular la longitud libre (Lf = Longitud libre):

```
private void calcularLf(){
      Lf = Li + Fi/k;
}
```

En el caso de la deflexión en longitud de operación (fo = deflexión en longitud de operación) se tiene el método:

```
private void calcularfo(){
      fo = Lf - Lo;
}
```

El diámetro del alambre del resorte (Dw = Diámetro del alambre) debe asignarse tomando en cuenta los diámetros disponibles para los alambres según el tipo de material (vea la Tabla II-2), los diámetros del alambre se encuentran alamacenados en vectores de la clase DatosRHC. Si se asigna un valor no disponible, por defecto se le asigna el valor más cercano, esto es posible a través del método buscarAsignar(), también se le asigna la tensión de diseño y la tensión máxima permisible con los métodos asignarSd() y asignarSmax(). El método asignarDw() permite la asignación del diámetro del alambre del resorte:

```
public void asignarDw(double diaAla){

      if ( material == 1 )
      {
            Arrays.sort(DatosRHC.diaA227pulg);
            buscarAsignar(DatosRHC.diaA227pulg,diaAla);
      }
      else if (material == 2 )
      {
            Arrays.sort(DatosRHC.diaA228pulg);
            buscarAsignar(DatosRHC.diaA228pulg,diaAla);
      }
      else if (material == 3 )
      {
            Arrays.sort(DatosRHC.diaA229pulg);
            buscarAsignar(DatosRHC.diaA229pulg,diaAla);
      }
      else if (material == 4 )
      {
            Arrays.sort(DatosRHC.diaA231pulg);
            buscarAsignar(DatosRHC.diaA231pulg,diaAla);
      }
```

```
        else if (material == 5 )
        {
              Arrays.sort(DatosRHC.diaA401pulg);
              buscarAsignar(DatosRHC.diaA401pulg,diaAla);
        }
        else if (material == 6 )
        {
              Arrays.sort(DatosRHC.diaA313pulg);
              buscarAsignar(DatosRHC.diaA313pulg,diaAla);
        }
        else if (material == 7 )
        {
              Arrays.sort(DatosRHC.diaA304pulg);
              buscarAsignar(DatosRHC.diaA304pulg,diaAla);
        }
        else if (material == 8 )
        {
              Arrays.sort(DatosRHC.diaA316pulg);
              buscarAsignar(DatosRHC.diaA316pulg,diaAla);
        }
        asignarSd(Dw,material,tipo_servicio);
        asignarSmax(Dw,material);
}
```

Para hacer la búsqueda del diámetro del alambre entre los valores disponibles, el método buscarAsignar() utiliza el algoritmo de búsqueda binaria, este algoritmo de búsqueda se utiliza cuando los datos están ordenados ascendentemente, los datos se ordenan ascendentemente con el método sort() de la clase Array que dispone Java. El código del método buscarAsignar() es:

```
private void buscarAsignar(double[] datos, double diaAla){

        int bajo = 0 ;
        int alto = datos.length - 1 ;
        int medio = ( bajo + alto + 1 ) / 2 ;
        int ubicacion = -1;

        do
        {
              if ( diaAla == datos[ medio ] )
                    ubicacion = medio;
              else if ( diaAla < datos[ medio ] )
                    alto = medio - 1 ;
              else
                    bajo = medio + 1 ;
              medio = ( bajo + alto + 1 ) / 2 ;
  } while ( ( bajo <= alto ) && ( ubicacion == -1 ) );
        if (ubicacion == -1 )
        {
              if ( diaAla < datos[0] )
                    Dw = datos[0];
              else if ( diaAla > datos[datos.length - 1] )
                    Dw = datos[datos.length - 1];
              else
                    Dw = datos[medio];
        }
        else
              Dw = datos[ubicacion];
```

```
}
```

La tensión de diseño se puede determinar en función al diámetro del alambre, el tipo de material y el tipo de servicio que cumplirá el resorte con el método asignarSd(). Las figuras del II-24 al II-29 son gráficas que permiten encontrar estas tensiones de diseño, se trasladaron los valores de estos gráficos a tablas por cada curva que se tiene en estas tablas a la clase DatosRHC, el método asignarSd() usa estas tablas para calcular la tensión de diseño a través del método de interpolación de Lagrange, el código es el siguiente:

```
private void asignarSd(double diaAla,int mat, int tipSer){
      if (mat==1 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA227pulgKsiLig,diaAla)*1000.0;
      else if (mat==1 && tipSer==2)
            Sd = lagrange(DatosRHC.datosA227pulgKsiPro,diaAla)*1000.0;
      else if (mat==1 && tipSer==3)
            Sd = lagrange(DatosRHC.datosA227pulgKsiSev,diaAla)*1000.0;
      else if (mat==2 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA228pulgKsiLig,diaAla)*1000.0;
      else if (mat==2 && tipSer==2)
            Sd = lagrange(DatosRHC.datosA228pulgKsiPro,diaAla)*1000.0;
      else if (mat==2 && tipSer==3)
            Sd = lagrange(DatosRHC.datosA228pulgKsiSev,diaAla)*1000.0;
      else if (mat==3 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA229pulgKsiLig,diaAla)*1000.0;
      else if (mat==3 && tipSer==2)
            Sd = lagrange(DatosRHC.datosA229pulgKsiPro,diaAla)*1000.0;
      else if (mat==3 && tipSer==3)
            Sd = lagrange(DatosRHC.datosA229pulgKsiSev,diaAla)*1000.0;
      else if (mat==4 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA231pulgKsiLig,diaAla)*1000.0;
      else if (mat==4 && tipSer==2)
            Sd = lagrange(DatosRHC.datosA231pulgKsiPro,diaAla)*1000.0;
      else if (mat==4 && tipSer==3)
            Sd = lagrange(DatosRHC.datosA231pulgKsiSev,diaAla)*1000.0;
      else if (mat==5 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA401pulgKsiLig,diaAla)*1000.0;
      else if (mat==5 && tipSer==2)
            Sd = lagrange(DatosRHC.datosA401pulgKsiPro,diaAla)*1000.0;
      else if (mat==5 && tipSer==3)
            Sd = lagrange(DatosRHC.datosA401pulgKsiSev,diaAla)*1000.0;
      else if (mat==6 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA313pulgKsiLig,diaAla)*1000.0;
      else if (mat==6 && tipSer==2)
            Sd = lagrange(DatosRHC.datosA313pulgKsiPro,diaAla)*1000.0;
      else if (mat==6 && tipSer==3)
            Sd = lagrange(DatosRHC.datosA313pulgKsiSev,diaAla)*1000.0;
      else if (mat==7 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA304pulgKsiLig,diaAla)*1000.0;
      else if (mat==7 && tipSer==2)
            Sd = lagrange(DatosRHC.datosA304pulgKsiPro,diaAla)*1000.0;
      else if (mat==7 && tipSer==3)
            Sd    =    lagrange(DatosRHC.datosA304pulgKsiSev,diaAla)*1000.0;
      else if (mat==8 && tipSer==1)
            Sd = lagrange(DatosRHC.datosA316pulgKsiLig,diaAla)*1000.0;
```

```
        else if (mat==8 && tipSer==2)
                Sd = lagrange(DatosRHC.datosA316pulgKsiPro,diaAla)*1000.0;
        else if (mat==8 && tipSer==3)
                Sd = lagrange(DatosRHC.datosA316pulgKsiSev,diaAla)*1000.0;
}
```

La tensión máxima permisible se puede calcular utilizando las mismas figuras (del IV-6 al IV-11) pero considerando las curvas para servicio ligero, tal como se explicó en el capítulo IV. Al igual que en el caso anterior utilizamos los valores de las ilustraciones llevados a tablas en la clase DatosRHC, y son necesarios como datos de entrada el diámetro del alambre y el material, así el método interpola el valor deseado con el método de interpolación de Lagrange de la siguiente manera:

```
private void asignarSmax(double diaAla,int mat){
        if (mat==1)
         Smax = lagrange(DatosRHC.datosA227pulgKsiLig,diaAla)*1000.0;
        else if (mat==2)
         Smax = lagrange(DatosRHC.datosA228pulgKsiLig,diaAla)*1000.0;
        else if (mat==3)
         Smax = lagrange(DatosRHC.datosA229pulgKsiLig,diaAla)*1000.0;
        else if (mat==4)
         Smax = lagrange(DatosRHC.datosA231pulgKsiLig,diaAla)*1000.0;
        else if (mat==5)
         Smax = lagrange(DatosRHC.datosA401pulgKsiLig,diaAla)*1000.0;
        else if (mat==6)
         Smax = lagrange(DatosRHC.datosA313pulgKsiLig,diaAla)*1000.0;
        else if (mat==7)
         Smax = lagrange(DatosRHC.datosA304pulgKsiLig,diaAla)*1000.0;
        else if (mat==8)
         Smax = lagrange(DatosRHC.datosA316pulgKsiLig,diaAla)*1000.0;
}
```

En los dos métodos anteriores utilizamos el método de interpolación de Lagrange, este método de interpolación se utiliza con tres puntos para que sea de segundo orden, conectando tres puntos cercanos al valor que se quiera determinar. El código del método que permite interpolar con este método es el siguiente:

```
        private double lagrange(double[][] datos,double xint){

                int u=0,t=0;
                boolean hallado=false;

                int    n = datos.length;
                double l[] = new double[n];
                double prod1=1;
                double prod2=1;
                double y=0;

                for(u=0; u <= n-1; u++){
                        if (datos[u][0] > xint)
                        {
```

```
                        hallado = true;
                        if(u==0){
                                u=0;
                                t=u+2;
                                break;
                        }
                        else if (u==n-1) {
                                u=u-2;
                                t=u+2;
                                break;
                        } else {
                                u=u-1;
                                t=u+2;
                                break;
                        }

                }
        }

        if (hallado == false){
                t=n-1;
                u=t-2;
        }

        //con tres puntos
        for (int j=u; j<=t; j++)
        {
                for (int i=u; i<=t; i++)
                if (i!=j){
                        prod1 = prod1*(xint-datos[i][0]);
                        prod2 = prod2*(datos[j][0]-datos[i][0]);
                }
                l[j] = prod1/prod2;
                prod1 = 1;
                prod2 = 1;
                y = y + datos[j][1]*l[j];
        }
        return y;
}
```

El método asignarNa() asigna el número de bobinas activas (Na = Número de bobinas activas). Si se quisiera aproximar el número de bobinas activas podemos utilizar las recomendaciones que se nos da en el paso xiv y xv del Método 2 de diseño de resortes helicoidales de compresión del capítulo II. Además se verifica que el número de bobinas activas sea por lo menos uno. El código del método es:

```
private void calcularNa(){
        if(tipo_extremo == 1 || tipo_extremo == 2)
                Namax = (Lo - 2*Dw)/Dw;
        else if (tipo_extremo == 3)
                Namax = (Lo - Dw)/Dw;
        else if (tipo_extremo == 4)
                Namax = Lo/Dw;

        Na = (int) Math.floor(Namax);

        if(Na > 1)
```

```
                Na_OK = true;
}
```

De la ecuación II-14 podemos calcular el índice de resorte (C = índice de resorte), el índice tendrá un valor válido si es mayor a 5:

```
private void calcularC() {
        C = Math.cbrt(G*Dw/(8*k*Na));

        if ( C > 5)
                C_OK = true;
}
```

El método asignarDm() asigna el diámetro medio del resorte (Dm = Diámetro medio), en el caso de ser necesario su cálculo tenemos el método calcularDm():

```
private void calcularDm(){
        Dm = C*Dw;
}
```

Para calcular el diámetro exterior (OD = Diámetro exterior) tenemos el método calcularOD(), además si se tiene un orificio donde se instalará el resorte se debe cumplir que este debe ser mayor al diámetro exterior más la décima parte del diámetro del alambre:

```
private void calcularOD(){
        OD = Dm + Dw;

        if (Do > (OD + Dw/10))
                OD_OK = true;
}
```

Para calcular el diámetro interior (ID = Diámetro interior) tenemos el método calcularID(), si se instala el resorte en una varilla, el diámetro de la varilla debe ser menor al diámetro interior menos un décimo el diámetro del alambre:

```
private void calcularID(){
        ID = Dm - Dw;

        if(Dv < (ID - Dw/10))
                ID_OK = true;
}
```

En el caso del análisis del resorte en cuanto al pandeo, se utiliza la Figura II-30 para calcular la razón crítica (RC = razón crítica) que dependerá del tipo de sujeción en los extremos del resorte, los valores correspondientes a la figura están en la clase DatosRHC, el valor de RC se utiliza el método de

interpolación de Lagrange. Si la razón entre la deflexión y la longitud libre es menor que la razón crítica, el resorte no se pandeará:

```
private void calcularRC(){
	if (sujecion == 1){
		if( Lf/Dm < DatosRHC.RCExtremosFijos[0][0]){
			RC = 1E100;
		}
		else {
			RC = lagrange(DatosRHC.RCExtremosFijos,Lf/Dm);
		}
	}
	else if (sujecion == 2){
		if( Lf/Dm < DatosRHC.RCUnExtremoFijo[0][0]){
			RC = 1E100;
		}
		else {
			RC = lagrange(DatosRHC.RCUnExtremoFijo,Lf/Dm);
		}
	}
	else if (sujecion == 3){
		if( Lf/Dm < DatosRHC.RCExtremosAtornillados[0][0]){
			RC = 1E100;
		}
		else {
			RC= Lagrange(DatosRHC.RCExtremosAtornillados,Lf/Dm);
			}
	}
	if( fo/Lf <= RC )
		No_Pandeo = true;
}
```

El factor de Wahl (K = factor de Wahl) se calcula en base a la ecuación 7-5:

```
	private void calcularK(){
		K = (4*C - 1)/(4*C - 4) + 0.615/C;
	}
```

La tensión de operación (S_o = Tensión de operación) se calcula con la ecuación 4-11, considerando $F = F_o$ y reemplazando $C = D_m/D_w$, se considera adecuada la tensión de operación si esta es menor a la tensión de diseño:

```
	private void calcularSo(){
		So = 8*K*Fo*Dm/(Math.PI*Math.pow(Dw,3.0));

		if (So < Sd)
			So_OK = true;

	}
```

El número total de bobinas (N = número total de bobinas) se calcula considerando el tipo de extremo del resorte:

```
	private void calcularN(){
		if(tipo_extremo == 1 || tipo_extremo == 2)
			N = Na + 2;
		else if (tipo_extremo == 3)
			N = Na + 1;
```

```
        else if (tipo_extremo == 4)
                N = Na;
}
```

La longitud comprimido (Ls = longitud comprimido) se calcula de la siguiente manera (la longitud comprimido) debe ser menor a la longitud de operación:

```
private void calcularLs(){
        Ls = Dw*N;

        if(Ls < Lo)
                Ls_OK = true;
}
```

La fuerza en el resorte en longitud comprimido (Fs = Fuerza en longitud comprimido) se calcula como el producto de la razón del resorte por la deflexión en longitud de comprimido:

```
private void calcularFs(){
        Fs = k*(Lf - Ls);
}
```

Para la tensión en longitud comprimido (Ss = Tensión en longitud comprimido) consideramos que la tensión o esfuerzo en el resorte es directamente proporcional a la fuerza, además la tensión en longitud de comprimido debe ser menor a la tensión máxima permisible:

```
private void calcularSs(){
        Ss = So*Fs/Fo;

        if(Ss < Smax)
                Ss_OK = true;
}
```

El margen de bobinas (cc = margen de bobinas) se calcula según lo explicado en el Capítulo II. El margen de bobinas es correcto si es mayor a la décima parte del diámetro del alambre:

```
private void calcularcc(){
        cc = (Lo - Ls)/Na;

        if(cc > Dw/10)
                cc_OK = true;
}
```

El espaciamiento (p = espaciamiento) del resorte depende del tipo de extremo del resorte:

```
private void calcularp(){
        if (tipo_extremo == 1)
                p = (Lf - 2*Dw)/Na;
        else if (tipo_extremo == 2)
                p = (Lf - 3*Dw)/Na;
```

```
        else if (tipo_extremo == 3)
                p = Lf/(Na+1);
        else if (tipo_extremo == 4)
                p = (Lf-Dw)/Na;
}
```

El ángulo de espaciamiento (ae = ángulo de espaciamiento) se calcula con:

```
private void calcularae(){
        ae = Math.atan(p/(Math.PI*Dm));  }
```

El diámetro exterior en longitud comprimido (ODs = Diámetro exterior en longitud comprimido) se calcula con:

```
private void calcularODs(){
        ODs = Math.sqrt(Math.pow(Dm,2.0)+
                (Math.pow(p,2.0)-Math.pow(Dw,2.0))/Math.pow(Math.PI,2.0))
                        + Dw;
}
```

El método calcularvol() calcula el volumen del resorte (vol = volumen del resorte), esto depende de la geometría considerada:

```
private void calcularvol(){
        vol = 0.25*Math.pow(Math.PI,2.0)*Math.pow(Dw,2.0)*Dm*N;
}
```

La variable pen (pen = penalización del resorte) acumula 1E6 por cada restricción de diseño que no se cumpla, si se cumplen con todas las restricciones pen tiene un valor igual a cero, si no se cumplen dos restricciones pen vale 2E6, así tenemos:

```
private void calcularpen(){
        if ( C_OK == false )
                pen = pen + 1E6;
        if ( OD_OK == false )
                pen = pen + 1E6;
        if ( ID_OK == false )
                pen = pen + 1E6;
        if ( No_Pandeo == false )
                pen = pen + 1E6;
        if ( So_OK == false )
                pen = pen + 1E6;
        if ( Ls_OK == false )
                pen = pen + 1E6;
        if ( Ss_OK == false )
                pen = pen + 1E6;
        if ( cc_OK == false )
                pen = pen + 1E6;
        if ( Na_OK == false )
                pen = pen + 1E6;
}
```

El método analizar() calcula las otras características del resorte cuando se tiene determinado: Material, Tipo de Extremo, Tipo de Servicio, Tipo de Sujeción en los Extremos, Longitud de Operación, Fuerza en Longitud de Ope-

ración, Longitud de Instalado, Fuerza en Longitud de Instalado y Diámetro del Alambre:

```
public void analizar(){
        calculark();
        calcularLf();
        calcularfo();
        calcularNa();
        calcularC();
        calcularDm();
        calcularOD();
        calcularID();
        calcularRC();
        calcularK();
        calcularSo();
        calcularN();
        calcularLs();
        calcularFs();
        calcularSs();
        calcularcc();
        calcularvol();
        calcularp();
        calcularODs();
        calcularae();
        calcularpen();
}
```

El método analizarAG() calcula las otras características del resorte cuando se tiene determinado: Material, Tipo de Extremo, Tipo de Servicio, Tipo de Sujeción en los Extremos, Longitud de Operación, Fuerza en Longitud de Operación, Longitud de Instalado, Fuerza en Longitud de Instalado, Diámetro del Alambre, Diámetro Medio y Numero de Bobinas Activas.

```
public void analizarAG(){
        calculark();
        calcularLf();
        calcularfo();
        calcularC();
        calcularOD();
        calcularID();
        calcularRC();
        calcularK();
        calcularSo();
        calcularN();
        calcularLs();
        calcularFs();
        calcularSs();
        calcularcc();
        calcularvol();
        calcularp();
        calcularODs();
        calcularae();
        calcularpen();
}
```

El método inicializar(), inicializa las variables que determinan las restricciones de diseño en los Resortes Helicoidales de Compresión, es útil cada vez que se quiere hacer un nuevo análisis luego de hacer algún cambio en algún valor previamente especificado:

```
public void inicializar(){
        pen = 0.0;
        C_OK = false;
        OD_OK = false;
        ID_OK = false;
        No_Pandeo = false;
        So_OK = false;
        Ls_OK = false;
        Ss_OK = false;
        cc_OK = false;
        Na_OK = false;
}
```

Complementan a estos métodos aquellos que sirven para obtener los valores de las distintas variables. La tabla V-3 describe los métodos públicos y su utilidad.

Tabla III-1. Métodos públicos de la clase ResorteHelicoidalCompresion.

Tipo devuelto	Método
Void	**analizar**() Calcula las otras características del resorte cuando se tiene determinado: Material, Tipo de Extremo, Tipo de Servicio, Tipo de Sujeción en los Extremos, Longitud de Operación, Fuerza en Longitud de Operación, Longitud de Instalado, Fuerza en Longitud de Instalado y Diámetro del Alambre.
Void	**analizarAG**() Calcula las otras características del resorte cuando se tiene determinado: Material, Tipo de Extremo, Tipo de Servicio, Tipo de Sujeción en los Extremos, Longitud de Operación, Fuerza en Longitud de Operación, Longitud de Instalado, Fuerza en Longitud de Instalado, Diámetro del Alambre, Diámetro Medio, Numero de Bobinas Activas. Este método resulta útil en el caso de la aplicación de los algoritmos genéticos
Void	**asignarDm**(double valDm) Asigna el Diámetro medio del resorte (Dm = Diámetro medio).
Void	**asignarDo**(double diaOri) Asigna el Diámetro del orificio donde se instalará el resorte (Do = Diámetro del orificio).
void	**asignarDv**(double diaVar) Asigna el Diámetro de la varilla donde se instalará el resorte (Dv =

	Diámetro de la varilla).
void	**asignarDw**(double diaAla) Asigna el Diámetro del alambre del resorte (Dw = Diámetro del alambre).
void	**asignarFi**(double fueIns) Asigna la Fuerza en longitud instalado (Fi = Fuerza en longitud de instalado).
void	**asignarFo**(double fueOpe) Asigna la fuerza en longitud de operación (Fo = Fuerza en longitud de operación).
void	**asignarLi**(double longIns) Asigna la Longitud instalado (Li = Longitud de instalado).
void	**asignarLo**(double lonOpe) Asigna la longitud de operación (Lo = longitud de operación).
void	**asignarMaterial**(int id_mat) Asigna el Tipo de Material. Para ASTM A227 Clase II asignar 1. Para ASTM A228 asignar 2. Para ASTM A229 Clase II asignar 3. Para ASTM A231 asignar 4. Para ASTM A401 asignar 5. Para ASTM A313 (A 302) asignar 6. Para ASTM A304 asignar 7. Para ASTM A316 asignar 8.
void	**asignarNa**(int valNa) Asigna el Número de bobinas activas (Na = Número de bobinas activas).
void	**asignarSujecion**(int id_suj) Asigna el tipo de sujeción en los extremos del resorte. Para extremos fijos asignar 1. Para un extremo fijo y un extremo atornillado asignar 2. Para ambos extremos atornillado asignar 3.
void	**asignarTipoExtremo**(int id_ext) Asigna el tipo de extremo del resorte.
void	**asignarTipoServicio**(int id_ser) Asigna el tipo de servicio del resorte.
void	**inicializar**() Inicializa las variables que determinan las restricciones de diseño en los Resortes Helicoidales de Compresión para hacer un nuevo análisis.
double	**obtenerae**() Devuelve el ángulo de espaciamiento ae = ángulo de espaciamiento
boolean	**obtenerC_OK**() Devuelve true si el valor del Índice de resorte está permitido
double	**obtenerC**() Devuelve el Índice del resorte C = Indice de resorte

boolean	obtenercc_OK() Devuelve true si el Margen de las Bobinas tiene un valor permitido
double	obtenercc() Devuelve el Margen de bobinas cc = Margen de bobinas
double	obtenerDm() Devuelve el Diámetro medio del resorte Dm = Diámetro medio
double	obtenerDo() Devuelve el Diámetro del Orificio.
double	obtenerDv() Devuelve el Diámetro de la varilla.
double	obtenerDw() Devuelve el diámetro del alambre del resorte.
double	obtenerFi() Devuelve la Fuerza en longitud instalado.
double	obtenerfo() Devuelve la deflexión en longitud de operación (fo = deflexión en longitud de operación).
double	obtenerFo() Devuelve la fuerza en longitud de operación.
double	obtenerFs() Devuelve la Fuerza en longitud comprimido (Fs = Fuerza en longitud comprimido).
double	obtenerG() Devuelve el módulo de elasticidad de acuerdo al tipo de material asignado.
boolean	obtenerID_OK() Devuelve true si el valor del Diámetro interno está permitido.
double	obtenerID() Devuelve el Diámetro interior (ID = Diametro interior).
double	obtenerk() Devuelve la razón del resorte (k = razón de resorte).
double	obtenerK() Devuelve el Factor de Wahl (K = Factor de Wahl).
double	obtenerLf() Devuelve la Longitud libre (Lf = Longitud libre).
double	obtenerLi() Devuelve la Longitud instalado.
double	obtenerLo() Devuelve la Longitud libre.
boolean	obtenerLs_OK() Devuelve true si la Longitud de Comprimido tiene un valor permitido.

double	`obtenerLs()` Devuelve la longitud comprimido (Ls = longitud comprimido).
int	`obtenerMaterial()` Devuelve la identificación del tipo de material.
int	`obtenerN()` Devuelve el Número total de bobinas (N = Número total de bobinas).
boolean	`obtenerNa_OK()` Devuelve true si el Número de Bobinas Activas tiene un valor permitido
int	`obtenerNa()` Devuelve el Número de bobinas activas
boolean	`obtenerNo_Pandeo()` Devuelve true si el resorte no se pandeará
boolean	`obtenerOD_OK()` Devuelve true si el valor del Diámetro externo está permitido
double	`obtenerOD()` Devuelve el Diámetro exterior OD = Diámetro exterior
double	`obtenerODs()` Devuelve el Diámetro exterior en longitud comprimido (ODs = Diámetro exterior en longitud comprimido).
double	`obtenerp()` Devuelve el Espaciamiento (p = Espaciamiento).
double	`obtenerpen()` Devuelve la penalización del resorte, por cada restricción de diseño que no se cumpla se penaliza 1E6 (pen = penalización del resorte).
double	`obtenerRC()` Devuelve la Razón Crítica del Resorte (RC = Razón Crítica).
double	`obtenerSd()` Devuelve la tensión de diseño.
double	`obtenerSmax()` Devuelve la Tensión Máxima permisible
boolean	`obtenerSo_OK()` Devuelve true si la Tensión de Operación tiene un valor permitido
double	`obtenerSo()` Devuelve la Tensión de operación (So = Tensión de operación).
boolean	`obtenerSs_OK()` Devuelve true si la Tensión de Comprimido tiene un valor permitido
double	`obtenerSs()` Devuelve la Tensión en longitud comprimido (Ss = Tensión en longitud comprimido).
int	`obtenerSujecion()`

	Devuelve la identificación del tipo de sujeción.
int	`obtenerTipoExtremo()` Devuelve la identificación del tipo de extremo.
Int	`obtenerTipoServicio()` Devuelve la identificación del tipo de servicio.
double	`obtenervol()` Devuelve el volumen del resorte (vol = volumen del resorte).

Fuente: Elaboración Propia

3.5. ALGORITMO GENÉTICO DE OPTIMIZACIÓN PARA EL DISEÑO

La aplicación que implemente el algoritmo de optimización para el diseño de resortes helicoidales de compresión deberá en primer lugar, inicializar la población con el número de genes de cada cromosoma, el tamaño de cada gene en bits, el número de generaciones en la evolución y el número de cromosomas en cada generación. Para esta aplicación consideramos 3 genes que representarán respectivamente al diámetro del alambre del resorte (Dw), diámetro medio (Dm) y número de bobinas activas (Na). Cada uno de estos genes será de tamaño 13, recuerde la discusión del apartado 3.5.3. acerca de la codificación de los números reales y su precisión; para el diámetro del alambre que va de 0.004 a 0.496 pulgadas usando una precisión de 4 decimales, necesitaríamos $\log_2(4960 - 40) \approx 13$ bits para representar cualquier número real dentro de ese rango, no obstante el diámetro del alambre tiene valores discretos a los que se aproximara el valor resultante; para el diámetro medio que va de 0.2 a .49.99 pulgadas usando una precisión de 2 decimales, necesitaríamos $\log_2(4999 - 20) \approx 13$ bits para representar cualquier número real dentro de ese rango; finalmente el número de bobinas activas es un número entero, con 13 bits podemos obtener valores entre 0 y 8191 para esta variable. Así en la aplicación tenemos, para el número de genes y el tamaño de cada gene las siguientes variables:

```
static final int numGenes = 3;
static final int tamGenes = 13;
```

Para tener 1000 generaciones en la evolución y 300 cromosomas en cada uno de ellos, consideramos:

```
static final int generaciones = 1000;
static final int numCromosomas = 300;
```

La aplicación necesita manejar las características de un resorte helicoidal de compresión para ello creamos un objeto de la clase ResorteHelicoidalCompresion:

```
ResorteHelicoidalCompresion miRes = new ResorteHelicoidalCompresion();
```

Las características deseadas asociadas al tipo de material, tipo de extremo, tipo de servicio, tipo de sujeción en los extremos, fuerza en longitud de operación, longitud de operación, fuerza en longitud instalada y longitud instalada se pueden asignar de la siguiente manera:

```
miRes.asignarMaterial(6);
miRes.asignarTipoExtremo(1);
miRes.asignarTipoServicio(2);
miRes.asignarSujecion(1);
miRes.asignarFo(14);
miRes.asignarLo(1.25);
miRes.asignarFi(1.50);
miRes.asignarLi(2);
```

Las características señaladas anteriormente se pueden modificar de acuerdo al problema de diseño de resortes helicoidales de compresión que se quiera resolver. Estas características se compartirán por la población de resortes helicoidales de compresión de cada generación.

La población de resortes helicoidales de compresión con estas características de entrada, necesita para completar sus datos de entrada; los valores correspondientes al diámetro del alambre, diámetro medio y número de bobinas activas. La complejidad del diseño de resortes helicoidales de compresión radica en encontrar estos 3 valores en forma conjunta, para ello la población se genera con estos valores de manera aleatoria dentro de los rangos especificados para que puedan evolucionar en base a los valores de las variables ya declaradas:

```
Poblacion resortes = new Poblacion(numCromosomas,tamGenes,numGenes,
      numGenes, 0.3);
```

En el código previo se asocia a la población dos operadores, el de mutación con una probabilidad de 0.30 y por defecto se le asigna un operador de cruce de dos puntos. Estos operadores permiten mantener la diversidad en la población y la evolución aleatoria de sus características.

Para interpretar los cromosomas es necesario crear vistas que representen los valores correspondientes al diámetro del alambre, diámetro medio y número de bobinas activas:

```
Vista visDw = new VistaFija(13,0.004,0.496);
Vista visDm = new VistaFija(13,0.2,49.99);
Vista visNa = new VistaInt(13,false);
```

También establecemos una tasa de elitismo, es decir una parte de la población que no cambia, que se establece en base a los individuos más aptos, es decir aquellos que tienen menor volumen en la función objetivo, además se puede establecer una tasa de incapacidad que determina el porcentaje de individuos de la población que no pueden reproducirse..

```
resortes.establecerTasaElitismo(0.10);
resortes.establecerTasaIncapacidad(0.4);
```

Luego para cada generación creamos un iterador que nos permita acceder a cada uno de los individuos de la población, se crea un iterador para el total de los elementos solamente para la primera generación, después será necesario iterar solo sobre los individuos nuevos, la variable censo sirve para este propósito. Cada 10 generaciones restablecemos los valores de capacidad asignándole un iterador para el total de individuos:

```
for (int i = 0; i < generaciones; ++i){

      Iterator censo;

      if(i == 0){
            censo = resortes.iteradorTodo();
      }
      else {
            censo = resortes.iteradorSoloNuevos();
      } ….
```

En cada generación para cada cromosoma inicializamos su valor de capacidad con cero, la variable val sirve para tal fin, la variable tmp recupera al cromosoma que se encuentra en la población al que se hace referencia en ese momento en la variable censo para posteriormente asignarle su capacidad, se aísla el cromosoma que representa una solución para poder analizarlo:

```
      while (censo.hasNext()){
            double val = 0.0;
            ItemCrom tmp = (ItemCrom) censo.next();
            Crom crom = tmp.obtenerCrom();
```

El cromosoma aislado representa al diámetro del alambre, diámetro medio y número de bobinas activas de un resorte; las demás características del resorte se determinarán a partir de estos valores generados aleatoriamente y de los datos ingresados inicialmente, inicializamos el resorte:

```
miRes.inicializar();
```

A partir de la representación binaria del cromosoma, asignamos al resorte los valores correspondientes al diámetro del alambre, diámetro medio y número de bobinas activas del resorte:

```
miRes.asignarDw(
      ((Double)visDw.obtenerGene(crom,0)).doubleValue());
miRes.asignarDm(
      ((Double)visDm.obtenerGene(crom,13)).doubleValue());
miRes.asignarNa(
      ((Long)visNa.obtenerGene(crom,26)).intValue());
```

Determinamos las otras características del resorte asociado al cromosoma:

```
miRes.analizarAG();
```

Con las otras características halladas se puede determinar la capacidad del resorte que se almacenará en la variable val, la capacidad está asociada al volumen del resorte, pues tratamos de obtener el resorte con el menor volumen posible además a este volumen lo penalizamos con 1E6 por cada restricción de diseño que no se cumpla. A través de la variable tmp asociamos la capacidad penalizada de este resorte.

```
val = - miRes.obtenervol() - miRes.obtenerpen();
tmp.establecerCapacidad(val);
```

Se puede observar en el código anterior que a val se le asigna un valor negativo, esto es porque de manera natural la mayor capacidad está asociada a un mayor valor y el que tiene más capacidad tiene mayor posibilidad de sobrevivir y reproducirse, en el caso del resorte se busca que se reproduzcan los que tienen el menor volumen posible, para conseguir ello simplemente lo que hacemos es asignarle a la capacidad el valor negativo del volumen.

Una vez determinadas y asociadas las capacidades (volumen del resorte) de los cromosomas de una población en una generación se debe determinar una nueva población en la nueva generación en base a la población anterior

que tiene asociados operadores genéticos para generar la nueva población y tomando en cuenta la tasa de elitismo y la tasa de incapacidad. A esto le llamamos evolución.

```
if ( i < generaciones - 1 )
      resortes.nuevaGeneracion();
```

Cuando la evolución llegue a su fin según el número de generaciones determinadas bastara ordenar el resultado a través de su capacidad con una lista ordenada y obtener sus valores a través de las vistas:

```
ListaOrdenada resultados = new ListaOrdenada( resortes );
ListIterator iter = resultados.iteradorLista( resultados.size() );

while( iter.hasPrevious() ){

      ItemCrom item = (ItemCrom) iter.previous();
      Crom crom = item.obtenerCrom();

      System.out.print(visDw.obtenerGene(crom,0) + " ");
      System.out.print(visDm.obtenerGene(crom,13) + " ");
      System.out.print(visNa.obtenerGene(crom,26) + " ");
      System.out.println(" = " + item.obtenerDobleCapacidad() );
}
```

El código completo del ejemplo de una aplicación para consola se encuentra en el Anexo D. En el anexo E se presenta el código de una aplicación con una interfaz gráfica.

3.6. LIMITACIONES DEL ALGORITMO GENÉTICO DE OPTIMIZACIÓN

El algoritmo encuentra el resorte óptimo cuando se le proporcionan los siguientes datos:

- Material
- Tipo de Extremo
- Tipo de Servicio
- Tipo de Sujeción en los Extremos
- Longitud de Operación
- Fuerza en Longitud de Operación

- Longitud de Instalado
- Fuerza en Longitud de Instalado
- Diámetro del orificio de instalación
- Diámetro de la varilla de instalación

En la Figura III-9 se presenta la pantalla de ingreso de datos de una aplicación con una interfaz gráfica que sirve para este propósito.

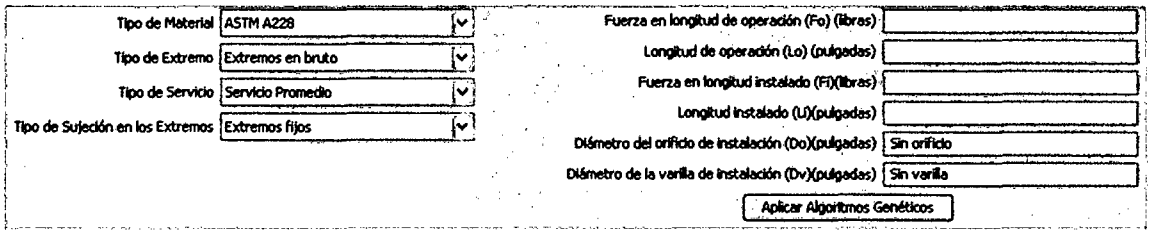


Figura III-9. Pantalla de ingreso de datos.

Si no se proporcionan los datos del resorte relacionados al material, extremo, servicio, sujeción, diámetro del orificio de instalación y diámetro de la varilla de instalación. Estos tomarán valores por defecto, el material se considerará como ASTM A228, los extremos en bruto, el servicio promedio, la sujeción con extremos fijos, sin orificio y sin varilla. Las entradas que son obligatorias son la fuerza en longitud de operación, longitud de operación, fuerza en longitud instalado y longitud instalado.

El material a elegir está restringido a ocho materiales: ASTM A227, ASTM A228, ASTM A229, ASTM A231, ASTM A401, ASTM A302, ASTM A304 y ASTM A316. Los extremos pueden ser: extremos a escuadra y lijados, extremos a escuadra, extremos en bruto y lijados, y extremos en bruto. El tipo de servicio puede ser ligero, promedio o severo. Los extremos pueden ser fijos, un extremo fijo y otro atornillado, y ambos extremos atornillados.

El algoritmo genético propuesto considera 300 individuos por generación y se tienen 1000 generaciones durante la evolución.

El algoritmo devuelve 300 soluciones entre factibles e infactibles con sus datos correspondientes a:

- Módulo de elasticidad de alambre para resorte en corte (G)
- Razón de resorte (k)
- Longitud libre (Lf)
- Deflexión en longitud de operación (fo)
- Diámetro del alambre del resorte (Dw)
- Tensión de diseño (Sd)
- Tensión máxima permisible (Smax)
- Número de bobinas activas (Na)
- Índice de resorte (C)
- Diámetro medio (Dm)
- Diámetro exterior (OD)
- Diámetro interior (ID)
- Razón Crítica (RC)
- Factor de Wahl (K)
- Esfuerzo o tensión en carga de operación (So)
- Número total de bobinas (N)
- Longitud comprimido (Ls)
- Fuerza en longitud comprimido (Fs)
- Tensión o esfuerzo en longitud comprimido (Ss)
- Margen de bobina (cc)
- Volumen (vol)
- Penalización acumulada (pen)

Además el algoritmo verifica si se cumplen con las restricciones de diseño asociadas a las siguientes variables:

- Factor de Wahl
- Índice de resorte
- Diámetro exterior
- Diámetro interior
- Pandeo
- Esfuerzo o tensión en carga de operación
- Longitud comprimido
- Tensión o esfuerzo en longitud comprimido
- Margen de bobina
- Número de espiras activas

El algoritmo puede ser modificado para poder tener entradas y salidas que se consideren apropiadas y que no estén consideradas en esta tesis. El código presentado se puede usar de manera distinta a la planteada.

3.7. ORGANIZACIÓN DE LAS CLASES DEL ALGORITMO

Las clases utilizadas en este algoritmo han sido organizadas en los siguientes paquetes:

Paquete agjava.util

Tabla III-2. Interfaces del paquete agjava.util.

Interfaces	
Collection	Un duplicado de la interfaz Collection del JDK1.4 para la portabilidad posterior y la comprensión del uso de esta.
Comparable	Esta interfaz es implementada por las clases que pueden comparar con otros objetos de su clase.
Comparator	Esta interfaz es implementada por las clases que pueden comparar elementos.
Iterator	Esta interfaz es implementada por los iteradores sobre las colecciones.
ListIterator	Esta interfaz es implementada por los iteradores sobre las co-

	lecciones.

Fuente: Elaboración Propia

Tabla III-3. Clases del paquete agjava.util

Clases	
AleatorioInt	Una clase utilitaria para proveer un generador de números aleatorios.
BitSetExtendido	Clase de cadena de bits que implementa un conjunto más flexible de operadores que java.util.BitSet, si bien este implementa todas las rutinas de esa clase para que se haga compatible al usarlo directamente.
ListaOrdenada	Extiende ListaSimple para manejar elementos en orden ascendente (parcial) basado en el resultado de llamar a compareTo en el objeto o al usar un Comparator provisto en la construcción.
ListaSimple	Una implementación simple de una lista y su clase iterador.
ListaSimple.Nodo	Un nodo de una lista

Fuente: Elaboración Propia

Tabla III-4. Excepciones del paquete agjava.util

Excepciones	
ConcurrentModificationException	Lanzado por el iterador para ListaSimple en el caso de modificación concurrente.

Fuente: Elaboración Propia

Paquete agjava.ag

Tabla III-5. Clases del paquete agjava.ag

Crom	Una clase que representa un cromosoma y tiene varias formas apropiadas para construir y manipular el cromosoma.
ItemCrom	Un cromosoma y la valuación de su capacidad
ItemOp	Un operador genético y su peso (usados para predisponer la selección entre operadores que compiten).
OpCalcar	Una clase que representa un operador genético binario que copia un segmento de tamaño fijo (gene) de un cromosoma a otro.
OpCambio	Una clase que representa un operador genético unario que al

	azar suma o resta uno del valor de un gene aleatorio en un cromosoma y hace lo opuesto al próximo gene en el cromosoma.
OpCruzBits	Una clase que representa un operador genético binario que realiza un cruzamiento aleatorio a nivel de bits entre dos cromosomas.
OpCruzGene	Una clase que representa un operador genético binario que realiza un cruzamiento de gene simple entre dos cromosomas.
OpCruzPt	Una clase que representa un operador genético binario que realiza un cruzamiento de n puntos entre dos cromosomas.
OpDesp	Una clase que representa un operador genético unario que aleatoriamente suma o resta uno del valor de un gene seleccionado en un cromosoma.
OpDup	Una clase que representa un operador genético unario que duplica un bit de un gene al azar.
OpElim	Una clase que representa un operador genético unario que quita un único gene al azar, aunque siempre deja por lo menos uno.
OpGen	Una clase que representa un operador genético que puede manipular un cromosoma.
OpIncAleat	Una clase que representa un operador genético unario que agrega un gene aleatoriamente al final del cromosoma.
OpMut	Una clase que representa un operador genético unario que muta un cromosoma a una tasa dada.
OpTransp	Una clase que representa un operador genético unario que transpone segmentos de tamaño fijo (genes) alrededor del interior de un cromosoma.
Poblacion	Una clase básica para representar una población de cromosomas.
Vista	Las clases derivadas de Vista se usan para imponer cierta interpretación en un cromosoma.
VistaFija	Una vista que interpreta cromosomas como una cadena de genes de tamaño fijo, codificando los valores de punto flotante en un rango dado
VistaInt	Una vista que interpreta los cromosomas como una cadena de genes de tamaño fijo, codificando los valores numéricos enteros.
VistaVar	Una vista que interpreta un cromosoma como un cierto número de genes de tamaño variable, cada uno de los cuales codi-

	fica un valor long (posiblemente truncado).

Fuente: Elaboración Propia

Paquete agrhc

Tabla III-6. Clases del paquete agrhc

DatosRHC	Almacena los datos correspondientes a los materiales, diámetros de los alambres, las tensiones y las tensiones por esfuerzo de corte de diseño. También se almacena información correspondiente a la razón crítica que se usa para determinar si habrá pandeo.
ResorteHelicoidalCompresion	Representa a los resortes helicoidales de compresión, tiene los métodos necesarios para manejarlos.

Fuente: Elaboración Propia

Con estos paquetes se implementó el algoritmo de optimización para el diseño de resortes helicoidales de compresión.

3.8. USO DEL PROGRAMA QUE APLICA EL ALGORITMO DE OPTIMIZACIÓN

El programa que tiene una interfaz gráfica presenta tres pestañas (vea la Figura III-10). La primera pestaña agrupa el conjunto necesario de elementos que hacen posible ingresar los datos y aplicar los algoritmos genéticos generando 300 posibles respuestas. La segunda pestaña permite seleccionar cualquiera de los resultados obtenidos y analizarlos en detalle: La tercera pestaña permite hacer un análisis donde se tengan que ingresar todos los datos de diseño, incluidos los valores del diámetro del alambre del resorte, diámetro medio y número de bobinas activas del resorte.

Figura III-10. Pestañas de opciones.

Al elegir la primera pestaña: "Diseño de Resortes" podemos ingresar los datos correspondientes para el diseño y aplicar los algoritmos genéticos sobre

el conjunto de datos obteniendo 300 soluciones entre factibles e infactibles (véase la Figura III-11).

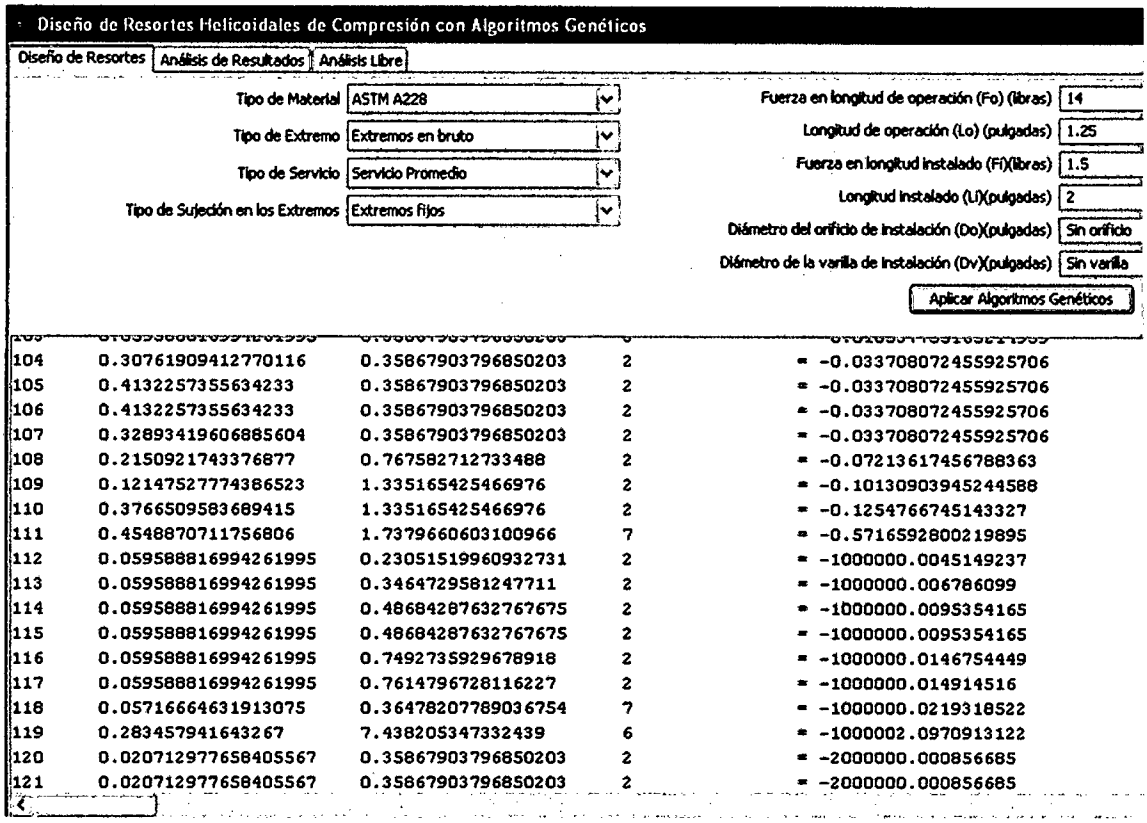


Figura III-11. Pantalla de aplicación de algoritmos genéticos.

En la salida de los resultados la primera columna corresponde a una identificación que se le da a la solución entre 1 y 300, la segunda columna corresponde al diámetro del alambre asignado, la tercera columna representa el diámetro medio del resorte, la cuarta columna representa la cantidad de bobinas activas y la quinta columna representa la capacidad asignada a la solución según los datos considerados en las columnas anteriores. Las soluciones con una capacidad menor a -1000000 (por ejemplo -1000001) son soluciones infactibles que violan por lo menos una de las restricciones de diseño, por cada restricción no cumplida se le suma -1000000 a su capacidad (-5000000.233 representaría una solución que viola cinco restricciones de diseño). Las soluciones que tienen una capacidad mayor a -1000000 (por ejemplo -0.5716592800219895) son soluciones factibles.

Con la segunda pestaña (véase la Figura III-12) podemos analizar cualquiera de las 300 soluciones finales sin importar que sean factibles o infactibles y se puede obtener un análisis detallado de la solución requerida.

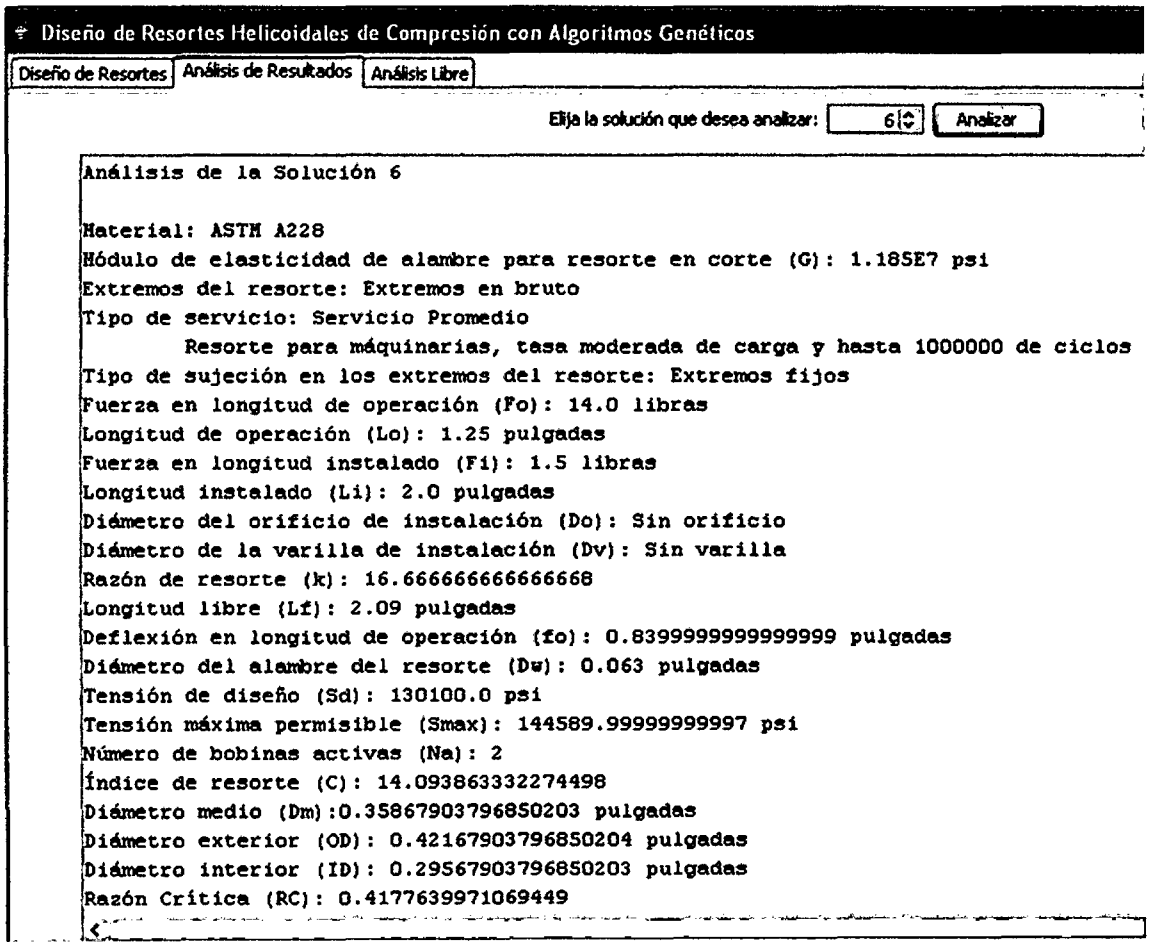


Figura III-12. Pantalla de análisis de una solución.

La tercera pestaña permite hacer un análisis libre con los datos que se consideren apropiados, independientemente de los resultados de la primera o segunda pestaña (véase la Figura III-13).

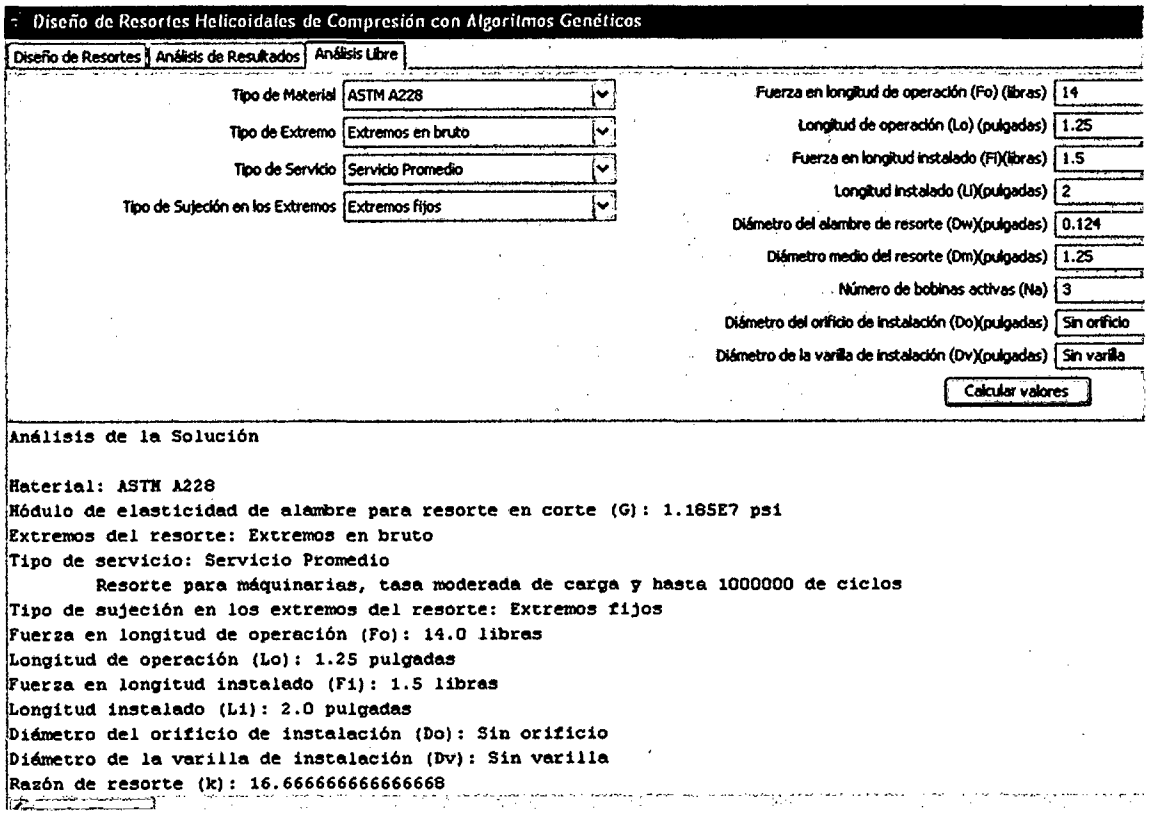


Figura III-13. Pantalla de análisis libre.

Al igual que en la segunda pestaña se obtiene un análisis detallado al conjunto de valores propuesto.

CAPÍTULO IV

PRESENTACIÓN Y ANÁLISIS DE RESULTADOS

Para la contrastar las hipótesis se utilizó un diseño cuasi experimental con tratamientos múltiples y un solo grupo. El grupo seleccionado es un conjunto de diez problemas de diseño de resortes Helicoidales que utilizará el método tradicional de solución con computadora (1er tratamiento) para estos problemas, luego, el mismo grupo utilizará como método de solución el algoritmo de optimización basado en algoritmos genéticos (2do tratamiento). Por lo que tenemos:

G único X1 O1 X2 O2

La hipótesis general se comprueba por la diferencia en los resultados al aplicar los dos tratamientos.

Las hipótesis específicas se prueban por observación directa luego de aplicado el algoritmo de optimización.

G único X2 O2

4.1. EJECUCIÓN DEL CUASI EXPERIMENTO

Se seleccionó como grupo único de prueba a 10 casos de problemas característicos de diseño de resortes helicoidales de compresión, a los que se someterá a dos tratamientos que es llevar los datos de los problemas a dos programas de computadora.

El primer programa (primer tratamiento) corresponde a uno de los propuestos en el libro de Mott[40] para el diseño de resortes helicoidales de compresión, Mott proporciona dos programas a los que los llama método 1 y 2 respectivamente. Se aplicará el método 1 porque es el que ofrece la posibilidad de trabajar con diferentes materiales a diferencia del método 2 que está adaptado para resolver problemas que usen como material el ASTM A231. El código en el lenguaje de programación Java de los dos programas de los métodos 1 y 2 se muestran en el Anexo F y G respectivamente, aunque originalmente estos se presentan por Mott en el lenguaje BASIC.

El segundo programa (segundo tratamiento) es el que usa el algoritmo de optimización para el diseño de resortes helicoidales de compresión y que se presenta como propuesta en esta tesis.

Para cada caso el programa para el método 1 se ejecuta con 30 iteraciones para tratar de obtener el menor volumen posible. En el Anexo H se muestra un ejemplo de cómo se ejecuta el programa con el método 1. Para validar el resultado del primer tratamiento (programa con el método 1) se usó la opción de análisis libre del programa que usa el algoritmo de optimización para el diseño de resortes helicoidales de compresión (del programa que proponemos) porque el método 1 no controla otras restricciones de diseño que son necesarias al diseñar un resorte helicoidal de compresión.

Para la ejecución del cuasi experimento se escogieron 10 casos de diseño de resortes helicoidales de compresión estándar.

4.1.1. CASO 1

Un resorte helicoidal de compresión debe ejercer una fuerza de 8.0 lb cuando se comprime a una longitud de 1.75". A una longitud de 1.25", la fuerza debe ser de 12.0 lb. El resorte se instalará en una máquina que cumple ciclos con lentitud, y aproximadamente se esperan 200,00 ciclos en total. La

---

[40] Mott, R. L. Diseño de Elementos de Máquinas.Pag. 227 - 242. Prentice Hall, México, 1995.

temperatura no excederá los 200° F. Se contempla instalar el resorte en un orificio cuyo diámetro es de 0.75". Especifique un material adecuado para esta aplicación, diámetro del alambre, diámetro medio, OD, ID, longitud libre, longitud comprimido, número de bobinas y tipo de condición en los extremos. Verifique la tensión en la carga máxima de operación y la condición de longitud comprimido.

4.1.2. CASO 2

Diseñe un resorte de compresión para que ejerza una fuerza de 22.0 lb cuando se le comprima hasta que alcance una longitud de 1.75". Cuando su longitud es 3.00" debe ejercer una fuerza de 5.0 lb. El resorte completará ciclos con rapidez y se requiere servicio severo. Utilice alambre de acero ASTM A401.

4.1.3. CASO 3

Diseñe un resorte helicoidal de compresión para una válvula de alivio de presión. Cuando la válvula está cerrada, la longitud de resorte es de 2.00" y la fuerza del resorte debe ser 1.50 lb. A medida que se incrementa la presión de la válvula, una fuerza de 14.0 lb hace que se abra la válvula y se comprima el resorte hasta una longitud de 1.25". Utilice alambre de acero resistente a la corrosión ASTM A313 tipo 302.

4.1.4. CASO 4

Diseñe un resorte helicoidal de compresión que se utilizará para regresar un cilindro neumático a su posición original después que ha sido accionado. A una longitud de 10.50", el resorte debe ejercer una fuerza de 60 lb. A una longitud de 4.00" debe ejercer una fuerza de 250 lb. Se espera servicio severo. Utilice alambre de acero ASTM A231.

4.1.5. CASO 5

Diseñe un resorte helicoidal de compresión, utilizando alambre para instrumentos musicales, que ejercerá una fuerza de 14.00 lb cuando su longitud sea 0.68". La longitud libre debe ser 1.75". Utilice servicio promedio.

4.1.6. CASO 6

Diseñe un resorte helicoidal de compresión utilizando alambre de acero inoxidable, ASTM A313, tipo 316, para servicio promedio, el cual ejercerá una fuerza de 8.00 lb después de deflexionarse 1.75", a partir de una longitud libre de 2.75".

4.1.7. CASO 7

Diseñe un resorte helicoidal de compresión utilizando alambre de acero inoxidable, ASTM A313, tipo 316, para servicio promedio, el cual ejercerá una fuerza de 8.00 lb después de deflexionarse 1.75", a partir de una longitud libre de 2.75". El resorte debe operar alrededor de una varilla cuyo diámetro es 0.625"

Al deflexionarse 1.75" llegará a una altura de 2.75" – 1.75" = 1" en longitud de operación.

4.1.8. CASO 8

Diseñe un resorte helicoidal de compresión, utilizando alambre para instrumentos musicales, que ejercerá una fuerza de 14.00 lb cuando su longitud sea 0.68". La longitud libre debe ser 1.75". Utilice servicio promedio. El resorte se instalará dentro de un orificio que tiene 0.750" de diámetro.

4.1.9. CASO 9

Diseñe un resorte helicoidal de compresión utilizando alambre de acero ASTM A231 para servicio severo, que ejercerá una fuerza de 45.00 lb a una longitud de 3.05" y una fuerza de 22.00 lb a una longitud de 3.50".

4.1.10. CASO 10

Diseñe un resorte helicoidal de compresión utilizando alambre redondo de acero, ASTM 227. El resorte activará un embrague y debe soportar múltiples ciclos de operación. Cuando los discos de embrague estén en contacto, el resorte tendrá una longitud de 2.50" y debe ejercer una fuerza de 20 lb. Cuando el embrague no esté activado, la longitud del resorte será de 2.10" y debe ejercer una fuerza de 35 lb. El resorte se instalará alrededor de una flecha redonda cuyo diámetro es 1.50". El diámetro exterior no debe ser mayor a 2. 50".

Los detalles de la solución para cada caso con el método 1 y con el algoritmo de optimización basado en algoritmos genéticos se presentan en el Anexo I y todas las soluciones logradas son factibles (cumplen con todas las restricciones de diseño). La presentación de estas soluciones cuenta con un análisis completo acerca de cada propuesta de diseño.

4.2. CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 1

4.2.1. VARIABLES E INDICADORES DE LA HIPÓTESIS ESPECÍFICA 1

Hipótesis: La funcionalidad de los cromosomas en el componente AGJava permite representar las variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

Variable independiente: Funcionalidad de los cromosomas del componente AGJava en el algoritmo genético para la solución óptima de problemas de diseño de resortes helicoidales de compresión.	Variable dependiente: Representación de las variables en el Diseño de Resortes Helicoidales de Compresión.

Indicador: Número de problemas de diseño de resortes helicoidales de compresión estándar. (Se necesita el conjunto de datos de entrada para el problema: tipo de material, tipo de extremo, tipo de servicio, tipo de sujeción, longitud de operación, fuerza en longitud de operación, longitud de instalado, fuerza en longitud de instalado).	Indicador: Número de ejecuciones del programa de diseño de resortes helicoidales sin presentar errores debidos a la representación de las variables.

4.2.2. PRUEBA DE HIPÓTESIS DE LA HIPÓTESIS ESPECÍFICA 1

Se dice que el algoritmo permite representar estas variables, si en la ejecución de un programa que use este algoritmo no se producen errores como consecuencia del uso de las variables representadas por los cromosomas con el componente AGJava. El diseño que se deba realizar debe ser lógico y coherente (por ejemplo el diámetro del orificio no debe ser menor al diámetro de la varilla).

La siguiente hipótesis es equivalente a la hipótesis específica 1:

Ho = En el 100% de las ejecuciones del programa que utilice el algoritmo de optimización usando el componente AGJava para representar los cromosomas no se producirán errores como consecuencia del uso de las variables de diseño para resortes helicoidales de compresión a un nivel de significación del 0.005.

H_0: p = 1.00

H_1: p < 100

Resultado del experimento: En la muestra de 10 problemas de diseño de resortes helicoidales de compresión escogidos y aplicados al programa con el algoritmo genético de optimización no se produjeron errores como conse-

cuencia del uso de las variables de diseño para resortes helicoidales de compresión representadas con el componente AGJava.

El nivel de significación escogido es del 0.005 ($\alpha = 0.005$)

La estadística de prueba es la binomial, porque tenemos una muestra pequeña de tamaño 10.

La Región crítica es: $P[X < x \mid H_0 \text{ es verdadera}] < \alpha$

$P[X < 10 \mid H0 \text{ es verdadera}] < 0.005$

Con x=10, n=10

$P[X < 10 \mid 10, 1.0] = 1 - P[X \geq 10 \mid 10, 1] = 1 - 0$

$1 < 0.005$ Falso $\therefore$ No se puede rechazar H_0.

4.2.3. RESULTADO DE LA CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 1

Aceptamos Ho, es decir no hay razón para dudar que en el 100% de las ejecuciones del programa que use el algoritmo genético con el componente AGJAva para representar los cromosomas no se produzcan errores como consecuencia del uso de las variables de diseño para resortes helicoidales de compresión. Por lo tanto, la funcionalidad de los cromosomas en el componente AGJava permite representar las variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

4.3. CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 2

4.3.1. VARIABLES E INDICADORES DE LA HIPÓTESIS ESPECÍFICA 2

Hipótesis: La funcionalidad de la población del componente AGJava permite trabajar con más de una solución en una generación en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

Variable independiente: Funcionalidad de la población del componente AGJava en el algoritmo genético para la solución óptima de problemas de diseño de resortes helicoidales de compresión.	Variable dependiente: Soluciones con las que se trabaja en una iteración del algoritmo genético.
Indicador: Número de problemas de diseño de resortes helicoidales de compresión estándar. (Se necesita el conjunto de datos de entrada para el problema: tipo de material, tipo de extremo, tipo de servicio, tipo de sujeción, longitud de operación, fuerza en longitud de operación, longitud de instalado, fuerza en longitud de instalado).	Indicador: Número de ejecuciones del algoritmo genético de diseño de resortes helicoidales sin presentar errores debidos al uso de más de una solución en una iteración del algoritmo genético.

4.3.2. PRUEBA DE HIPÓTESIS DE LA HIPÓTESIS ESPECÍFICA 2

Se dice que el algoritmo permite trabajar con más de una solución en una iteración, si en la ejecución del algoritmo genético que usa el componente AGJava para representar la población de soluciones no se producen errores como consecuencia del uso de más de una solución en una iteración.

La siguiente hipótesis es equivalente a la hipótesis específica 1:

Ho = En el 100% de las ejecuciones del programa que utilice el algoritmo genético de optimización que usa el componente AGJAva para representar la población no se producirán errores como consecuencia del uso de más de una solución en la población en una iteración del algoritmo a un nivel de significación del 0.005.

$$H_0: p = 1.00$$

H_1: p < 100

Resultado del experimento: En la muestra de 10 problemas de diseño de resortes helicoidales de compresión escogidos y aplicados al programa con el algoritmo de optimización no se produjeron errores como consecuencia del uso de más de una solución al diseño en una iteración del algoritmo a través del componente AGJava.

El nivel de significación escogido es del 0.005 (α = 0.005)

La estadística de prueba es la binomial, porque tenemos una muestra pequeña de tamaño 10.

La Región crítica es: P[X < x | H0 es verdadera] < α

P[X < 10 | H0 es verdadera] < 0.005

x=10, n=10

P[X < 10 | 10, 1.0] = 1 - P[X $\geq$ 10 | 10 , 1] = 1 – 0

1 < 0.005 Falso ∴No se puede rechazar Ho.

4.3.3. RESULTADO DE LA CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 2

Aceptamos Ho, es decir no hay razón para dudar que en el 100% de las ejecuciones del programa que use el algoritmo genético con el componente AGJava para representar al población no se produzcan errores como consecuencia del uso de más de una solución al diseño en una iteración del algoritmo a un nivel de significación del 0.005. Entonces podemos afirmar que la funcionalidad de la población del componente AGJava permite trabajar con más de una solución en una generación en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

4.4. CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 3

4.4.1. VARIABLES E INDICADORES DE LA HIPÓTESIS ESPECÍFICA 3

Hipótesis: La funcionalidad de los operadores genéticos del componente AGJava permite crear nuevas soluciones en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

Variable independiente: Funcionalidad de la operadores genéticos del componente AGJava en el algoritmo genético para la solución óptima de problemas de diseño de resortes helicoidales de compresión.	Variable dependiente: Nuevas soluciones con las que se trabaja en una iteración del algoritmo genético.
Indicador: Número de problemas de diseño de resortes helicoidales de compresión estándar. (Se necesita el conjunto de datos de entrada para el problema: tipo de material, tipo de extremo, tipo de servicio, tipo de sujeción, longitud de operación, fuerza en longitud de operación, longitud de instalado, fuerza en longitud de instalado).	Indicador: Número de ejecuciones del programa de diseño de resortes helicoidales sin presentar errores debidos al uso de operadores genéticos para crear nuevas soluciones en la optimización del Diseño de Resortes Helicoidales de Compresión.

4.4.2. PRUEBA DE HIPÓTESIS DE LA HIPÓTESIS ESPECÍFICA 3

Se dice que el algoritmo puede crear nuevas soluciones factibles o infactibles con los operadores genéticos del componente AGJava si al utilizar estos operadores en el algoritmo genético para el problema de optimización en el Diseño de Resortes Helicoidales de Compresión no se producen errores como consecuencia del uso de estos operadores genéticos.

La siguiente hipótesis es equivalente a la hipótesis específica 3:

Ho = En el 100% de las ejecuciones del programa que utilice el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión no se producirán errores como consecuencia del uso de operadores genéticos del componente AGJava a un nivel de significación del 0.005.

H_0: p = 1.00

H_1: p < 100

Resultado del experimento: En la muestra de 10 problemas de diseño de resortes helicoidales de compresión escogidos y aplicados al programa con el algoritmo genético de optimización no se produjeron errores como consecuencia del uso de operadores genéticos del componente AGJava para crear nuevas soluciones.

El nivel de significación escogido es del 0.005 (α = 0.005)

La estadística de prueba es la binomial, porque tenemos una muestra pequeña de tamaño 10.

La Región crítica es: P[X < x | H0 es verdadera] < α

P[X < 10 | H0 es verdadera] < 0.005

x=10, n=10

P[X < 10 | 10, 1.0] = 1 - P[X $\geq$ 10 | 10 , 1] = 1 – 0

1 < 0.005 Falso ∴No se puede rechazar Ho.

4.4.3. RESULTADO DE LA CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 3

Aceptamos Ho, es decir no hay razón para dudar que en el 100% de las ejecuciones del programa que use el algoritmo genético no se produzcan errores como consecuencia del uso de operadores genéticos del componente AGJava. Entonces la funcionalidad de los operadores genéticos del componente AGJava permiten crear nuevas soluciones en la solución óptima con

algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

4.5. CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 4

4.5.1. VARIABLES E INDICADORES DE LA HIPÓTESIS ESPECÍFICA 4

Hipótesis: La funcionalidad de las vistas del componente AGJava permite representar diferentes tipos de variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

Variable independiente: Funcionalidad de la vistas del componente AGJava en el algoritmo genético para la solución óptima de problemas de diseño de resortes helicoidales de compresión.	Variable dependiente: Soluciones con las que se trabaja en una iteración del algoritmo genético.
Indicador: Número de problemas de diseño de resortes helicoidales de compresión estándar. (Se necesita el conjunto de datos de entrada para el problema: tipo de material, tipo de extremo, tipo de servicio, tipo de sujeción, longitud de operación, fuerza en longitud de operación, longitud de instalado, fuerza en longitud de instalado).	Indicador: Número de ejecuciones del programa de diseño de resortes helicoidales sin presentar errores debido a la representación de diferentes tipos de variables con las vistas del componente AGJava en el algoritmo genético de optimización para el Diseño de Resortes Helicoidales de Compresión.

4.5.2. PRUEBA DE HIPÓTESIS DE LA HIPÓTESIS ESPECÍFICA 4

Se dice que el algoritmo puede representar diferentes tipos de variables con las vistas del componente AGJava en el algoritmo genético para la optimización en el Diseño de Resortes Helicoidales de Compresión, si en la ejecución de un programa que use estas vistas del componente AGJava no se

producen errores como consecuencia del uso de esta representación de optimización.

La siguiente hipótesis es equivalente a la hipótesis específica 3:

Ho = En el 100% de las ejecuciones del programa del algoritmo genético de optimización para el diseño de resortes helicoidales de compresión, no se producirán errores como consecuencia del uso de la representación de diferentes tipos de variables con el componente AGJava a un nivel de significación del 0.005.

H_0: p = 1.00

H_1: p < 100

Resultado del experimento: En la muestra de 10 problemas de diseño de resortes helicoidales de compresión escogidos y aplicados al programa con el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión no se produjeron errores como consecuencia de la representación de diferentes tipos de variables con el componente AGJava.

El nivel de significación escogido es del 0.005 (α = 0.005)

La estadística de prueba es la binomial, porque tenemos una muestra pequeña de tamaño 10.

La Región crítica es: P[X < x | H0 es verdadera] < α

P[X < 10 | H0 es verdadera] < 0.005

x=10, n=10

P[X < 10 | 10, 1.0] = 1 - P[X $\geq$ 10 | 10 , 1] = 1 – 0

1 < 0.005 Falso ∴No se puede rechazar Ho.

4.5.3. RESULTADO DE LA CONTRASTACIÓN DE LA HIPÓTESIS ESPECÍFICA 4

Aceptamos Ho, es decir no hay razón para dudar que en el 100% de las ejecuciones del algoritmo genético para la optimización en el diseño de resortes

helicoidales de compresión, no se produzcan errores como consecuencia del uso de las vistas del componente AGJava para representar diferentes tipos de variables en la solución. Entonces la funcionalidad de las vistas del componente AGJava permite representar diferentes tipos de variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.

4.6. CONTRASTACIÓN DE LA HIPÓTESIS GENERAL

4.6.1. VARIABLES E INDICADORES DE LA HIPÓTESIS GENERAL

Hipótesis: El Algoritmo Genético basado en el componente AGJava mejora la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión.

Variable independiente: Algoritmo Genético basado en el componente AGJava para la optimización en el Diseño de Resortes Helicoidales de Compresión.	Variable dependiente: Solución óptima en el Diseño de Resortes Helicoidales de Compresión.
Indicador: Número de problemas de diseño de resortes helicoidales de compresión estándar. (Se necesita el conjunto de datos de entrada para el problema: tipo de material, tipo de extremo, tipo de servicio, tipo de sujeción, longitud de operación, fuerza en longitud de operación, longitud de instalado, fuerza en longitud de instalado).	Indicador: Capacidad asignada al resorte (Si se cumple con todas las restricciones es igual al volumen del resorte).

4.6.2. PRUEBA DE HIPÓTESIS DE LA HIPÓTESIS GENERAL

Se considera óptimo el Diseño de Resortes Helicoidales de Compresión cuando este tiene la menor cantidad posible de un determinado material, así

el volumen del material del resorte determina su costo. Consideramos óptimo un diseño cuando este tiene el menor volumen posible. El espacio de soluciones factibles es prácticamente ilimitado, en sí el óptimo que se encuentra en el espacio de soluciones factibles es una solución heurística que da solución al problema de diseño de resortes helicoidales de compresión, a diferencia de los métodos de optimización tradicional donde no se pueden resolver este tipo de problemas.

La comprobación de si la solución óptima mejora, requeriría de comparar el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión basado en el componente AGES con otro que sea de optimización sin embargo los métodos de optimización tradicionales no se pueden aplicar a este tipo de problemas por el espacio de soluciones ilimitado. Por lo tanto comparamos el resultado del método 1 (tradicional) luego de 30 iteraciones con el resultado del programa que utiliza el algoritmo de optimización basado en algoritmos genéticos, el resultado del método 1 por suerte podría ser el óptimo pero no generalmente, por lo general el resultado del algoritmo de optimización basado en algoritmos genéticos debe ser mejor.

La prueba de diferencia pareada se hace para determinar si la diferencia entre los volúmenes resultantes con el método 1 (método tradicional) y el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión basado en el componente AGJava es diferente. En el caso de la aplicación del método 1 al problema de diseño se consideró parar de calcular luego de 30 iteraciones. La hipótesis que se plantea es:

Hipótesis: Los datos proporcionan suficiente evidencia para indicar que el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión ha sido efectivo al reducir el volumen del resorte en comparación al volumen obtenido por el método 1 a un nivel de significación de 0.15.

El nivel de significación que se escogió es porque existe la posibilidad de que al hacer el diseño con el método 1 por suerte se puede llegar a un valor

cercano al óptimo y además porque los valores de los volúmenes de los resortes para estos problemas de diseños son muy pequeños.

Ho: $\mu_D = 0$ No hay diferencia alguna entre las dos muestras relacionadas y

H_1: $\mu_D > 0$ Hay una diferencia mayor a cero

Los resultados del experimento se muestran en la Tabla VI-1.

Tabla IV-1. Tabla de resultados del experimento para diferencias.

	Volumen (pulgadas cúbicas)		Diferencia (Di)	Di^2
	Método 1	Método de optimización		
Caso 1	0.046047626	0.036192382	0.009855244	0.001309889
Caso 2	1.296927703	1.294049727	0.002877977	1.674564695
Caso 3	0.08993677	0.076108566	0.013828204	0.005792514
Caso 4	0.17987354	0.151211716	0.028661825	0.022864983
Caso 5	0.041585578	0.033004666	0.008580912	0.001089308
Caso 6	21.62647456	21.60888169	0.017592866	466.9437679
Caso 7	0.727666193	0.700896964	0.026769229	0.491256554
Caso 8	0.109014913	0.100434001	0.008580912	0.010086989
Caso 9	1.172305492	1.150335259	0.021970233	1.323271207
Caso 10	2.464162636	2.02822549	0.435937146	4.113698638
			27.17934046	474.5877027

Fuente: Elaboración propia.

El nivel de significación es de 0.15 ($\alpha = 0.15$)

La estadística de prueba es: $T = \frac{\bar{D}\sqrt{n}}{S_D}$ que tiene distribución t con n - 1 = 9

grados de libertad.

La región de rechazo es $t > t_\alpha$, donde t_α es tal que:

$P[T > t_\alpha] = 0.15$, de la tabla $t_\alpha = 1.1$

Luego R.C. $t > 1.1$

En nuestro caso:

$$\bar{D} = \frac{\sum_{i=1}^{n} D_i}{n} = \frac{27.17934046}{10} = 2.717934046 \qquad (6\text{-}1)$$

y

$$S_D = \sqrt{\frac{n\sum_{i=1}^{n} D_i^2 - \left(\sum_{i=1}^{6} D_i\right)^2}{n(n-1)}} = \sqrt{\frac{10(474.59) - 21.18^2}{10(9)}} = 6.91 \qquad (6\text{-}2)$$

Así tenemos:

$$t = \frac{2.72\sqrt{10}}{6.91} = 1.25 \qquad (6\text{-}3)$$

Observamos que: 1.25 > 1.1, se rechaza Ho, es decir si hay evidencia de la diferencia entre el resultado de los dos métodos.

Una prueba adicional que se hace es determinar si en proporción los volúmenes obtenidos por el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión basado en el componente AGJava son pequeños en comparación con el resultado obtenido con el método 1. Así planteamos que en el 80% de los casos el volumen que se obtiene por el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión basado en el componente AGJava es por lo menos el 20% del volumen que se obtiene con el método 1 (método tradicional). La Hipótesis planteada es:

Hipótesis: En el 80% de los casos el volumen que se obtiene por el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión basado en el componente AGJava es como máximo el 20% del volumen que se obtiene con el método 1 a un nivel de significación de 0.005

Ho: p = 0.80

H1: p < 0.80

Los resultados del experimento son:

Tabla IV-2. Tabla de resultados del experimento para proporciones.

	Volumen (pulgadas cúbicas)		Proporción de resultado del método de optimización con referencia al método 1
	Método 1	Método de optimización	
Caso 1	0.046047626	0.009855244	0.214022846
Caso 2	1.296927703	0.002877977	0.002219073
Caso 3	0.08993677	0.013828204	0.153754732
Caso 4	0.17987354	0.028661825	0.159344307
Caso 5	0.041585578	0.008580912	0.206343456
Caso 6	21.62647456	0.017592866	0.000813487
Caso 7	0.727666193	0.026769229	0.036787787
Caso 8	0.109014913	0.008580912	0.078713193
Caso 9	1.172305492	0.021970233	0.018741047
Caso 10	2.464162636	0.435937146	0.176910866

Se pude observar que son 8 los casos en los que la proporción es como máximo 0.20.

El nivel de significación escogido es de 0.005 ($\alpha = 0.005$)

La estadística de prueba es la binomial, porque tenemos una muestra pequeña de tamaño 10.

La Región crítica es: $P[X < x \mid H0 \text{ es verdadera}] < \alpha$

$P[X < 8 \mid H0 \text{ es verdadera}] < 0.005$

$x = 8, n=10$

$P[X < 8 \mid 10, 0.8] = 1 - P[X \geq 8 \mid 10, 0.8] = 1 - 0.6778 = 0.3222$

$0.3222 < 0.005$ Falso $\therefore$ No se puede rechazar Ho.

4.6.3. RESULTADO DE LA CONTRASTACIÓN DE LA HIPÓTESIS GENERAL

Los datos proporcionan suficiente evidencia para indicar que el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión basado en el componente AGJava ha sido efectivo al reducir el volumen del resorte en comparación al volumen obtenido por el método 1 a un nivel de significación de 0.15.

No hay razón para dudar que en el 80% de los casos el volumen que se obtiene por el algoritmo genético de optimización para el diseño de resortes helicoidales de compresión basado en el componente AGJava sea como máximo el 20% del volumen que se obtiene con el método 1 a un nivel de significación de 0.005.

Entonces podemos afirmar que el Algoritmo Genético basado en el componente AGJava mejora la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES:

1. La funcionalidad de los cromosomas en el componente AGJava permite representar las variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.
2. La funcionalidad de la población del componente AGJava permitió trabajar con más de una solución en una generación en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.
3. La funcionalidad de los operadores genéticos del componente AGJava permitió crear nuevas soluciones en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.
4. La funcionalidad de las vistas del componente AGJava permitió representar diferentes tipos de variables en la solución óptima con algoritmos genéticos en problemas de Diseño de Resortes Helicoidales de Compresión.
5. El Algoritmo Genético basado en el componente AGJava mejora la solución óptima de problemas de Diseño de Resortes Helicoidales de Compresión en comparación con el método tradicional de diseño..
6. Los problemas de diseño de elementos mecánicos son por lo general complejos con un espacio de soluciones infinito y un óptimo difícil o im-

posible de encontrar, los algoritmos genéticos representan una buena opción heurística para la solución de estos problemas.

7. La aplicación que se le puede dar a los algoritmos genéticos es vasta, el componente AGJava es un componente reutilizable y extensible que se puede aplicar a algoritmos genéticos para dar solución a problemas en diversos campos de aplicación donde el espacio de soluciones sea ilimitado o demasiado grande.
8. Las limitaciones del componente AGJava se pueden eliminar añadiendo al programa que use el componente, el código necesario que permita modelar el comportamiento que se le quiera dar o también se puede modificar o extender el componente AGJava, pues se ofrece como parte de esta tesis el código completo.

Recomendaciones:

1. Se recomienda considerar el uso del componente AGJava para desarrollar algoritmos genéticos para otros tipos de problemas donde el espacio de soluciones sea grande o ilimitado.
2. Se sugiere usar la documentación navegable del componente AGJava para entender las interfaces del componente y desarrollar otros algoritmos genéticos.
3. Se propone utilizar el resto de operadores del componente AGJava para analizar el efecto de estos en el proceso de optimización.
4. Se exhorta a modificar el tamaño de la población y el número de generaciones para analizar el efecto de estos en el proceso de optimización utilizando el componente AGJava.
5. Se recomienda mejorar la funcionalidad de las vistas del componente AGJava para obtener una mayor variedad en lo que se refiere a la interpretación de los cromosomas obtenidos producto de la evolución.
6. Se sugiere investigar nuevas estructuras para el algoritmo genético de optimización para mejorar los resultados que se obtuvieron en esta tesis.

Es decir analizar el efecto de los pesos de los operadores genéticos y del orden en el que se aplican, del porcentaje de elitismo y tipo de selección aplicada.

7. Se insta a usar el componente AGJava para entrenar redes neuronales, para mejorar el resultado de la aplicación de las redes neuronales.
8. Se propone a usar el componente AGJava para crear algoritmos genéticos de optimización asociados a sistemas expertos, pues la propuesta de solución inicial que pueda dar un sistema experto puede ser un buen punto de partida para la evolución.

GLOSARIO DE TÉRMINOS

Algoritmo Genético.- Programa elaborado donde se genera un ciclo iterativo que directamente toma a las soluciones y crea una nueva generación de soluciones que reemplaza a la antigua una cantidad de veces determinada por su propio diseño. Una de sus características principales es la de ir perfeccionando su propia heurística en el proceso de ejecución, por lo que no requiere largos períodos de entrenamiento especializado por parte del ser humano.

Archivo jar.- Los ficheros Jar (Java ARchives) permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio. La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados, es decir que el intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.

Capacidad.- Llamada también función de aptitud no es más que la función objetivo de nuestro problema de optimización. Una característica que debe tener esta función es que tiene ser capaz de "castigar" a las malas soluciones, y de "premiar" a las buenas, de forma que sean estas últimas las que se propaguen con mayor rapidez.

Componente AGJava.- Componente java que ofrece la funcionalidad para crear cromosomas, población, operadores genéticos y vistas para la creación de algoritmos genéticos.

Componente.- Un componente es un objeto de software específicamente diseñado para cumplir con cierto propósito. Los principios fundamentales cuando se diseña un componente es que estos deben ser: reusable, sin contexto especifico, extensible, encapsulado e independiente.

Cruce.- Se denomina operador de cruce a la forma de calcular el genoma del nuevo individuo en función del genoma del padre y de la madre. El operador de cruce es fuertemente responsable de las propiedades del algoritmo genético, y determinará en gran medida la evolución de la población.

Mutación.- Se define mutación como una variación de las informaciones contenidas en el cromosoma del algoritmo genético, un cambio de un gen a otro producido por algún factor exterior al algoritmo genético

Operador Genético.- Un operador genético es una función empleada en los algoritmos genéticos para mantener la diversidad genética de una población, básicamente de cruce y mutación.

Optimización.- El proceso de encontrar los mínimos y máximos de una función, en algortimos genéticos se habla de optimización heurística, esto es encontrar un valor cercano al mínimo o máximo.

Paquete.- Un Paquete en Java es un contenedor de clases que permite agrupar las distintas partes de un programa cuya funcionalidad tienen elementos comunes

Resorte Helicoidal de Compresión.- Se los reconoce porque tienen sus espiras separadas. Trabajan con cargas aplicadas que comprimen al resorte

RUP.- El Proceso Racional Unificado (Rational Unified Process en inglés, habitualmente resumido como RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP no es un sis-

tema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Selección.- Es la parte del algoritmo que se encarga de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no. Puesto que se trata de imitar lo que ocurre en la naturaleza, se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto la selección de un individuo estará relacionada con su valor de capacidad. No se debe sin embargo eliminar por completo las opciones de reproducción de los individuos menos capaces, pues en pocas generaciones la población se volvería homogénea

Tasa de elitismo.- Es el porcentaje de la población que no cambiará. En esta estrategia se busca el mejor individuo de la población anterior e independientemente de la selección y variación, se lo copia exactamente en la nueva población. De esta manera nos aseguramos de no perder la mejor solución generación tras generación.

BIBLIOGRAFÍA

1. Ceballos Sierra Fco. Java 2 Curso de Programación. México: AlfaOmega; 4ta. Edición, 2011.
2. Chapra, Steven. Métodos Numéricos para Ingenieros. México: McGRAW-HILL; 5ta. Edición, 2007.
3. Dasgupta D., Michalewicz Z. Evolutionary Algorithms in Engineering Applications. U.S.A.: Springer; 1997.
4. Deitel, H. M. JAVA How To Program. U.S.A.: Prentice Hall; 8va. Edición 2009.
5. Eliason S. Maximum Likelihood Estimation, Logic and Practice. Londres: SAGE Publications, 1993.
6. Gere y Timoshenko. Mecánica de Materiales. México: Thomson Editores; 6ta. Edición 2005.
7. Hamrock, B. J., Jacobson, B. y Schmid S.R. Elementos de Máquina. México: McGraw-Hill; 2000.
8. Johnson R. Probabilidad y Estadística Para Ingenieros. México: Pearson/Prentice Hall, 8va. Edición 2011.
9. Mendenhall W. y Sincich T. Probabilidad y Estadística Para Ingeniería y Ciencias. México: Mc Graw Hill; 1997.
10. Michalewics, Zbigniew. Genetics Algorithms + Data Structures = Evolution Programs. U.S.A.: Springer; 1998.

11. Mott, R. L. Diseño de Elementos de Máquinas. México: Prentice Hall; 4ta. Edición 1995.

12. Moya R. y Saravia G. Probabilidad e Inferencia Estadística. Perú: Editorial San Marcos; 3ra. reimp. 2009.

13. Norton R. L. Diseño de Máquinas. México: Mc Graw Hill; 3ra. Edición 2004.

14. Spotts M. F. y Shoup, T.E. Elementos de Máquinas. México: Prentice Hall; 2003.

15. Taha H. Investigación de Operaciones. México: Pearson; 2004.

Tesis

1. Amaya M. C., Tesis para optar el Grado de Ingeniería de Sistemas e Informática. Un algoritmo genético para optimizar la asignación de aulas y docentes en la generación de programas académicos en centros de idiomas. UNMSM, 2009.

2. Bejarano G. M., Tesis para optar por el Título de Ingeniero Informático. Planificación de Horarios del Personal de Cirugía de un Hospital del Estado Aplicando Algoritmos Genéticos. PUCP, 2009

3. Rivera J. E., Tesis para optar por el Título de Ingeniero Informático. Análisis Comparativo entre el Algoritmo Cuántico de Grover y un Algoritmo Grasp, Aplicados a la Búsqueda de Individuos Óptimos en la Población Inicial de un Algoritmo Genético. PUCP, 2011.

4. Rodriguez G., Tesis de Maestría en Ciencias con mención en Ingeniería de Sistemas. Optimización de celosias bidimensionales mediante algoritmos genéticos. UNI, 2011

ANEXOS

A. CODIFICACIÓN DEL PAQUETE UTIL DEL COMPONENTE AGJAVA

```
//////////////////////////////////////////////////////////////////////////
// AleatorioInt

package agjava.util;

/**
 * Una clase utilitaria para proveer un generador de numeros aleatorios.
 *
 */
public class AleatorioInt
{
   /**
    * Devuelve un entero aleatorio uniformemente seleccionado del rango
    * entre 0 a tamRango-1.
    *
    * @param tamRango el número de posibles respuestas necesitadas
    * @return un entero aleatorio en el rango requerido
    */
   public static int proximo( int tamRango )
   {
      int aleatorio = (int) (Math.random() * tamRango);

      // Afronta la oportunidad de 1 en 2^64 que random devuelva 1.0

      if (aleatorio == tamRango)
         --aleatorio;

      return aleatorio;
   }
}

//////////////////////////////////////////////////////////////////////////
// BitSetExtendido

package agjava.util;

import java.io.Serializable;
import java.util.Random;
```

```
/**
 * Clase de cadena de bits que implementa un conjunto más flexible de
 * operadores que java.util.BitSet, si bien este implementa todas las
 * rutinas de esa clase para que se haga compatible al usarlo directamente.
 * Este también emula la funcionalidad de BitSet de extender
 * transparentemente el conjunto con bits cero si se accesa a un bit por
 * fuera de su longitud actual.
 *
 * @see java.util.BitSet
 */
public class BitSetExtendido implements Cloneable, Serializable
{
   /**
    * Mantiene los bits del conjunto de bits
    */
   protected long [] bits;

   /**
    * Número de bits en el conjunto de bits
    */
   protected int lonBits;

   protected static final int BITSENBYTE = 8;
   protected static final int BITSENCHAR = 16;
   protected static final int BITSENSHORT = 16;
   protected static final int BITSENINT = 32;
   protected static final int BITSENLONG = 64;

   protected static Random aleatorio = new Random();

   /**
    * Constructor que establece longitud cero
    */
   public BitSetExtendido()
   {
      lonBits = 0;
   }

   /**
    * Construye a partir de un patrón de cadena de bits . Cada char deberá
    * ser un 1 o 0 (aunque cualquier no '0' es tratado como '1'). Usando
    * este constructor en el resultado de toString llamado en otro conjunto
    * de bits debería resultar en una copia duplicada.
    *
    * @param patronBits conjunto de bits como cadena de 1s y 0s.
    */
   public BitSetExtendido( String patronBits )
   {
      int índice = 0;

      lonBits = patronBits.length();
      bits = new long[(lonBits-1) / BITSENLONG + 1];

      for (int i = lonBits; i > 0; i -= BITSENLONG)
      {
```

```
            long masc = 1;
            long val = 0;

            for (int j = 0; j < BITSENLONG && lonBits - j > 0; ++j)
            {
               if (i-j-1 >= 0)
               {
                  if (patronBits.charAt( i-j-1 ) != '0')
                  val |= masc;
                  masc <<= 1;
               }
            }

            bits[índice++] = val;
         }
      }

      /**
       * Construye a partir de una matriz de bytes. Los contenidos de los
       * bytes determinan directamente los bits. El bit 0 del elemento 0
       * del array es el bit 0 del conjunto de bits, el bit 7 del elemento
       * n es el bit n*8+63 del conjunto, sujeto a la longitud máxima dada
       * por <code>longitud</code>.
       *
       * @param longitud número de bits en el conjunto.
       * @param patronBits valores de los bits iniciales.
       */
      public BitSetExtendido( int longitud, byte [] patronBits )
      {
         bits = new long[(longitud - 1) / BITSENLONG + 1];
         lonBits = longitud;

         for (int índice = 0; longitud > 0; longitud -= BITSENLONG, ++índice)
         {
            for (int j = 0; j < 8 && (índice * 8 + j) < patronBits.length;
                 ++j)
               bits[índice] |=
                  (patronBits[índice * 8 + j] & 0xFFL) << (8*j);
         }
      }

      /**
       * Construye a partir de un array de longs.El contenido de los longs
       * determinan directamente los bits. El bit 0 del elemento 0 del
       * array es el bit 0 del conjunto de bits, el bit 63 del elemento n
       * del array es el bit n*64+63 del conjunto, sujeto a la máxima
       * longitud dada por <code>longitud</code>.
       *
       * @param longitud número de bits en el conjunto.
       * @param patronBits valores de los bits iniciales.
       */
      public BitSetExtendido( int longitud, long [] patronBits )
      {
         inicializarTodo( longitud, patronBits );
      }
```

```
/**
 * Construye con una longitud dada inicializada a ceros.
 *
 * @param longitud número de bits en el conjunto
 */
public BitSetExtendido( int longitud )
{
   if (longitud < 1)
      longitud = 1;
   lonBits = longitud;
   bits = new long[(lonBits-1) / BITSENLONG + 1];
}

/**
 * Contruye con una longitud dada inicializada con valores aleatorios.
 *
 * @param longitud número de bits en el conjunto.
 * @param semillaAleatoria semilla para el generador; si es cero
 *        el generador no utiliza la misma semilla.
 */
public BitSetExtendido( int longitud, long semillaAleatoria )
{
   this( longitud );

   // Asigna valores aleatorios

   synchronized (aleatorio)
   {
      if (semillaAleatoria != 0)
         aleatorio.setSeed( semillaAleatoria );

      for (int i = 0; i < bits.length; ++i)
         bits[i] = aleatorio.nextLong();
   }

   // Establece a cero los bits no usados arriba del último long.
   // El cambio trabaja puesto que el JLS define que solo los bits
   // de más abajo del resultado se usan para generar un valor
   // entre 0-63.

   bits[bits.length-1] &= -1L >>> (BITSENLONG - longitud);
}

/**
 * Construye una copia de otro conjunto de bits extendidos.
 *
 * @param conjBits the otro bit set
 */
public BitSetExtendido( BitSetExtendido conjBits )
{
   lonBits = conjBits.lonBits;
   if (bits == null || bits.length < conjBits.bits.length)
      bits = new long[conjBits.bits.length];
   System.arraycopy( conjBits.bits, 0, bits, 0, conjBits.bits.length );
}
```

```
/**
 * Copia los contenidos de otro conjunto de bits extendidos.
 *
 * @param conjBits el otro conjunto de bits
 * @return el conjunto de bits actual
 */
public BitSetExtendido copiar( BitSetExtendido conjBits )
{
   if (conjBits != this)
   {
      lonBits = conjBits.lonBits;
      if (bits == null || bits.length < conjBits.bits.length)
         bits = new long[conjBits.bits.length];
      System.arraycopy( conjBits.bits, 0, bits, 0, conjBits.bits.length );
   }

   return this;
}

/**
 * Crea y devuelve una copia de este conjunto de bits.
 *
 * @return una copia de este conjunto de bits
 */
public Object clone()
{
   return new BitSetExtendido( this );
}

/**
 * Devuelve el conjunto de bits representado como una cadena binaria
 *
 * @return la representación de cadena del conjunto
 */
public String toString()
{
   StringBuffer buf = new StringBuffer( lonBits );

   if (bits != null)
   {
      long masc = 1L << (lonBits-1);

      for (int i = bits.length - 1; i >= 0; --i, masc = 1L << 63)
      {
         for ( ; masc != 0; masc >>>= 1)
         {
            if ((bits[i] & masc) != 0)
               buf.append( "1" );
            else
               buf.append( "0" );
         }
      }
   }

   return buf.toString();
}
```

```
/**
 * Compara este objeto contra el objeto especificado.
 *
 * @param obj el objeto con el cual comparar
 * @return true si los objetos son los mismos; false de otra manera
 */
public boolean equals( Object obj )
{
   boolean valorDev = false;

   if (obj instanceof BitSetExtendido)
   {
      BitSetExtendido otro = (BitSetExtendido) obj;

      if (this == otro)
         valorDev = true;
      else if (bits == null)
         valorDev = (otro.bits == null);
      else if (lonBits == otro.lonBits)
      {
         // Ignora las partes no utilizadas del conjunto de bits
         // (la cual podría existir si el conjunto ha reducido su
         // tamaño)

         int limite = (bits.length < otro.bits.length) ?
            bits.length : otro.bits.length;

         valorDev = true;
         for (int i = 0 ; i < limite; ++i)
         {
            if (bits[i] != otro.bits[i])
            {
               valorDev = false;
               break;
            }
         }
      }
   }

   return valorDev;
}

/**
 * Obtiene el código hash.
 */
public int hashCode()
{
   int valorDev = lonBits;

   if (bits != null)
   {
      for (int i = 0 ; i < bits.length; ++i)
         valorDev ^= (int) (bits[i] ^ (bits[i] >>> 32));
   }
```

```
    return valorDev;
}

/**
 * Devuelve el número de bits en el conjunto
 *
 * @return el número de bits en el conjunto
 */
public int tamaño()
{
    return lonBits;
}

/**
 * Extiende o reduce el tamaño del conjunto de bits
 *
 * @param longitud nuevo tamaño del conjunto de bits.
 * @return this
 */
public BitSetExtendido tamaño( int longitud )
{
    if (longitud < lonBits)
    {
        // Limpia los bits para asegurar que todos los bits no
        // usados sean siempre cero. Esto hace más comparar
        // y extender.

        limpiar( longitud, lonBits - longitud );
        lonBits = longitud;
    }
    else
        acasoExtendido( longitud );

    return this;
}

/**
 * Obtiene el valor de un bit en particular
 *
 * @param ubicacion del bit a leer
 * @return el valor del bit
 */
public boolean obtener( int ubicacion )
{
    acasoExtendido( ubicacion + 1 );

    int esteLong = ubicacion / BITSENLONG;
    int esteBit = ubicacion % BITSENLONG;

    return ((bits[esteLong] & (1L<<esteBit)) != 0);
}

/**
 * Establece el valor de un bit en particular a true
 *
 * @param ubicacion el bit a establecer
```

```
 * @return this
 */
public BitSetExtendido establecer( int ubicacion )
{
   acasoExtendido( ubicacion + 1 );

   int esteLong = ubicacion / BITSENLONG;
   long esteBit = ubicacion % BITSENLONG;

   bits[esteLong] |= (1L << esteBit);

   return this;
}

/**
 * Establece un rango de bits a true. Una cantidad de bits igual a
 * <code>longitud</code> en la <code>ubicacion</code> son
 * establecidos a true.
 *
 * @param ubicacion inicio del rango
 * @param longitud longitud del rango
 * @return this
 */
public BitSetExtendido establecer( int ubicacion, int longitud )
{
   acasoExtendido( ubicacion + longitud );

   while (longitud > 0)
   {
      establecerLongEn( ubicacion, longitud, -1L );
      ubicacion += BITSENLONG;
      longitud -= BITSENLONG;
   }

   return this;
}

/**
 * Establece el valor de un bit en particular a false
 *
 * @param ubicacion el bit a limpiar
 * @return this
 */
public BitSetExtendido limpiar( int ubicacion )
{
   acasoExtendido( ubicacion + 1 );

   int esteLong = ubicacion / BITSENLONG;
   long esteBit = ubicacion % BITSENLONG;

   bits[esteLong] &= ~(1L << esteBit);

   return this;
}

/**
```

```
 * Establece un rango de bits a false. Una cantidad igual a
 * <code>ubicacion</code> de bits en la <code>ubicacion</code> son
 * establecidos a false.
 *
 * @param ubicacion inicio del rango
 * @param longitud longitud del rango
 * @return this
 */
public BitSetExtendido limpiar( int ubicacion, int longitud )
{
   acasoExtendido( ubicacion + longitud );

   while (longitud > 0)
   {
      establecerLongEn( ubicacion, longitud, 0 );
      ubicacion += BITSENLONG;
      longitud -= BITSENLONG;
   }

   return this;
}

/**
 * Invierte el valor de un bit en particular.
 *
 * @param ubicacion el bit a invertir
 * @return this
 */
public BitSetExtendido invertir( int ubicacion )
{
   acasoExtendido( ubicacion + 1 );

   int esteLong = ubicacion / BITSENLONG;
   long esteBit = ubicacion % BITSENLONG;

   bits[esteLong] ^= 1L << esteBit;

   return this;
}

/**
 * Invierte el valor de un rango de bits. Una cantidad de bits igual
 * a <code>longitud</code> en la <code>ubicacion</code> son invertidos.
 *
 * @param ubicacion inicio del rango
 * @param longitud longitud del rango
 * @return this
 */
public BitSetExtendido invertir( int ubicacion, int longitud )
{
   acasoExtendido( ubicacion + longitud );

   while (longitud > 0)
   {
      long valorNuevo = obtenerLongEn( ubicacion, longitud ) ^ -1L;
```

```
      establecerLongEn( ubicacion, longitud, valorNuevo );
      ubicacion += BITSENLONG;
      longitud -= BITSENLONG;
    }

    return this;
}

/**
 * Realiza la operación AND al <strong>total</strong> de este
 * conjunto de bits con el conjunto de bits especificados. Si
 * <code>conjBits</code> es más corto que este conjunto, los valores
 * adicionales se asumen como cero; si es más largo, se trunca.
 *
 * @param conjBits el conjunto de bits con el que se hace el AND
 * @return this
 */
public BitSetExtendido and( BitSetExtendido conjBits )
{
    if (conjBits.lonBits < lonBits)
        conjBits = (new BitSetExtendido( lonBits )).establecerSubConjunto(0,
          conjBits );

    return and( 0, conjBits );
}

/**
 * Realiza la operación AND a un rango de bits en este conjunto de
 * bits con el conjunto de bits especificado.<code>ubicacion</code>
 * da el comienzo del rango, y la longitud de <code>conjBits</code>
 * define su longitud.
 *
 * @param ubicacion el comienzo del rango de bits con el que se hace el AND
 * @param conjBits el conjunto de bits con el que se hace el AND
 * @return this
 */
public BitSetExtendido and( int ubicacion, BitSetExtendido conjBits )
{
    acasoExtendido( ubicacion + conjBits.lonBits );

    int otraUbicacion = 0;

    for (int longitud = conjBits.lonBits; longitud > 0;
          longitud -= BITSENLONG)
    {
        long valorNuevo = obtenerLongEn( ubicacion, longitud ) &
          conjBits.obtenerLongEn( otraUbicacion, longitud );

        establecerLongEn( ubicacion, longitud, valorNuevo );
        ubicacion += BITSENLONG;
        otraUbicacion += BITSENLONG;
        longitud -= BITSENLONG;
    }

    return this;
}
```

```
/**
 * Realiza la operación OR al <strong>total</strong> de este
 * conjunto de bits con el conjunto de bits especificados. Si
 * <code>conjBits</code> es más corto que este conjunto, los valores
 * adicionales se asumen como cero; si es más largo, se trunca.
 *
 * @param conjBits el conjunto de bits con el que se hace el OR
 * @return this
 */
public BitSetExtendido or( BitSetExtendido conjBits )
{
   if (conjBits.lonBits < lonBits)
      conjBits = (new BitSetExtendido( lonBits )).establecerSubConjunto(0,
         conjBits );

   return or( 0, conjBits );
}

/**
 * Realiza la operación OR a un rango de bits en este conjunto de
 * bits con el conjunto de bits especificado.<code>ubicacion</code>
 * da el comienzo del rango, y la longitud de <code>conjBits</code>
 * define su longitud.
 *
 * @param ubicacion el comienzo del rango de bits con el que se hace OR
 * @param conjBits el conjunto de bits con el que se hace el OR
 * @return this
 */
public BitSetExtendido or( int ubicacion, BitSetExtendido conjBits )
{
   acasoExtendido( ubicacion + conjBits.lonBits );

   int otraUbicacion = 0;

   for (int longitud = conjBits.lonBits; longitud > 0;
        longitud -= BITSENLONG)
   {
      long valorNuevo = obtenerLongEn( ubicacion, longitud ) |
         conjBits.obtenerLongEn( otraUbicacion, longitud );

      establecerLongEn( ubicacion, longitud, valorNuevo );
      ubicacion += BITSENLONG;
      otraUbicacion += BITSENLONG;
      longitud -= BITSENLONG;
   }

   return this;
}

/**
 * Realiza la operación XOR al <strong>total</strong> de este
 * conjunto de bits con el conjunto de bits especificados. Si
 * <code>conjBits</code> es más corto que este conjunto, los valores
 * adicionales se asumen como cero; si es más largo, se trunca.
 *
```

```
 * @param conjBits el conjunto de bits con el que se hace el XOR
 * @return this
 */
public BitSetExtendido xor( BitSetExtendido conjBits )
{
   if (conjBits.lonBits < lonBits)
      conjBits = (new BitSetExtendido( lonBits )).establecerSubConjunto(0,
        conjBits );

   return xor( 0, conjBits );
}

/**
 * Realiza la operación XOR a un rango de bits en este conjunto de
 * bits con el conjunto de bits especificado.<code>ubicacion</code>
 * da el comienzo del rango, y la longitud de <code>conjBits</code>
 * define su longitud.
 *
 * @param ubicacion el comienzo del rango de bits con el que se hace XOR
 * @param conjBits el conjunto de bits con el que se hace el XOR
 * @return this
 */
public BitSetExtendido xor( int ubicacion, BitSetExtendido conjBits )
{
   acasoExtendido( ubicacion + conjBits.lonBits );

   int otraUbicacion = 0;

   for (int longitud = conjBits.lonBits; longitud > 0;
        longitud -= BITSENLONG)
   {
      long valorNuevo = obtenerLongEn( ubicacion, longitud ) ^
        conjBits.obtenerLongEn( otraUbicacion, longitud );

      establecerLongEn( ubicacion, longitud, valorNuevo );
      ubicacion += BITSENLONG;
      otraUbicacion += BITSENLONG;
      longitud -= BITSENLONG;
   }

   return this;
}

/**
 * Copia bits de un array de longs. Los contenidos de los longs
 * determinana directamente los bits. El bit 0 del elemento 0 del
 * array es el bit 0 del conjunto de bits, el bit 63 del elemento n
 * del array es el bit n*64+63 del conjunto, sujeto a la máxima
 * longitud dada por <code>longitud</code>.
 *
 * @param longitud número de bits en el conjunto
 * @param inBits valores iniciales de los bits
 * @return this
 */
public BitSetExtendido inicializarTodo( int longitud, long [] inBits )
{
```

```
    int longsNecesarios = (longitud-1) / BITSENLONG + 1;

    if (bits == null || (longsNecesarios > bits.length))
        bits = new long[longsNecesarios];
    if (inBits.length < longsNecesarios)
        longsNecesarios = inBits.length;

    System.arraycopy( inBits, 0, bits, 0, longsNecesarios );
    lonBits = longitud;

    return this;
}

/**
 * Establece el valor de un bit en particular al boolean dado
 *
 * @param ubicacion el bit a establecer
 * @param valor el valor a establecer
 * @return this
 */
public BitSetExtendido establecerBooleanEn( int ubicacion, boolean valor )
{
    if (valor)
        return establecer( ubicacion );
    else
        return limpiar( ubicacion );
}

/**
 * Retorna el valor del conjunto de bits establecido como un byte
 * (truncandolo si este es más grande que ocho bits).
 *
 * @return el conjunto de bits representado como un byte
 */
public byte valorByte()
{
    return obtenerByteEn( 0, BITSENBYTE );
}

/**
 * Retorna el valor de los ocho bits en una ubicación dada en el
 * conjunto como un byte.
 *
 * @param ubicacion la ubicación desde donde leer
 * @return los bits en la ubicación representadas como un byte
 */
public byte obtenerByteEn( int ubicacion )
{
    return obtenerByteEn( ubicacion, BITSENBYTE );
}

/**
 * Retorna como un byte el valor de una cantidad de bits igual a
 * <code>longitud</code> en una ubicación dada. Si <code>longitud</code>
 * es menor que ocho, el valor es cero extendido; si es más grande
 * este es truncado.
```

```
 *
 * @param ubicacion la ubicación desde donde leer
 * @param longitud el número de bits a leer
 * @return los bits representados como un byte
 */
public byte obtenerByteEn( int ubicacion, int longitud )
{
   if (longitud > BITSENBYTE)
      longitud = BITSENBYTE;

   return (byte) obtenerLongEn( ubicacion, longitud );
}

/**
 * Establece el valor de los ocho bits en una ubicación dada en el
 * para representar un byte.
 *
 * @param ubicacion la ubicación para la asignación
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerByteEn( int ubicacion, byte valor )
{
   return establecerByteEn( ubicacion, BITSENBYTE, valor );
}

/**
 * Establece el valor de una cantidad igual a <code>longitud</code>
 * de bits en una ubicación dada en el conjunto para representar un
 * byte. Si <code>longitud</code> es menor que ocho, el valor se
 * trunca; longitudes más grandes que ocho son tratadas como ocho.
 *
 * @param ubicacion la ubicación para asignar
 * @param longitud el número de bits del valor a escribir
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerByteEn( int ubicacion, int longitud,
  byte valor )
{
   if (longitud > BITSENBYTE)
      longitud = BITSENBYTE;

   return establecerLongEn( ubicacion, longitud, valor );
}

/**
 * Retorna el valor del conjunto de bits como un char (truncandolo
 * si este más grande que dieciseis bits).
 *
 * @return el conjunto de bits representado como un char
 */
public char valorChar()
{
   return obtenerCharEn( 0, BITSENCHAR );
}
```

```
/**
 * Retorna el valor de los dieciséis bits en una ubicación dada en
 * el conjunto como un char.
 *
 * @param ubicacion la ubicación desde donde leer
 * @return los bits en esa ubicación representada como un char
 */
public char obtenerCharEn( int ubicacion )
{
   return obtenerCharEn( ubicacion, BITSENCHAR );
}

/**
 * Retorna el valor de una cantidad igual a <code>longitud</code> de
 * bits en una ubicación dada en el conjunto como un char. Si
 * code>longitud</code> es menor que dieciséis, el valor es cero
 * extendido; si es más grande este es truncado.
 *
 * @param ubicacion la ubicación desde donde leer
 * @param longitud el número de bits a leer
 * @return los bits representados como un char
 */
public char obtenerCharEn( int ubicacion, int longitud )
{
   if (longitud > BITSENCHAR)
      longitud = BITSENCHAR;

   return (char) obtenerLongEn( ubicacion, longitud );
}

/**
 * Establece el valor de los dieciséis bits en una ubicación dada en
 * el conjunto para representar un char.
 *
 * @param ubicacion la ubicación para asignar
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerCharEn( int ubicacion, char valor )
{
   return establecerCharEn( ubicacion, BITSENCHAR, valor );
}

/**
 * Establece el valor de una cantidad igual a <code>longitud</code>
 * de bits en la ubicación dada en el conjunto para representar un
 * char. Si <code>longitud</code> es menor que dieciséis, el valor
 * se trunca; longitudes más grandes a dieciséis son tratadas como
 * de dieciseis.
 *
 * @param ubicacion la ubicación para asignar
 * @param longitud el número de bits del valor a escribir
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
```

```
public BitSetExtendido establecerCharEn( int ubicacion, int longitud,
  char valor )
{
   if (longitud > BITSENCHAR)
      longitud = BITSENCHAR;

   return establecerLongEn( ubicacion, longitud, valor );
}

/**
 * Retorna el valor del conjunto de bits como un short (truncandolo
 * si este es más largo que dieciséis bits).
 *
 * @return el conjunto de bits representados como un short
 */
public short valorShort()
{
   return obtenerShortEn( 0, BITSENSHORT );
}

/**
 * Retorna el valor de los dieciséis bits en una ubicación dada en
 * el conjunto como un short.
 *
 * @param ubicacion la ubicación desde donde leer
 * @return los bits en esa ubicación representada como un short
 */
public short obtenerShortEn( int ubicacion )
{
   return obtenerShortEn( ubicacion, BITSENSHORT );
}

/**
 * Retorna el valor como un short de una cantidad de bits igual a
 * <code>longitud</code> en una ubicación dada en el conjunto. Si
 * <code>longitud</code> es menor que dieciséis, el valor es cero
 * extendido; si es más grande se trunca.
 *
 * @param ubicacion la ubicación desde donde leer
 * @param longitud el número de bits a leer
 * @return los bits representados como un short
 */
public short obtenerShortEn( int ubicacion, int longitud )
{
   if (longitud > BITSENSHORT)
      longitud = BITSENSHORT;

   return (short) obtenerLongEn( ubicacion, longitud );
}

/**
 * Establece el valor de los dieciséis bits en una ubicación dada en
 * el conjunto para representar un short.
 *
 * @param ubicacion la ubicación para asignar
 * @param valor el valor a escribir detro del conjunto
```

```
 * @return this
 */
public BitSetExtendido establecerShortEn( int ubicacion, short valor )
{
   return establecerShortEn( ubicacion, BITSENSHORT, valor );
}

/**
 * Establece el valor de una cantidad de bits igual a <code>longitud</code>
 * para representar un short. Si <code>longitud</code> es menor que
 * dieciséis, el valor se trunca; longitudes más grandes que dieciséis
 * son tratados como de dieciséis.
 *
 * @param ubicacion la ubicación para asignar
 * @param longitud el número de bits del valor a escribir
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerShortEn( int ubicacion, int longitud,
   short valor )
{
   if (longitud > BITSENSHORT)
      longitud = BITSENSHORT;

   return establecerLongEn( ubicacion, longitud, valor );
}

/**
 * Retorna el valor del conjunto de bits como un int (truncandolo si
 * este más grande que treinta y dos bits).
 *
 * @return el conjunto de bits representados como un int
 */
public int valorInt()
{
   return obtenerIntEn( 0, BITSENINT );
}

/**
 * Devuelve el valor de los treinta y dos bits en la ubicación dada
 * en el conjunto como un int.
 *
 * @param ubicacion la ubicación desde donde leer
 * @return los bits en esa ubicación representadas como un int
 */
public int obtenerIntEn( int ubicacion )
{
   return obtenerIntEn( ubicacion, BITSENINT );
}

/**
 * Retorna el valor como un int de una cantidad de bits igual a
 * <code>longitud</code> en la ubicación dada en el conjunto. Si
 * <code>longitud</code> es menor que treinta y dos el valor es cero
 * extendido; si es más grande este se trunca.
 *
```

```
 * @param ubicacion la ubicación desde donde leer
 * @param longitud el número de bits a leer
 * @return los bits representados como un int
 */
public int obtenerIntEn( int ubicacion, int longitud )
{
   if (longitud > BITSENINT)
      longitud = BITSENINT;

   return (int) obtenerLongEn( ubicacion, longitud );
}

/**
 * Establece el valor de los trenta y dos bits en una ubicación dada
 * en el conjunto para representar un int.
 *
 * @param ubicacion la ubicación para asignar
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerIntEn( int ubicacion, int valor )
{
   return establecerIntEn( ubicacion, BITSENINT, valor );
}

/**
 * Establece el valor de una cantidad de bits igual a <code>longitud</code>
 * en una ubicación dada en el conjunto para representar un int. Si
 * <code>longitud</code> es menor que treinta y dos, el valor se
 * trunca; longitudes más grandes que treinta y dos son tratadas
 * como de trentaidos.
 *
 * @param ubicacion la ubicación para asignar
 * @param longitud el número de bits del valor a escribir
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerIntEn( int ubicacion, int longitud,
   int valor )
{
   if (longitud > BITSENINT)
      longitud = BITSENINT;

   return establecerLongEn( ubicacion, longitud, valor );
}

/**
 * Retorna el valor del conjunto de bits como un long (truncandolo
 * si este es más largo que sesenta y cuatro bits).
 *
 * @return el conjunto de bits representados como un long
 */
public long valorLong()
{
   return obtenerLongEn( 0, BITSENLONG );
}
```

```
/**
 * Retorna el valor de los sesenta y cuatro bits en una ubicación
 * dada en el conjunto como un long.
 *
 * @param ubicacion la ubicación desde donde leer
 * @return los bits en esa ubicación representada como un long
 */
public long obtenerLongEn( int ubicacion )
{
   return obtenerLongEn( ubicacion, BITSENLONG );
}

/**
 * Devuelve como long el valor de una cantidad de bits igual a
 * <code>longitud</code> en una ubicación dada en el conjunto. Si
 * <code>longitud</code> es menor que sesenta y cuatro, el valor es
 * cero extendido; si es más grande este se trunca.
 *
 * @param ubicacion la ubicación desde donde leer
 * @param longitud el número de bits a leer
 * @return los bits representados como un long
 */
public long obtenerLongEn( int ubicacion, int longitud )
{
   if (longitud > BITSENLONG)
      longitud = BITSENLONG;

   int bloque = ubicacion / BITSENLONG;
   int desp = ubicacion % BITSENLONG;
   long masc = -1L >>> (BITSENLONG - longitud);
   long valorDev;

   acasoExtendido( ubicacion + longitud );

   if (desp == 0)
      valorDev = bits[bloque];
   else
   {
      valorDev = bits[bloque] >>> desp;
      if (BITSENLONG - desp < longitud)
         valorDev |= bits[bloque+1] << (BITSENLONG - desp);
   }

   return valorDev & masc;
}

/**
 * Establece el valor de los sesenta y cuatro bits en una ubicación
 * dada en el conjunto para representar un long.
 *
 * @param ubicacion la ubicación para asignar
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerLongEn( int ubicacion, long valor )
```

```
{
   return establecerLongEn( ubicacion, BITSENLONG, valor );
}

/**
 * Establece el valor de una cantidad de bits igual a <code>longitud</code>
 * en una ubicación dada en el conjunto para representar un long. Si
 * <code>longitud</code> es menor que sesenta y cuatro, el valor es
 * truncado; longitudes más grandes que sesenta y cuatro son tratadas
 * como de sesenta y cuatro.
 *
 * @param ubicacion la ubicación para asignar
 * @param longitud el número de bits de el valor a escribir
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerLongEn( int ubicacion, int longitud,
   long valor )
{
   if (longitud > BITSENLONG)
      longitud = BITSENLONG;

   int bloque = ubicacion / BITSENLONG;
   int desp = ubicacion % BITSENLONG;
   long masc = -1L >>> (BITSENLONG - longitud);

   acasoExtendido( ubicacion + longitud );

   valor &= masc;

   if (desp == 0)
      bits[bloque] = (bits[bloque] & ~masc) | valor;
   else
   {
      bits[bloque] = (bits[bloque] & ~(masc << desp)) |
         (valor << desp);
      if (BITSENLONG - desp < longitud)
         bits[bloque+1] = (bits[bloque+1] &
            ~(masc >>> (BITSENLONG - desp))) |
            (valor >>> (BITSENLONG - desp));
   }

   return this;
}

/**
 * Devuelve el valor del conjunto de bots como un float (truncandolo
 * si este es más grande que treinta y dos bits).
 *
 * @return el conjunto de bits representados como un float
 */
public float valorFloat()
{
   return obtenerFloatEn( 0 );
}
```

```
/**
 * Devuelve el valor de los treinta y dos bits en una ubicación dada
 * en el conjunto como un float.
 *
 * @param ubicacion la ubicación desde donde leer
 * @return los bits en la ubicación representadas como un float
 */
public float obtenerFloatEn( int ubicacion )
{
   return Float.intBitsToFloat( obtenerIntEn( ubicacion, BITSENINT ) );
}

/**
 * Establece el valor de treinta y dos bits en una ubicación dada en
 * el conjunto para representar un float
 *
 * @param ubicacion la ubicación para asignar
 * @param valor el valor a escribir dentro del conjunto
 * @return this
 */
public BitSetExtendido establecerFloatEn( int ubicacion, float valor )
{
   return establecerIntEn( ubicacion, BITSENINT,
        Float.floatToIntBits( valor ) );
}

/**
 * Devuelve el valor del conjunto de bits como un double (trucándolo
 * si este es más largo que sesenta y cuatro bits).
 *
 * @return el conjunto de bits representados como un double
 */
public double valorDouble()
{
   return obtenerDoubleEn( 0 );
}

/**
 * Devuelve el valor de los sesenta y cuatro bits en una ubicación
 * dada en el conjunto como un double.
 *
 * @param ubicacion la ubicación desde donde leer
 * @return los bits en la ubicación representadas como un double
 */
public double obtenerDoubleEn( int ubicacion )
{
   return Double.longBitsToDouble( obtenerLongEn(ubicacion, BITSENLONG ));
}

/**
 * Establece el valor de sesenta y cuatro bits en una ubicación dada
 * en el conjunto para representar un double.
 *
 * @param ubicacion la ubicación para asignar
 * @param valor el valor para escribir dentro del conjunto
 * @return this
```

```
 */
public BitSetExtendido establecerDoubleEn( int ubicacion, double valor )
{
   return establecerLongEn( ubicacion, BITSENLONG,
      Double.doubleToLongBits( valor ) );
}

/**
 * Obtiene un subconjunto de bits. Construye otro conjunto de bits
 * formados por los bits desde el valor de la variable ubicacion
 * hasta el valor de ubicacion + longitud - 1
 *
 * @param ubicacion inicio del subconjunto
 * @param longitud longitud del subconjunto
 * @return el subconjunto de bits indicado
 */
public BitSetExtendido encontrarSubConjunto( int ubicacion, int longitud )
{
   acasoExtendido( ubicacion + longitud );

   int segmentos = (longitud-1) / BITSENLONG + 1;
   long [] nuevosBits = new long[segmentos];
   BitSetExtendido valorDev = new BitSetExtendido();

   valorDev.lonBits = longitud;

   for (int i = 0; i < segmentos; ++i)
   {
      nuevosBits[i] = obtenerLongEn( ubicacion, longitud );
      ubicacion += BITSENLONG;
      longitud -= BITSENLONG;
   }
   valorDev.bits = nuevosBits;

   return valorDev;
}

/**
 * Establece el valor de un subconjunto de bits. Los bits en la
 * ubicación <code>ubicacion</code> son establecidos a valores de
 * bits obtenidos del conjunto de bits <code>conjBits</code>. El
 * número de bits afectados es estipulado por el tamaño de conjBits.
 *
 * @param ubicacion inicio del subconjunto a alterar
 * @param conjBits conjunto de bits desde donde obtener los nuevos valores
 * @return this
 */
public BitSetExtendido establecerSubConjunto( int ubicacion,
  BitSetExtendido conjBits )
{
   acasoExtendido( ubicacion + conjBits.lonBits );

   int otraUbicacion = 0;

   for (int longitud = conjBits.lonBits; longitud > 0;
        longitud -= BITSENLONG)
```

```
    {
      establecerLongEn( ubicacion, longitud,
        conjBits.obtenerLongEn( otraUbicacion, longitud ) );
      ubicacion += BITSENLONG;
      otraUbicacion += BITSENLONG;
    }

    return this;
}

/**
 * Intercambia el valor de un subconjunto de bits con otro conjunto.
 * Una cantidad de bits igual a <code>longitud</code> en la ubicación
 * <code>ubicacion</code> en cada conjunto de bits son intercambiados
 * uno al otro.
 *
 * @param ubicacion incio del subconjunto a intercambiar
 * @param longitud numero de bits a intercambia
 * @param conjBits conjunto de bits con los que se intercambian valores
 * @return this
 */
public BitSetExtendido intercambiarSubConj( int ubicacion, int longitud,
    BitSetExtendido conjBits )
{
    acasoExtendido( ubicacion + longitud );

    for (int lon = longitud; lon > 0; lon -= BITSENLONG)
    {
      long guardado = obtenerLongEn( ubicacion, lon );

      establecerLongEn(ubicacion,lon,
        conjBits.obtenerLongEn(ubicacion,lon));
      conjBits.establecerLongEn( ubicacion, lon, guardado );
      ubicacion += BITSENLONG;
    }

    return this;
}

/**
 * Inserta un subconjunto de bits dentro del conjunto actual antes
 * del bit en la ubicación determinada por la variable
 * <code>ubicacion</code>. Los bits en esta ubicación y los de
 * ubicaciones mayores se adecuan para hacer lugar. Si la variable
 * <code>ubicacion</code> es negativa, la inserción se
 * hace al final.
 *
 * @param ubicacion punto de inserción
 * @param conjBits conjunto de bits a insertar
 * @return this
 */
public BitSetExtendido insertarSubConjunto( int ubicacion,
    BitSetExtendido conjBits )
{
    if (ubicacion < 0)
      ubicacion = lonBits;
```

```
    else
    {
      int bitsACambiar = lonBits - ubicacion + conjBits.bits.length;

      desplazarIzquierda( ubicacion, bitsACambiar, conjBits.bits.length );
    }
    return establecerSubConjunto( ubicacion, conjBits );
}

/**
 * Remueve un segemento de un conjunto de bits y acomoda al resto.
 * La longitud completa del conjunto de bits se reduce por el valor
 * de <code>longitud</code>
 *
 * @param ubicacion punto de borrado
 * @param longitud número de bits a remover
 * @return this
 */
public BitSetExtendido borrarSubConjunto( int ubicacion, int longitud )
{
    desplazarDerechaSinSigno( ubicacion, lonBits - ubicacion, longitud );
    lonBits -= longitud;

    return this;
}

/**
 * Mueve un subconjunto de bits a la izquierda. Se desplaza dentro de
 * un campo con una cantidad de bits igual a <code>longitud</code>
 * comenzando en la ubicación determinada por la variable
 * <code>ubicacion</code>, la variable <code>desp</code> determina
 * cuantos lugares a la izquierda lo coloca. Una cantidad de bits
 * igual a <code>desp</code> que son los más inferiores en el campo se
 * dejan como estaban. Para dar un ejemplo
 * <code>
 *                  76543210
 *     Bits iniciales:  01100101
 *
 *     arrastrarIzquierda( 2, 4, 1 )
 *
 *     01|1001|01 -> 01|0011|01
 * </code>
 * Los bits 0, 1, 6 y 7 no se tocan pues estan por fuera del campo
 * definido por las variables <code>ubicacion</code> y <code>longitud</code>.
 * El bit 5 "cae del final" del campo y se pierde, los bits 3 y 4
 * llegan a ser los bits 4 y 5, y el valor del bit 2 se mantiene sin
 * cambio.
 *
 * @param ubicacion inicio del campo a arrastrar.
 * @param longitud longitud del campo a arrastrar
 * @param desp número de posiciones a arrastrar a la izquierda
 * @return this
 */
public BitSetExtendido arrastrarIzquierda( int ubicacion, int longitud,
    int desp )
{
```

```
    acasoExtendido( ubicacion+longitud );

    // Emula el comportamiento del operador Java <<, es decir
    // normaliza el valor de desp para estar entre 0 y longitud - 1.

    desp %= longitud;
    if (desp < 0)
      desp += longitud;
    if (desp > 0)
    {
      // Desplaza los bits

      int ubicacionDestino = ubicacion + longitud - BITSENLONG;
      int ubicacionOrigen = ubicacionDestino - desp;
      if (ubicacionOrigen < 0)
      {
        ubicacionDestino = ubicacion + desp;
        ubicacionOrigen = ubicacion;
      }

      longitud -= desp;
      while (longitud > 0)
      {
        establecerLongEn( ubicacionDestino, longitud,
          obtenerLongEn( ubicacionOrigen, longitud ) );
        ubicacionOrigen -= BITSENLONG;
        ubicacionDestino -= BITSENLONG;
        longitud -= BITSENLONG;
      }
    }

    return this;
  }

  /**
   * Mueve un subconjunto de bits a la izquierda. Se desplaza dentro de
   * un campo con una cantidad de bits igual a <code>longitud</code>
   * comenzando en la ubicación determinada por la variable
   * <code>ubicacion</code>, la variable <code>desp</code> determina
   * cuantos lugares a la izquierda lo coloca. Una cantidad de bits
   * igual a <code>desp</code> que son los más inferiores en el campo se
   * hacen igual a cero. Ejemplo:
   * <code>
   *    01|1001|01 -> desplazarIzquierda( 2, 4, 1 ) -> 01|0010|01
   * </code>
   * Lea la explicación para arrastrarIzquierda si esta no esta clara.
   *
   * @param ubicacion inicio del campo a desplazar
   * @param longitud longitud del campo a desplazar
   * @param desp número de posiciones a desplazar
   * @return this
   *
   * @see #arrastrarIzquierda
   */
  public BitSetExtendido desplazarIzquierda( int ubicacion, int longitud,
    int desp )
```

```
{
   if (desp % longitud != 0)
   {
      // Desplaza los bits y hace cero los que sobran

      arrastrarIzquierda( ubicacion, longitud, desp );
      limpiar( ubicacion, desp );
   }

   return this;
}

/**
 * Mueve un subconjunto de bits a la derecha. Se desplaza dentro de
 * un campo con una cantidad de bits igual a <code>longitud</code>
 * comenzando en la ubicación determinada por la variable
 * <code>ubicacion</code>, la variable <code>desp</code> determina
 * cuantos lugares a la derecha la coloca. Una cantidad de bits
 * igual a <code>desp</code> que son los más altos en el campo se
 * dejan como estaban. Ejemplo:
 * <code>
 *     01|1001|01 -> arrastrarDerecha( 2, 4, 1 ) -> 01|1100|01
 * </code>
 * Lea la explicación para arrastrarIzquierda si esta explicación le
 * parece poca clara.
 *
 * @param ubicacion inicio del campo a arrastrar
 * @param longitud longitud del campo a arrastrar
 * @param desp número de posiciones a arrastrar a la derecha
 * @return this
 *
 * @see #arrastrarIzquierda
 */
public BitSetExtendido arrastrarDerecha( int ubicacion, int longitud,
   int desp )
{
   acasoExtendido( ubicacion+longitud );

   // Emula el comportamiento del operador Java >>>, es decir
   // normaliza el valor de desp entre 0 y longitud - 1.

   desp %= longitud;
   if (desp < 0)
      desp += longitud;
   if (desp > 0)
   {
      int ubicacionDestino = ubicacion;
      int ubicacionOrigen = ubicacion + desp;

      longitud -= desp;
      while (longitud > 0)
      {
         establecerLongEn( ubicacionDestino, longitud,
            obtenerLongEn( ubicacionOrigen, longitud ) );
         ubicacionOrigen += BITSENLONG;
         ubicacionDestino += BITSENLONG;
```

```
            longitud -= BITSENLONG;
        }
    }

    return this;
}

/**
 * Mueve un subconjunto de bits a la derecha. Se desplaza dentro de
 * un campo con una cantidad de bits igual a <code>longitud</code>
 * comenzando en la ubicación determinada por la variable
 * <code>ubicacion</code>, la variable <code>desp</code> determina
 * cuantos lugares a la derecha la coloca. Una cantidad de bits
 * igual a <code>desp</code> que son los más altos en el campo se
 * hacen igual a cero. Ejemplo:
 * <code>
 *     01|1001|01 -> desplazarDerechaSinSigno( 2, 4, 1 ) -> 01|0100|01
 * </code>
 * Lea la explicación para arrastrarIzquierda si esta le parece poco
 * clara.
 *
 * @param ubicacion inicio del campo a desplazar
 * @param longitud longitud del campo a desplazar
 * @param desp número de posiciones a desplazar a la derecha
 * @return this
 *
 * @see #arrastrarIzquierda
 */
public BitSetExtendido desplazarDerechaSinSigno( int ubicacion,int longitud,
    int desp )
{
    if (desp % longitud != 0)
    {
        // Shift the bits and zero what's left over

        arrastrarDerecha( ubicacion, longitud, desp );
        limpiar( ubicacion + longitud - desp, desp );
    }

    return this;
}

/**
 * Desplaza un subconjunto de bits a la derecha con extensión de
 * signo.Se desplaza dentro de un campo con una cantidad de bits
 * igual a <code>longitud</code> comenzando en la ubicación
 * determinada por la variable <code>ubicacion</code>, la variable
 * <code>desp</code> determina cuantos lugares a la derecha la
 * coloca. Una cantidad de bits igual a <code>desp</code> que son
 * los más altos en el campo se establecen todos a el valor del bit
 * más alto. Ejemplo
 * hacen igual a cero. Ejemplo:
 * <code>
 *     01|1001|01 -> desplazarDerechaConSigno( 2, 4, 1 ) -> 01|1100|01
 * </code>
 * Lea la explicación para arrastrarIzquierda si esta le parece poco
```

```
 * clara.
 *
 * @param ubicacion inicio del campo a desplazar
 * @param longitud longitud del campo a desplazar
 * @param desp número de posiciones a desplazar hacia la derecha
 * @return this
 *
 * @see #arrastrarIzquierda
 */
public BitSetExtendido desplazarDerechaConSigno( int ubicacion,int longitud,
   int desp )
{
   if (desp % longitud != 0)
   {
      // Desplaza los bits y extiende el signo al resto
      arrastrarDerecha( ubicacion, longitud, desp );
      if (obtener( ubicacion + longitud - 1 ))
         establecer( ubicacion + longitud - desp, desp - 1 );
      else
         limpiar( ubicacion + longitud - desp, desp - 1 );
   }

   return this;
}

/**
 * Rota un subconjunto de bits hacia la izquierda.Se rota dentro
 * de un campo con una cantidad de bits igual a <code>longitud</code>
 * comenzando en la ubicación determinada por la variable
 * <code>ubicacion</code>, la variable <code>desp</code> determina
 * cuantos lugares a la derecha la izquierda la coloca. Los bits
 * perdidos de la parte más alta del campo se reinsertan en la parte
 * más baja del campo. Ejemplo:
 * <code>
 *    01|1001|01 -> rotarIzquierda( 2, 4, 1 ) -> 01|0011|01
 * </code>
 * Lea la explicación para arrastrarIzquierda si esta parece poco
 * clara.
 *
 * @param ubicacion inicio del campo a rotar
 * @param longitud longitud del campo a rotar
 * @param desp número de posiciones a rotar hacia la izquierda
 * @return this
 *
 * @see #arrastrarIzquierda
 */
public BitSetExtendido rotarIzquierda( int ubicacion, int longitud,int desp)
{
   if (desp % longitud != 0)
   {
      BitSetExtendido guardado =
         encontrarSubConjunto( ubicacion + longitud - desp, desp );

      arrastrarIzquierda( ubicacion, longitud, desp );
      establecerSubConjunto( ubicacion, guardado );
   }
```

```
    return this;
}

/**
 * Rota un subconjunto de bits hacia la derecha.Se rota dentro
 * de un campo con una cantidad de bits igual a <code>longitud</code>
 * comenzando en la ubicación determinada por la variable
 * <code>ubicacion</code>, la variable <code>desp</code> determina
 * cuantos lugares a la derecha la izquierda la coloca. Los bits
 * perdidos de la parte más baja del campo se reinsertan en la parte
 * más alta del campo. Ejemplo:
 * <code>
 *     01|1001|01 -> rotarDerecha( 2, 4, 1 ) -> 01|1100|01
 * </code>
 * Lea la explicación para arrastrarIzquierda si esta parece poco
 * clara.
 *
 * @param ubicacion inicio del campo a rotar
 * @param longitud longitud del campo a rotar
 * @param desp número de posiciones a rotar hacia la derecha
 * @return this
 *
 * @see #arrastrarIzquierda
 */

public BitSetExtendido rotarDerecha( int ubicacion, int longitud, int desp )
{
    if (desp % longitud != 0)
    {
        BitSetExtendido guardado = encontrarSubConjunto( ubicacion, desp );

        arrastrarDerecha( ubicacion, longitud, desp );
        establecerSubConjunto( ubicacion + longitud - desp, guardado );
    }

    return this;
}

/**
 * Asegura que el conjunto de bits es lo suficientemente grande para
 * contener bits de esta <code>longitud</code> y extenderla si no lo es.
 *
 * @param longitud número de bits del conjunto que se requiere contener
 */
protected void acasoExtendido( int longitud )
{
    if (longitud > lonBits)
    {
        int segmentos = (longitud - 1) / BITSENLONG + 1;

        if (bits == null || segmentos > bits.length)
        {
            long [] nuevosBits = new long[segmentos];

            if (bits != null)
```

```
            System.arraycopy( bits, 0, nuevosBits, 0, bits.length );
            bits = nuevosBits;
        }
        lonBits = longitud;
    }
}
```

****

```
// ListaSimple

package agjava.util;


import java.io.Serializable;
import java.lang.reflect.Array;
import agjava.util.ListIterator;
import agjava.util.Collection;
import agjava.util.Iterator;
/**
 * Una implementación simple de una lista y su clase iterador. Este
 * implementa un subconjunto de la interfaz List para simplificar la
 * portabilidad posteriormente. Este tiene la propiedad de mantener el
 * orden de los elementos
 *
 */
public class ListaSimple implements Collection, Serializable
{
    /**
     * Número de elementos
     */
    protected int numElems;

    protected static class Nodo
    {
        public Object item;
        public Nodo prox;
        public Nodo prev;
    }

    /**
     * Cabeza y cola de la lista
     */
    protected Nodo cabeza;
    protected Nodo cola;

    /**
     * Cabeza de la lista de los iteradores asociados
     */
    protected IteradorListaSimple primerIter;

    public ListaSimple()
    {
        cabeza = new Nodo();
        cola = new Nodo();
        cabeza.prev = cola.prox = null;
```

```
    primerIter = null;
    clear();
}

public ListaSimple( Collection otra )
{
    this();
    addAll( otra );
}

/**
 * Devuelve el número de elementos de la lista.
 */
public int size()
{
    return numElems;
}

/**
 * Devuelve true si esta lista no contiene elementos.
 */
public boolean isEmpty()
{
    return numElems == 0;
}

/**
 * Devuelve true si esta lista contiene al elemento especificado.
 *
 * @param aEncontrar elemento cuya presencia en esta lista será probada
 */
public boolean contains( Object aEncontrar )
{
    boolean estaEn = false;

    for (Nodo elem = cabeza.prox; elem != cola && !estaEn; elem = elem.prox)
    {
        if (aEncontrar.equals( elem.item ))
            estaEn = true;
    }

    return estaEn;
}

/**
 * Devuelve un Iterador sobre los elementos en esta lista.
 */
public Iterator iterator()
{
    return iteradorLista();
}

/**
 * Devuelve un ListIterator sobre los elementos en esta lista.
 */
public ListIterator iteradorLista()
```

```
{
   return iteradorLista( 0 );
}

/**
 * Devuelve un ListIterator sobre los elementos en esta lista
 * comenzando en la posición especificada en la lista
 */
public ListIterator iteradorLista( int índice )
{
   return new IteradorListaSimple( índice );
}

/**
 * Devuelve un arreglo que contiene todos los elementos en esta lista
 */
public Object [] toArray()
{
   Object [] valorDev = new Object[numElems];
   int índice = 0;

   for (Nodo elem = cabeza.prox; elem != cola; elem = elem.prox)
      valorDev[índice++] = elem.item;

   return valorDev;
}

/**
 * Devuelve un arreglo que contiene todos los elementos en esta lista,
 * cuyo tipo de tiempo de ejecución es el del arreglo especificado.
 *
 * @param a el arreglo dentro del cual los elementos de la lista serán
 *        almacenados, si este es suficientemente grande; de otra
 *        manera, un nuevo arreglo del mismo tipo en tiempo de
 *        ejecución es asignado para este propósito.
 * @return un arreglo conteniendo los elementos de la lista
 */
public Object [] toArray( Object [] a )
{
   if (a == null)
      return toArray();
   else
   {
      Object [] valorDev = a;
      Class claseDev = a.getClass().getComponentType();
      int índice = 0;

      for (Nodo elem = cabeza.prox; elem != cola; elem = elem.prox)
      {
         if (claseDev.isInstance( elem.item ))
         {
            if (índice >= valorDev.length)
            {
               valorDev = (Object []) Array.newInstance(
                  claseDev, numElems );
               System.arraycopy( a, 0, valorDev, 0, índice );
```

```
            }
            valorDev[índice++] = elem.item;
        }
    }

    return valorDev;
  }
}

/**
 * Añade el elemento especificado.
 *
 * @param item elemento a añadir.
 * @return true si la lista cambió como resultado de la llamada.
 */
public boolean add( Object item )
{
  add( numElems, item );
  return true;
}

/**
 * Añade el elemento especificado en el índice dado.
 *
 * @param índice basado en un índice cero del elemento a insertar antes.
 * @param item elemento a añadir.
 */
public void add( int índice, Object item )
{
  iteradoresImpuros();

  //lista JDK lanza una excepción; la normalizamos por ahora

  if (índice < 0)
    índice = 0;
  else if (índice > numElems)
    índice = numElems;

  Nodo insertarAntes;
  Nodo temp = new Nodo();

  temp.item = item;

  if (índice * 2 > numElems)
  {
    insertarAntes = cola;
    for (int i = numElems; i > índice; --i)
      insertarAntes = insertarAntes.prev;
  }
  else
  {
    insertarAntes = cabeza.prox;
    for (int i = 0; i < índice; ++i)
      insertarAntes = insertarAntes.prox;
  }
```

```
    temp.prox = insertarAntes;
    temp.prev = insertarAntes.prev;
    temp.prev.prox = temp;
    insertarAntes.prev = temp;

    ++numElems;
}

/**
 * Remueve una instancia simple del elemento especificado de esta
 * Colección, si este esta presente.
 *
 * @param o elemento a ser removido de esta Colección, si esta presente
 * @return true si la Lista cambió como resultado de la llamada.
 */
public boolean remove( Object o )
{
    boolean encontrado = false;

    for (Nodo elem=cabeza.prox; elem != cola & !encontrado; elem=elem.prox)
    {
        if (o == elem.item)
        {
            iteradoresImpuros();

            elem.prev.prox = elem.prox;
            elem.prox.prev = elem.prev;
            --numElems;
            encontrado = true;
        }
    }

    return encontrado;
}

/**
 * Devuelve true si la lista contiene todo de los elementos en la
 * Colección específica.
 */
public boolean containsAll( Collection c )
{
    ListaSimple temp = new ListaSimple( this );

    temp.removeAll( c );

    return numElems - temp.size() == c.size();
}

/**
 * Añade cada uno de los elementos en la Colección especificada al final
 * de esta lista.
 *
 * @param otro elementos a ser insetados dentro de esta lista.
 * @return true si la lista cambió como resultado de la llamada.
 */
public boolean addAll( Collection otro )
```

```
{
   Iterator iter = otro.iterator();

   while (iter.hasNext())
      add( iter.next() );

   return true;
}

/**
 * Remueve de la lista cada uno de los elementos que están contenidos
 * en la lista especificada.
 *
 * @param otro elementos a ser removidos de esta lista.
 * @return true si la lista cambió como resultado de la llamada.
 */
public boolean removeAll( Collection otro )
{
   Iterator iter = otro.iterator();
   int oldCount = numElems;

   while (iter.hasNext())
      remove( iter.next() );

   return oldCount != numElems;
}

/**
 * Retiene solo los elementos en esta lista que estan contenidos en
 * la collección especificada.
 *
 * @param c elementos a ser retenidos en esta lista.
 * @return true si la lista cambió como resultado de la llamada.
 */
public boolean retainAll( Collection c )
{
   Iterator iter = c.iterator();
   Object [] elements = toArray();

   while (iter.hasNext())
   {
      Object item = iter.next();

      for (int i = 0; i < elements.length; ++i)
      {
         if (item.equals( elements[i] ))
         {
            elements[i] = null;
            break;
         }
      }
   }

   // Borra todos los elementos que no corresponden

   int i = 0;
```

```
    for (Nodo node = cabeza.prox; node != cola; node = node.prox)
    {
        if (elements[i++] != null)
        {
            iteradoresImpuros();
            node.prev.prox = node.prox;
            node.prox.prev = node.prev;
            --numElems;
        }
    }

    return numElems != elements.length;
}

/**
 * Remueve cada uno de los elementos de esta lista.
 */
public void clear()
{
    iteradoresImpuros();

    numElems = 0;
    cabeza.prox = cola;
    cola.prev = cabeza;
}

public boolean equals( Object otro )
{
    if (otro instanceof ListaSimple)
    {
        ListaSimple list = (ListaSimple) otro;

        if (numElems != list.numElems)
            return false;
        else if (numElems == 0)
            return true;
        else
            return containsAll( list );
    }
    else
        return false;
}

public int hashCode()
{
    int hash = numElems;

    for (Nodo elem = cabeza.prox; elem != cola; elem = elem.prox)
        hash ^= elem.item.hashCode();

    return hash;
}

public String toString()
{
    StringBuffer buf = new StringBuffer();
```

```
    buf.append( "(" );

    for (Nodo elem = cabeza.prox; elem != cola; elem = elem.prox)
    {
       buf.append( elem.item.toString() );
       if (elem.prox != cola)
          buf.append( "," );
    }

    buf.append( ")" );

    return buf.toString();
}

protected void iteradoresImpuros()
{
    if (primerIter != null)
       primerIter.impuro( null );
}

/**
 * Una falla del iterador rápido para ListaSimple. Si cualquier
 * operación se ejecuta en la lista asociada que modifique la lista de
 * otra manera que no sea a través de este iterador, el iterador lanzará
 * una excepción en el próximo uso.
 */
protected class IteradorListaSimple implements ListIterator
{
    IteradorListaSimple prox;
    Nodo posicionActual;
    int índice;
    boolean adelante;

    IteradorListaSimple( int ind )
    {
       adelante = false;

       if (ind < 0)
          ind = 0;
       else if (ind > numElems)
          ind = numElems;

       if (ind * 2 > numElems)
       {
          for (posicionActual = cola, índice = numElems;
               posicionActual != cabeza && índice > ind;
               posicionActual = posicionActual.prev, --índice)
             continue;
       }
       else
       {
          for (posicionActual = cabeza.prox, índice = 0;
               posicionActual != cola && índice < ind;
               posicionActual = posicionActual.prox, ++índice)
             continue;
```

```
    }

    synchronized (ListaSimple.this)
    {
      prox = primerIter;
      primerIter = this;
    }
  }

  public void finalize()
  {
    // Desencadenar este iterador de tal manera que no pasemos más
    // tiempo en este

    synchronized (ListaSimple.this)
    {
      IteradorListaSimple prev = null;

      for (IteradorListaSimple iter = primerIter; iter != null;
          iter = iter.prox)
      {
        if (iter == this)
        {
          if (prev == null)
            primerIter = prox;
          else
            prev.prox = prox;
          break;
        }
      }
    }
  }

  /**
   * Inserta el elemento especificado en la Lista.
   */
  public void add( Object o )
  {
    comprobar();
    primerIter.impuro( this );

    Nodo nuevoNodo = new Nodo();

    nuevoNodo.item = o;
    nuevoNodo.prox = posicionActual;
    nuevoNodo.prev = posicionActual.prev;
    posicionActual.prev = nuevoNodo;
    nuevoNodo.prev.prox = nuevoNodo;
    ++numElems;
    posicionActual = nuevoNodo;
    adelante = false;
  }

  /**
   * Devuelve true si este iterador de lista tiene más elementos
   * cuando recorre la lista en la dirección hacia adelante.
```

```
 */
public boolean hasNext()
{
   comprobar();

   return (índice < numElems);
}

/**
 * Devuelve true si este iterador de lista tiene más elementos
 * cuando recorre la lista en la dirección contraria.
 */
public boolean hasPrevious()
{
   comprobar();

   return (índice != 0);
}

/**
 * Devuelve el próximo elemento en la Lista.
 */
public Object next()
{
   comprobar();

   Object valorDev = posicionActual.item;

   posicionActual = posicionActual.prox;
   ++índice;
   adelante = true;

   return valorDev;
}

/**
 * Devuelve el índice del elemento que será devuelto por una
 * llamada subsecuente a next.
 */
public int nextIndex()
{
   comprobar();

   return índice;
}

/**
 * Devuelve el elemento previo en la Lista.
 */
public Object previous()
{
   comprobar();

   posicionActual = posicionActual.prev;
   --índice;
   adelante = false;
```

```
    return posicionActual.item;
}

/**
 * Devuelve el índice del elemento que será devuelto por una
 * llamada subsecuente a previous
 */
public int previousIndex()
{
    comprobar();

    return índice - 1;
}

/**
 * Remueve de la Lista el último elemento que será devuelto por
 * next o previous.
 */
public void remove()
{
    comprobar();
    primerIter.impuro( this );

    Nodo toRemove;

    if (adelante)
    {
        toRemove = posicionActual.prev;
        --índice;
    }
    else
    {
        toRemove = posicionActual;
        posicionActual = posicionActual.prox;
    }

    toRemove.prox.prev = toRemove.prev;
    toRemove.prev.prox = toRemove.prox;
    --numElems;
}

/**
 * Reemplaza el último elemento devuelto por next o previous con
 * el elemento especificado.
 */
public void set( Object o )
{
    comprobar();
    primerIter.impuro( this );

    if (adelante)
        posicionActual.prev.item = o;
    else
        posicionActual.item = o;
}
```

```
    protected void comprobar()
    {
       if (índice < 0)
          throw new ConcurrentModificationException();
    }

    protected void impuro( IteradorListaSimple enUso )
    {
       synchronized (ListaSimple.this)
       {
          for (IteradorListaSimple iter = primerIter; iter != null;
               iter = iter.prox)
          {
             if (iter != enUso)
                iter.índice = -1;
          }
          primerIter = enUso;
          if (enUso != null)
             enUso.prox = null;
       }
    }
  }
}
```

****

```
// ListaOrdenada

package agjava.util;

import java.io.Serializable;
import agjava.util.Comparator;
import agjava.util.Collection;
import agjava.util.ListIterator;

/**
 * Extiende ListaSimple para manejar elementos en orden ascendente
 * (parcial) basado en el resultado de llamar a compareTo en el objeto o
 * al usar un Comparator provisto en la construcción.
 *
 */
public class ListaOrdenada extends ListaSimple implements Serializable
{
   Comparator comp;

   public ListaOrdenada()
   {
      comp = null;
   }

   public ListaOrdenada( Comparator comparator )
   {
      comp = comparator;
   }
```

```
public ListaOrdenada( Collection c )
{
   super( c );
}

public ListaOrdenada( Comparator comparator, Collection c )
{
   comp = comparator;
   addAll( c );
}

/**
 * Devuelve un ListIterator de los elementos en esta lista,
 * comenzando en la posición especificada en la lista.
 */
public ListIterator iteradorLista( int índice )
{
   return new IteradorListaOrdenada( índice );
}

/**
 * Añade el elemento especificado a la lista. Este será insertado
 * después del último elemento existente que tiene un valor más bajo
 * o igual.
 *
 * @param item elemento a añadir.
 * @return true si la lista cambió como resultado de la llamada.
 */
public boolean add( Object item )
{
   add( numElems, item );
   return true;
}

/**
 * Añade el elemento especificado en el índice dado. Si al añadir el
 * elemento en el índice dado resultaría que el ordenamiento parcial
 * de la lista sea destruida, el elemento emigra hacia arriba o abajo
 * de la lista como sea apropiado hasta que la ordenación sea asentada.
 *
 * @param índice índice basado en cero del elemento a insertar antes.
 * @param item elemento a añadir.
 */
public void add( int índice, Object item )
{
   iteradoresImpuros();

   // la lista JDK lanza una excepción; la normalizaremos por ahora

   if (índice < 0)
      índice = 0;
   else if (índice > numElems)
      índice = numElems;

   Nodo insertarAntes;
   Nodo temp = new Nodo();
```

```
    temp.item = item;

    // Encuentra aproximadamente donde queremos insertar el nuevo
    // elemento
    if (índice * 2 > numElems)
    {
        insertarAntes = cola;
        for (int i = numElems; i > índice; --i)
            insertarAntes = insertarAntes.prev;
    }
    else
    {
        insertarAntes = cabeza.prox;
        for (int i = 0; i < índice; ++i)
            insertarAntes = insertarAntes.prox;
    }

    // Luego aseguramos retener el orden parcial

    while (compare( item, insertarAntes.item ) > 0)
        insertarAntes = insertarAntes.prox;
    while (compare( item, insertarAntes.prev.item ) < 0)
        insertarAntes = insertarAntes.prev;

    temp.prox = insertarAntes;
    temp.prev = insertarAntes.prev;
    temp.prev.prox = temp;
    insertarAntes.prev = temp;

    ++numElems;
}

/**
 * Rutina interna para comparar los elementos de la lista. Devuelve
 * -1, 0 o 1 para indicar que el primer objeto es menor, igual a o
 * más grande que el segundo. Este devuelve 0 si el segundo objeto es
 * nulo para asegurar el comportamiento correcto cuando se inserta
 * al comienzo o fin de la lista.
 */
protected int compare( Object o1, Object o2 )
{
    if (o2 == null)
        return 0;
    else if (comp != null)
        return comp.compare( o1, o2 );
    else
        return ((Comparable) o1).compareTo( o2 );
}

/**
 * Una falla del iterador rápido para ListaOrdenada. Si cualquier
 * operación se ejecuta en la lista asociada que modifique la lista
 * de otra manera que no sea a través de este iterador, el iterador
 * lanzará una excepción en el próximo uso.
 */
```

```
protected class IteradorListaOrdenada extends IteradorListaSimple
{
   IteradorListaOrdenada( int ind )
   {
      super( ind );
   }

   /**
    * Inserta el elemento especificado en la lista, asegurando que
    * retendremos el orden parcial. Esto significa que si éste está
    * insertado en el lugar erróneo, éste no será el próximo
    * elemento a ser leído por el iterador.
    */
   public void add( Object item )
   {
      comprobar();
      primerIter.impuro( this );

      Nodo nuevoNodo = new Nodo();
      Nodo insertarAntes = posicionActual;
      boolean masBajo = false;

      while (compare( item, insertarAntes.item ) > 0)
         insertarAntes = insertarAntes.prox;
      while (compare( item, insertarAntes.prev.item ) < 0)
      {
         masBajo = true;
         insertarAntes = insertarAntes.prev;
      }

      nuevoNodo.item = item;
      nuevoNodo.prox = insertarAntes;
      nuevoNodo.prev = insertarAntes.prev;
      insertarAntes.prev = nuevoNodo;
      nuevoNodo.prev.prox = nuevoNodo;
      ++numElems;
      if (posicionActual == insertarAntes)
         posicionActual = nuevoNodo;
      if (masBajo)
         ++índice;
      adelante = false;
   }

   /**
    * Reemplaza el último elemento devuelto por next o previous con
    * el elemento especificado. Si el reemplazo no tiene el mismo
    * valor como el elemento reemplazado, este será reorganizado en
    * la lista.
    */
   public void set( Object item )
   {
      remove();
      add( item );
   }
}
}
```

```
*****************************************************************************
// ConcurrentModificationException

package agjava.util;

/**
 * Lanzado por el iterador para ListaSimple en el caso de modificación
 * concurrente. Un duplicado de la funcionalidad del JDK1.4, se proporciona
 * para simplificar la portabilidad posterior.
 */
public class ConcurrentModificationException extends RuntimeException
{
   public ConcurrentModificationException()
   {
   }

   public ConcurrentModificationException( String mensaje )
   {
      super( mensaje );
   }
}

*****************************************************************************

// Collection

package agjava.util;

/**
 * Un duplicado de la interfaz Collection del JDK1.4 para la portabilidad
 * posterior y la comprensión del uso de esta.
 *
 */
public interface Collection
{
   /**
    * Devuelve el número de elementos en esta colección.
    */
   public int size();

   /**
    * Devuelve true si la colección no tiene elementos
    */
   public boolean isEmpty();

   /**
    * Devuelve true si la colección contiene al elemento especificado.
    *
    * @param o elemento cuya presencia en esta colección se probará.
    */
   public boolean contains( Object o );

   /**
    * Devuelve un iterador sobre los elementos en esta colección.
    */
   public Iterator iterator();
```

```
/**
 * Devuelve un array que contiene todos los elementos en esta colección.
 */
public Object [] toArray();

/**
 * Devuelve un array que contiene todos los elementos en esta colección,
 * el tipo devuelto en tiempo de ejecución es el del array especificado.
 *
 * @param a el array en el cual los elementos de la colección serán
 *        almacenados, si este es adecuadamente grande;
 *        de otra manera, un nuevo array del mismo tipo en
 *        tiempo de ejecución es asignado para este propósito.
 *
 * @return  un array que contiene los elementos de la colección
 */
public Object [] toArray( Object [] a );

/**
 * Asegura que esta colleción contiene el elemento especificado.
 *
 * @param o elemento cuya presencia en esta colección será asegurada.
 * @return true si la colección cambio como resultado de la llamada.
 */
public boolean add( Object o );

/**
 * Remueve una instancia simple del elemento especificado de esta
 * colección, si está presente.
 *
 * @param o elemento a ser removido de esta colección, si está presente
 * @return true si la colección cambio como resultado de la llamada.
 */
public boolean remove( Object o );

/**
 * Devuelve true si esta colección contiene todos los elementos de la
 * colección especificada.
 */
public boolean containsAll( Collection c );

/**
 * Añade todos los elementos de la colección especificada a esta
 * colección.
 *
 * @param c elementos a insertarse en esta colección.
 * @return true si la colección cambio como resultado de la llamada.
 */
public boolean addAll( Collection c );

/**
 * Remueve de esta colección cada uno de los elementos que están
 * contenidos en la colección especificada.
 *
 * @param c elementos a ser removidos de esta colección.
```

```
     * @return true si la colección cambio como resultado de la llamada.
     */
    public boolean removeAll( Collection c );

    /**
     * Retiene solo los elementos en esta colección que están contenidos en
     * la colección especificada.
     *
     * @param c elementos a retenerse en esta colección.
     * @return true si la colección cambio como resultado de la llamada.
     */
    public boolean retainAll( Collection c );

    /**
     * Remueve todos los elementos de esta colección.
     */
    public void clear();

    /**
     * Compara el objeto especificado con esta colección para su igualdad.
     *
     * @param o Objeto a ser comparado para su igualdad con esta colección.
     * @return true si el Objeto especificado es igual a esta colección.
     */
    public boolean equals( Object o );

    /**
     * Devuelve el valor del código hash para esta colección.
     */
    public int hashCode();
}

*****************************************************************************
// Comparable

package agjava.util;

/**
 * Esta interfaz es implementada por las clases que pueden comparar con
 * otros objetos de su clase. Es identica a la interfaz Comparable del
 * JDK 1.4 para ayudar a la portabilidad en un tiempo posterior y la
 * comprensión del uso de esta.
 *
 */
public interface Comparable
{
    /**
     * Compara este Objeto con el objeto especificado por su ordenación.
     */
    public int compareTo( Object o );
}

*****************************************************************************
// Comparator

package agjava.util;
```

```
/**
 * Esta interfaz es implementada por las clases que pueden comparar
 * elementos. Es idéntica a la interfaz Comparator del JDK 1.2 para
 * ayudar a la portabilidad en una fecha posterior y la
 * comprensión del uso de esta.
 */
public interface Comparator
{
   /**
    * Compara estos dos argumentos por su ordenación.
    */
   public int compare( Object o1, Object o2 );


   /**
    * Indica si algún otro objeto es "igual a" este Comparator
    */
   public boolean equals(Object obj);

}

//////////////////////////////////////////////////////////////////////////////
// Iterator

package agjava.util;

/**
 * Esta interfaz es implementada por los iteradores sobre las colecciones.
 * Este es idéntico a la interfaz Iterator del JDK 1.4 para ayudar a la
 * portabilidad en una fecha posterior y la comprensión del uso de esta.
 */
public interface Iterator
{
   /**
    * Devuelve true si la iteración tiene más elementos.
    */
   public boolean hasNext();


   /**
    * Devuelve el próximo elemento en la iteración.
    */
   public Object next();


   /**
    * Remueve de la colección subyacente el último elemento devuelto por
    * el Iterador.
    */
   public void remove();
}

//////////////////////////////////////////////////////////////////////////////
// ListIterator
```

```
package agjava.util;

/**
 * Esta interfaz es implementada por los iteradores sobre las colecciones.
 * Este es idéntico a la interfaz Iterator del JDK 1.4 para ayudar a la
 * portabilidad en una fecha posterior.
 */
public interface ListIterator extends Iterator
{
   /**
    * Inserta el elemento especificado en la Lista.
    */
   public void add( Object o );

   /**
    * Devuelve true si este iterador de lista tiene más elementos cuando se
    * recorre la Lista en la dirección hacia adelante.
    */
   public boolean hasNext();

   /**
    * Devuelve true si este iterador de lista tiene más elementos cuando se
    * recorre la Lista en la dirección contraria.
    */
   public boolean hasPrevious();

   /**
    * Devuelve el proximo elemento en la Lista.
    */
   public Object next();

   /**
    * Devuelve el índice del elemento que será devuelto por una subsecuente
    * llamada a next
    */
   public int nextIndex();

   /**
    * Devuelve el elemento previo en la Lista.
    */
   public Object previous();

   /**
    * Devuelve el índice del elemento que será devuelto por una
    * llamada subsecuente a previous.
    */
   public int previousIndex();

   /**
    * Remueve de la Lista el último elemento que fue devuelto por next o
    * previous.
    */
   public void remove();

   /**
    * Reemplaza el último elemento devuelto por next o previous con el
```

```
     * elemento especificado.
     */
    public void set( Object o );
}
```

B. CODIFICACIÓN DEL PAQUETE AG DEL COMPONENTE AGJAVA

```
//////////////////////////////////////////////////////////////////////////
// Crom

package agjava.ag;

import java.io.Serializable;
import java.util.Random;
import agjava.util.BitSetExtendido;

/**
 * Una clase que representa un cromosoma y tiene varias formas apropiadas
 * para construir y manipular el cromosoma.
 */
public class Crom extends BitSetExtendido
   implements Serializable, Cloneable
{
   /**
    * Id único para el cromosoma
    */
   protected long esteId;

   /**
    * Origen del generador de id único
    */
   protected static long ultimoId = 0;

   /**
    * Constructor por defecto
    */
   public Crom()
   {
   }

   /**
    * Crea un cromosoma aleatoriamente con una longitud dada
    *
    * @param longitud longitud del cromosoma
    */
```

```
public Crom( int longitud )
{
   super( longitud, 0 );
   esteId = ++ultimoId;
}

/**
 * Crea un cromosoma a partir de un conjunto existente de bits
 *
 * @param conjBits conjunto de bits a utilizar
 */
public Crom( BitSetExtendido conjBits )
{
   super( conjBits );
   esteId = ++ultimoId;
}

/**
 * Crea un cromosoma al copiar otro
 *
 * @param padre cromosoma a copiar
 */
public Crom( Crom padre )
{
   this( (BitSetExtendido) padre );
}

/**
 * Crea un cromosoma con dos cromosomas padre, seleccionando
 * bits aleatoriamente de cada uno.
 *
 * @param padre1 mamá
 * @param padre2 papá
 */
public Crom( Crom padre1, Crom padre2 )
{
   this( padre1, padre2, 0.5 );
}

/**
 * Crea un cromosoma con dos cromosomas padres, seleccionando bits
 * de cada uno de acuerdo a una tasa de cruzar dada. Un valor
 * de 0.0 significa que todos los bits provienen del padre1, 1.0
 * significa que todos los bits provienen del padre2. El hijo tiene
 * siempre la misma longitud que el padre1.
 *
 * @param padre1 mamá
 * @param padre2 papá
 * @param tasaCru valor de la tasa de cruzar entre 0.0 and 1.0
 */
public Crom( Crom padre1, Crom padre2, double tasaCru )
{
   this( padre1 );

   if (tasaCru > 0)
      cruzar( padre2, tasaCru );
```

```
}

/**
 * Crea un cromosoma al mutar otro.
 *
 * @param padre cromosoma a copiar
 * @param tasaMut tasa de mutación. 0.0 = sin mutación, 1.0 es
 *  equivalente a invertir todo el cromosoma
 */
public Crom( Crom padre, double tasaMut )
{
   this( padre );

   if (tasaMut > 0.0)
      mutar( tasaMut );
}

/**
 * Copia el contenido de otro cromosoma
 *
 * @param otro el cromosoma a copiar
 * @return this
 */
public Crom copiar( Crom otro )
{
   super.copiar( (BitSetExtendido) otro );

   return this;
}

/**
 * Crea y retorna una copia de este cromosoma
 *
 * @return una copia de este cromosoma
 */
public Object clone()
{
   return new Crom( this );
}

/**
 * Muta el cromosoma con la probabilidad de cambio dada de cada bit.
 *
 * @param tasaMut probabilidad de cambio del bit de 0.0 a 1.0.
 * @return this
 */
Crom mutar( double tasaMut )
{
   for (int i = 0; i < lonBits; ++i)
   {
      if (tasaMut >= 1.0 || Math.random() < tasaMut)
         invertir( i );
   }

   return this;
}
```

```
/**
 * Cruzamiento uniforme. Desempeña un cruzamiento uniforme de este cromosoma
 * con otro con una probabilidad dada de seleccionar cada bit de el
 * otro cromosoma.
 *
 * @param otro otro cromosoma
 * @param tasaCruzamiento tasa de cruzamiento (0.0 <= tasa <= 1.0)
 * @return this
 */
Crom cruzar( Crom otro, double tasaCruzamiento )
{
   // Usa el más corto de los dos

   int limite = (lonBits < otro.lonBits) ? lonBits : otro.lonBits;

   for (int i = 0; i < limite; ++i)
   {
      if (tasaCruzamiento >= 1.0 || Math.random() < tasaCruzamiento)
         establecerBooleanEn( i, otro.obtener( i ) );
   }

   return this;
}

/**
 * Intercambia dos segmentos de bits dentro del cromosoma. Si los
 * segmentos se traslapan los valores contenidos se extraerán
 * correctamente, pero entonces el segmento1 será reescrito en la
 * posición del segmento2 después que el segmento2 sea escrito en la
 * posicion del segmento1.
 *
 * @param segmento1 ubicación del primer segmento
 * @param segmento2 ubicación del segundo segmento
 * @param longitud Longitud de los segmentos
 * @return this
 */
Crom transponerSegmento( int segmento1, int segmento2, int longitud )
{
   BitSetExtendido bs1 = encontrarSubConjunto( segmento1, longitud );
   BitSetExtendido bs2 = encontrarSubConjunto( segmento2, longitud );

   establecerSubConjunto( segmento1, bs2 );
   establecerSubConjunto( segmento2, bs1 );

   return this;
}

/**
 * Añade un segmento de bits al final. Los bits del nuevo segmento
 * son inicializados aleatoriamente.
 *
 * @param longitud Longitud del segmento a añadir
 * @return this
 */
Crom añadirSegmentoAlFinal( int longitud )
```

```
    {
        BitSetExtendido bs = new BitSetExtendido( longitud, 0 );

        insertarSubConjunto( -1, bs );
        return this;
    }

    /**
     * Remueve un segmento de bits. <code>longitud</code> bits en la
     * ubicación <code>ubicación</code> son removidos.
     *
     * @param ubicación Posición del segmento
     * @param longitud Longitud del segmento a remover
     * @return this
     */
    Crom eliminarSegmento( int ubicación, int longitud )
    {
        borrarSubConjunto( ubicación, longitud );
        return this;
    }

    /**
     * Duplica un segmento de bits. <code>longitud</code> bits en la
     * ubicación <code>ubicación</code> son duplicados y la copia es
     * insertada después del original en el cromosoma
     *
     * @param ubicación Posición del segmento
     * @param longitud longitud del segmento a duplicar
     * @return this
     */
    Crom duplicarSegmento( int ubicación, int longitud )
    {
        BitSetExtendido bs = encontrarSubConjunto( ubicación, longitud );

        insertarSubConjunto( ubicación, bs );
        return this;
    }

    public String toString()
    {
        return "[" + esteId + ":" + super.toString() + "]";
    }
}

*****************************************************************************
// ItemCrom

package agjava.ag;

import java.io.Serializable;
import agjava.util.Comparable;

/**
 * Un cromosoma y la valuación de su capacidad
 *
 */
```

```
public class ItemCrom implements Comparable, Serializable
{
   // Actualmente se necesitan capacidad y dobleCapacidad porque las
   // clases de número del JDK1.1 (Integer etc.) no implementan
   // Comparable. Esto puede omitirse cuando se trabaje con JDK1.2 o
   // superior.

   protected Comparable capacidad;
   protected double doubleCapacidad;
   protected Crom crom;

   public ItemCrom()
   {
   }

   public ItemCrom( ItemCrom otro )
   {
      // Copia profunda del cromosoma de tal manera que cuando se inserta
      // dentro de una nueva generación, su valor no se altera por ninguna
      // procreación en la que se involucre como parte de la  generación
      // antigua (eso afectaría la exactitud de la capacidad asociada).

      capacidad = otro.capacidad;
      doubleCapacidad = otro.doubleCapacidad;
      crom = new Crom( otro.crom );
   }

   public ItemCrom( double capac, Crom cromosoma )
   {
      capacidad = null;
      doubleCapacidad = capac;
      crom = cromosoma;
   }

   public ItemCrom( Comparable capac, Crom cromosoma )
   {
      capacidad = capac;
      doubleCapacidad = 0.0;
      crom = cromosoma;
   }

   public int compareTo( Object otro )
   {
      ItemCrom otroC = (ItemCrom) otro;

      if (capacidad == null)
      {
         if (doubleCapacidad > otroC.doubleCapacidad)
            return 1;
         else if (doubleCapacidad < otroC.doubleCapacidad)
            return -1;
         else
            return 0;
      }
      else
         return capacidad.compareTo( ((ItemCrom) otro).capacidad );
```

```
}

public boolean equals( Object otro )
{
   if (otro instanceof ItemCrom)
      return (compareTo( (ItemCrom) otro ) == 0) &&
         crom.equals( ((ItemCrom) otro).obtenerCrom() );
   else
      return false;
}

public int hashCode()
{
   if (capacidad == null)
      return (int) Double.doubleToLongBits( doubleCapacidad ) ^
         crom.hashCode();
   else
      return capacidad.hashCode() ^ crom.hashCode();
}

public void establecerCrom( Crom nuevoCrom )
{
   crom = nuevoCrom;
}

public Crom obtenerCrom()
{
   return crom;
}

public void establecerCapacidad( double capac )
{
   capacidad = null;
   doubleCapacidad = capac;
}

public void establecerCapacidad( Comparable capac )
{
   capacidad = capac;
}

public double obtenerDobleCapacidad()
{
   return doubleCapacidad;
}

public Comparable obtenerCapacidad()
{
   return capacidad;
}

public String toString()
{
   if (capacidad == null)
      return "[" + doubleCapacidad + "," + crom.toString() + "]";
   else
```

```
        return "[" + capacidad.toString() + "," + crom.toString() + "]";
    }
}

*****************************************************************************
// OpGen

package agjava.ag;

import agjava.util.AleatorioInt;

/**
 * Una clase que representa un operador genético que puede manipular un
 * cromosoma.
 *
 */
public class OpGen
{
    protected boolean binario;

    public OpGen( boolean esOpBinario )
    {
        binario = esOpBinario;
    }

    public Crom aplicar( Crom crom1 )
    {
        return crom1;
    }

    public Crom aplicar( Crom crom1, Crom crom2 )
    {
        return aplicar( crom1 );
    }

    public Crom aplicarEn( int índice, Crom crom1 )
    {
        return crom1;
    }

    public Crom aplicarEn( int índice, Crom crom1, Crom crom2 )
    {
        return aplicarEn( índice, crom1 );
    }

    /**
     * Si el operador es binario o no lo es.
     */
    public boolean esOpBinario()
    {
        return binario;
    }

    protected static int calcNumGenes( Crom crom, int longitudGene )
    {
        return (crom.tamaño() - 1) / longitudGene + 1;
```

```
    }

    protected static int seleccionarGene( Crom crom, int longitudGene )
    {
        int numGenes = calcNumGenes( crom, longitudGene );
        int seleccionado = AleatorioInt.proximo( numGenes );

        return seleccionado * longitudGene;
    }
}

*****************************************************************************

// OpCalcar

package agjava.ag;

import agjava.util.BitSetExtendido;

/**
 * Una clase que representa un operador genético binario que copia un
 * segmento de tamaño fijo (gene) de un cromosoma a otro.
 *
 */
public class OpCalcar extends OpGen
{
    /**
     * Longitud de cada gene
     */
    protected int lon;

    public OpCalcar( int longitudGene )
    {
        super( false );
        lon = longitudGene;
    }

    public Crom aplicar( Crom crom1 )
    {
        return aplicar( crom1, crom1 );
    }

    public Crom aplicarEn( int índice, Crom crom1 )
    {
        return aplicarEn( índice, crom1, crom1 );
    }

    public Crom aplicar( Crom crom1, Crom crom2 )
    {
        return aplicarEn( seleccionarGene( crom1, lon ), crom1, crom2 );
    }

    public Crom aplicarEn( int índice, Crom crom1, Crom crom2 )
    {
        BitSetExtendido supresor = crom2;
```

```
      if (supresor.tamaño() != lon)
         supresor = supresor.encontrarSubConjunto( 0, lon );

      índice -= índice % lon;

      return (Crom) crom1.establecerSubConjunto( índice, supresor );
   }
}

*****************************************************************************

// OpCambio

package agjava.ag;

import agjava.util.AleatorioInt;

/**
 * Una clase que representa un operador genético unario que al azar suma o
 * resta uno del valor de un gene aleatorio en un cromosoma y hace lo
 * opuesto al próximo gene en el cromosoma.
 *
 */
public class OpCambio extends OpGen
{
   /**
    * Longitud de cada gene
    */
   protected int lon;

   public OpCambio( int longitudGene )
   {
      super( false );
      lon = longitudGene;
   }

   public Crom aplicar( Crom crom1 )
   {
      return aplicarEn( AleatorioInt.proximo(
         calcNumGenes( crom1, lon ) - 1 ) * lon, crom1 );
   }

   public Crom aplicarEn( int índice, Crom crom1 )
   {
      int numGenes = calcNumGenes( crom1, lon );

      // Necesita al menos dos genes

      if (numGenes > 1)
      {
         índice -= índice % lon;

         if (índice <= crom1.tamaño() - (lon * 2))
         {
            long value1 = crom1.obtenerLongEn( índice, lon );
            long value2 = crom1.obtenerLongEn( índice + lon, lon );
```

```
            if (Math.random() < 0.5)
            {
               --value1;
               ++value2;
            }
            else
            {
               ++value2;
               --value1;
            }

            crom1.establecerLongEn( índice, lon, value1 );
            crom1.establecerLongEn( índice + lon, lon, value2 );
         }
      }

      return crom1;
   }

   public Crom aplicarEn( int índice, Crom crom1, Crom crom2 )
   {
      return aplicarEn( índice, crom1 );
   }
}
```

****

```
// OpCruzBits

package agjava.ag;

/**
 * Una clase que representa un operador genético binario que realiza un
 * cruzamiento aleatorio a nivel de bits entre dos cromosomas.
 *
 */
public class OpCruzBits extends OpGen
{
   protected double tasa;

   public OpCruzBits( double tasaCruz )
   {
      super( true );
      tasa = tasaCruz;
   }

   public Crom aplicar( Crom crom1, Crom crom2 )
   {
      return crom1.cruzar( crom2, tasa );
   }

   public Crom aplicarEn( int índice, Crom crom1, Crom crom2 )
   {
      return aplicar( crom1, crom2 );
   }
```

```
}

**********************************************************************

// OpCruzGene

package agjava.ag;

/**
 * Una clase que representa un operador genético binario que realiza un
 * cruzamiento de gene simple entre dos cromosomas.
 *
 */
public class OpCruzGene extends OpGen
{
   /**
    * Longitud de un gene.
    */
   protected int lon;

   public OpCruzGene( int longitudGene )
   {
      super( true );
      lon = longitudGene;
   }

   public Crom aplicar( Crom crom1, Crom crom2 )
   {
      Crom masCorto;

      if (crom1.tamaño() < crom2.tamaño())
         masCorto = crom1;
      else
         masCorto = crom2;

      return aplicarEn( seleccionarGene( masCorto, lon ), crom1, crom2 );
   }

   public Crom aplicarEn( int índice, Crom crom1, Crom crom2 )
   {
      índice -= índice % lon;

      return (Crom) crom1.intercambiarSubConj( índice, lon, crom2 );
   }
}

**********************************************************************

// OpCruzPt

package agjava.ag;

import agjava.util.AleatorioInt;

/**
 * Una clase que representa un operador genético binario que realiza un
```

```
 * cruzamiento de n puntos entre dos cromosomas. Los límites del gene no
 * se respetan.
 *
 */
public class OpCruzPt extends OpGen
{
   protected int [] ptsCruz;

   public OpCruzPt( int numPts )
   {
      super( true );

      // Dos elementos extra para manejar los ceros al inicio y la
      // longitud máxima al final para proporcionar puntos finales fijos
      // a los cruzamientos.

      ptsCruz = new int[((numPts >= 1) ? numPts : 1) + 2];
      ptsCruz[0] = 0;
   }

   public Crom aplicar( Crom crom1, Crom crom2 )
   {
      // Calcula la longitud mínima

      int lonMin = crom1.tamaño();

      if (crom2.tamaño() < lonMin)
         lonMin = crom2.tamaño();

      ptsCruz[ptsCruz.length-1] = lonMin - 1;

      // Genera los puntos de cruzamiento. Los cromosomas son divididos
      // más o menos regularmente

      int divLen = lonMin / ( ptsCruz.length - 1);

      for (int i = 1; i < ptsCruz.length - 1; ++i)
      {
         // Par evitar el hacer 2 puntos los mismos

         ptsCruz[i] = ptsCruz[i-1] + 1 +
            AleatorioInt.proximo( (i+1)*divLen - 1 - ptsCruz[i-1] );
      }

      // Hacer el cruzamiento

      for (int i = 1; i <= ptsCruz.length - 1; i += 2)
      {
         if (ptsCruz[i] < lonMin)
            crom1.intercambiarSubConj( ptsCruz[i-1],
               ptsCruz[i] - ptsCruz[i-1], crom2 );
      }

      return crom1;
   }
```

```
    public Crom aplicarEn( int índice, Crom crom1, Crom crom2 )
    {
        return aplicar( crom1, crom2 );
    }
}

*****************************************************************************

// OpDesp

package agjava.ag;

/**
 * Una clase que representa un operador genético unario que aleatoriamente
 * suma o resta uno del valor de un gene seleccionado en un cromosoma.
 *
 */
public class OpDesp extends OpGen
{
    /**
     * Longitud de cada gene
     */
    protected int lon;

    public OpDesp( int longitudGene )
    {
        super( false );
        lon = longitudGene;
    }

    public Crom aplicar( Crom crom1 )
    {
        return aplicarEn( seleccionarGene( crom1, lon ), crom1 );
    }

    public Crom aplicarEn( int índice, Crom crom1 )
    {
        índice -= índice % lon;

        if (índice <= crom1.tamaño() - lon)
        {
            long valor = crom1.obtenerLongEn( índice, lon );

            if (Math.random() < 0.5)
                --valor;
            else
                ++valor;

            crom1.establecerLongEn( índice, lon, valor );
        }

        return crom1;
    }
}
```

```
*****************************************************************************

// OpDup

package agjava.ag;

/**
 * Una clase que representa un operador genético unario que duplica un bit
 * de un gene al azar.
 *
 */
public class OpDup extends OpGen
{
   /**
    * La longitud de un gene.
    */
   protected int lon;

   public OpDup( int longitudGene )
   {
      super(false);
      lon = longitudGene;
   }

   public Crom aplicar( Crom crom1 )
   {
      // solo duplica en los límites del gene

      return crom1.duplicarSegmento( seleccionarGene( crom1, lon ), lon );
   }

   public Crom aplicarEn( int índice, Crom crom1 )
   {
      índice -= índice % lon;

      return crom1.duplicarSegmento( índice, lon );
   }
}

*****************************************************************************

// OpElim

package agjava.ag;

/**
 * Una clase que representa un operador genético unario que quita un
 * único gene al azar, aunque siempre deja por lo menos uno.
 *
 */
public class OpElim extends OpGen
{
   /**
    * Longitud de un gene
    */
```

```
    private int lon;

    public OpElim( int longitudGene )
    {
        super( false );
        lon = longitudGene;
    }

    public Crom aplicar( Crom crom1 )
    {
        return aplicarEn( seleccionarGene( crom1, lon ), crom1 );
    }

    public Crom aplicarEn( int índice, Crom crom1 )
    {
        int numGenes = calcNumGenes( crom1, lon );

        // No deja vacio completamente un cromosoma de genes.

        if (numGenes > 1)
        {
            índice -= índice % lon;

            crom1.borrarSubConjunto( índice, lon );
        }

        return crom1;
    }
}

*****************************************************************************

// OpIncAleat

package agjava.ag;

/**
 * Una clase que representa un operador genético unario que agrega un gene
 * aleatoriamente al final del cromosoma.
 *
 */
public class OpIncAleat extends OpGen
{
    /**
     * Longitud de un gene
     */
    protected int lon;

    public OpIncAleat( int longitudGene )
    {
        super( false );
        lon = longitudGene;
    }

    public Crom aplicar( Crom crom1 )
    {
```

```
        return crom1.añadirSegmentoAlFinal( lon );
    }

    public Crom aplicarEn( int índice, Crom crom1 )
    {
        return aplicar( crom1 );
    }
}
```

****

// OpMut

```
package agjava.ag;

/**
 * Una clase que representa un operador genético unario que muta un
 * cromosoma a una tasa dada.
 *
 */
public class OpMut extends OpGen
{
    protected double tasa;

    public OpMut( double tasaMut )
    {
        super( false );
        tasa = tasaMut;
    }

    public Crom aplicar( Crom crom1 )
    {
        return crom1.mutar( tasa );
    }

    public Crom aplicarEn( int índice, Crom crom1 )
    {
        return aplicar( crom1 );
    }
}
```

****

// OpTransp

```
package agjava.ag;

/**
 * Una clase que representa un operador genético unario que transpone
 * segmentos de tamaño fijo (genes) alrededor del interior de un cromosoma.
 *
 */
public class OpTransp extends OpGen {

    /**
```

```
     * Longitud de cada gene
     */
    protected int lon;

    public OpTransp( int longitudGene )
    {
        super( false );
        lon = longitudGene;
    }

    public Crom aplicar( Crom crom1 )
    {
        // Genera puntos de transposición aleatorios, pero solamente en los
        // límites del gene

        return crom1.transponerSegmento( seleccionarGene( crom1, lon ),
            seleccionarGene( crom1, lon ), lon );
    }

    public Crom aplicarEn( int índice, Crom crom1 )
    {
        // Genera puntos de transposición aleatorios, pero solamente en los
        // límites del gene

        índice -= índice % lon;
        return crom1.transponerSegmento( índice,
            seleccionarGene( crom1, lon ), lon );
    }
}
```

****

```
// ItemOp

package agjava.ag;

import java.io.Serializable;
import agjava.util.Comparable;

/**
 * Un operador genético y su peso (usados para predisponer la
 * selección entre operadores que compiten).
 *
 */
public class ItemOp implements Comparable, Serializable
{
    protected double peso;
    protected OpGen oper;

    public ItemOp()
    {
    }

    public ItemOp( double ponderacion, OpGen op )
    {
```

```
        peso = ponderacion;
        oper = op;
    }

    public int compareTo( Object otro )
    {
        ItemOp otroO = (ItemOp) otro;

        if (peso > otroO.peso)
            return 1;
        else if (peso < otroO.peso)
            return -1;
        else
            return 0;
    }

    public void establecerOp( OpGen opNuevo )
    {
        oper = opNuevo;
    }

    public OpGen obtenerOp()
    {
        return oper;
    }

    public void establecerPeso( double ponderacion )
    {
        peso = ponderacion;
    }

    public double obtenerPeso()
    {
        return peso;
    }
}
```

****

```
// Vista

package agjava.ag;

import java.io.Serializable;

/**
 * Las clases derivadas de Vista se usan para imponer cierta interpretación
 * en un cromosoma. La vista por defecto (tal como se provee por esta clase
 * base), simplemente trata el cromosoma como una cadena de bits de genes
 * simples, cada uno codificando un boolean. Interpretaciones más útiles
 * se pueden codificar por las subclases.
 *
 */
public class Vista implements Serializable
{
```

```
    public Vista()
    {
    }

    /**
     * Obtiene el valor del gene en <code>índice</code>. La implementación
     * por defecto simplemente considera el cromosoma como genes de n bits
     * simples y retorna el valor del bit en la ubicación dada codificada
     * como un Boolean. Se pretende que esta sea sobrescrita por las
     * subclases.
     */
    public Object obtenerGene( Crom crom, int índice )
    {
        return new Boolean( crom.obtener( índice ) );
    }

    /**
     * Retorna el número de genes. Por defecto es el mismo que el número
     * de bits.
     */
    public int tamaño( Crom crom )
    {
        return crom.tamaño();
    }

    /**
     * Devuelve una representación de cadena de un cromosoma tal como se
     * ve a través de esta vista.
     */
    public String toString( Crom crom )
    {
        return crom.toString();
    }
}

*****************************************************************************

// VistaFija

package agjava.ag;

/**
 * Una vista que interpreta cromosomas como una cadena de genes de tamaño fijo,
 * codificando los valores de punto flotante en un rango dado
 */
public class VistaFija extends Vista
{
    protected int tamGene;
    protected double inicio;
    protected double escala;

    /**
     * Define la vista, indicando el tamaño de cada gene. El valor
     * devuelto del gene será un Double en el rango de <code>inicio</code>
     * a <code>inicio + rango</code>.
```

```
     */
    public VistaFija( int tamGene, double inicio, double rango )
    {
        this.tamGene = tamGene;
        this.inicio = inicio;

        // Calcula el valor máximo para ese tamaño de gene y de esa deriva
        // un factor de escala que asegura que todos los valores devueltos
        // caen dentro del rango requerido.

        this.escala = rango / (double) ((1L << tamGene) - 1);
    }

    /**
     * Obtiene el valor del gene en <code>índice</code>. El resultado es
     * un Double que soporta el valor codificado por ese gene interpretado
     * para caer dentro del rango requerido.
     */
    public Object obtenerGene( Crom crom, int índice )
    {
            return new Double( crom.obtenerLongEn( tamGene * índice, tamGene ) *
          escala + inicio );
    }

    /**
     * Devuelve el número de genes.
     */
    public int tamaño( Crom crom )
    {
        return crom.tamaño() / tamGene;
    }

    /**
     * Devuelve una representación de cadena de un cromosoma como se
     * vería a través de esta vista.
     */
    public String toString( Crom crom )
    {
        StringBuffer buf = new StringBuffer();

        buf.append( obtenerGene( crom, 0 ) );
        for (int i = 1; i < tamaño( crom ); ++i)
        {
            buf.append( "," );
            buf.append( obtenerGene( crom, i ) );
        }
        return buf.toString();
    }
}

*****************************************************************************

// VistaInt

package agjava.ag;
```

```
/**
 * Una vista que interpreta los cromosomas como una
 * cadena de genes de tamaño fijo, codificando los
 * valores numéricos enteros.
 *
 */
public class VistaInt extends Vista
{
   protected int tamGene;
   protected boolean signado;
   protected long mascaraSigno;
   protected long mascaraExtendida;

   /**
    * Define la vista, indicando el tamaño de cada gene y si o no los
    * valores codificados en los genes deberían ser interpretados como
    * signados.
    */
   public VistaInt( int tamGene, boolean signado )
   {
      if (tamGene > 64)
         tamGene = 64;
      else if (tamGene < 1)
         tamGene = 1;

      this.tamGene = tamGene;
      this.signado = signado;

      if (signado)
      {
         mascaraSigno = 1 << (tamGene-1);
         mascaraExtendida = 0x8000000000000000L >> (64-tamGene);
      }
   }

   /**
    * Obtiene el valor del gene en <code>índice</code>. El resultado es
    * un Long que soporta el valor codificado por ese gene.
    */
   public Object obtenerGene( Crom crom, int índice )
   {
      long valorDev = crom.obtenerLongEn( tamGene * índice, tamGene );

      if (signado && (valorDev & mascaraSigno) != 0)
         valorDev |= mascaraExtendida;

       return new Long( valorDev );
   }

   /**
    * Devuelve el número de genes.
    */
   public int tamaño( Crom crom )
   {
      return crom.tamaño() / tamGene;
```

```
    }

    /**
     * Devuelve una representación de cadena de un cromosoma como se vería
     * a través de esta vista.
     */
    public String toString( Crom crom )
    {
        StringBuffer buf = new StringBuffer();

        buf.append( obtenerGene( crom, 0 ) );
        for (int i = 1; i < tamaño( crom ); ++i)
        {
            buf.append( "," );
            buf.append( obtenerGene( crom, i ) );
        }
        return buf.toString();
    }
}
```

****

```
// VistaVar

package agjava.ag;

/**
 * Una vista que interpreta un cromosoma como un cierto número de genes de
 * tamaño variable, cada uno de los cuales codifica un valor long
 * (posiblemente truncado).
 *
 */
public class VistaVar extends Vista
{
    protected int [] ptosLimiteGene;

    public VistaVar()
    {
    }

    /**
     * Inicializa una vista que interpreta los cromosomas como un cierto
     * numero de genes de tamaño variable, cada uno de los cuales codifica
     * un valor long (posiblemente truncado). limitesGene codifica las
     * posiciones de inicio de cada gene, ordenados en forma ascendente,
     * con el último elemento siendo la ubicación final del último gene + 1,
     * es decir, un array de tamaño n configura n-1 genes.
     */
    public VistaVar( int [] limitesGene )
    {
        ptosLimiteGene = new int[limitesGene.length];

        System.arraycopy( limitesGene, 0, ptosLimiteGene, 0,
                          limitesGene.length );
    }
```

```
    /**
     * Obtiene el valor del gene en <code>índice</code>. El resultado es
     * un Long que maneja el valor long codificado por ese gene, o null si
     * el índice esta fuera de rango.
     */
    public Object obtenerGene( Crom crom, int índice )
    {
     if ( índice >= 0 && índice < ptosLimiteGene.length - 1)
        return new Long( crom.obtenerLongEn( ptosLimiteGene[índice],
             ptosLimiteGene[índice+1] - ptosLimiteGene[índice] ) );
       else
        return null;
    }

    /**
     * Devuelve el número de genes.
     */
    public int tamaño( Crom crom )
    {
       return ptosLimiteGene.length - 1;
    }

    /**
     * Devuelve una representación de cadena de un cromosoma tal como se
     * ve a través de esta vista.
     */
    public String toString( Crom crom )
    {
       StringBuffer buf = new StringBuffer();

       buf.append( obtenerGene( crom, 0 ) );
       for (int i = 1; i < tamaño( crom ); ++i)
       {
          buf.append( "," );
          buf.append( obtenerGene( crom, i ) );
       }
       return buf.toString();
    }
}
```

****

// Poblacion

```
package agjava.ag;

import java.io.Serializable;
import agjava.util.ListIterator;
import agjava.util.AleatorioInt;
import agjava.util.ListaSimple;
import agjava.util.ListaOrdenada;

/**
 * Una clase básica para representar una población de cromosomas.
```

```
 *
 */
public class Poblacion extends ListaSimple implements Serializable
{
   /**
    * Porcentaje de la población llevado hacia adelante sin cambio
    * de cada generación.
    */
   protected double porcentajeElite;

   /**
    * Porcentaje estimado incapacitado para la reproducción
    */
   protected double porcentajeIncapaz;

   /**
    * Los operadores usados para modificar la generación actual cuando
    * se engendra la próxima.
    */
   protected ListaOrdenada listaOp;

   /**
    * Indice en nuestra lista del primer elemento que ha sido generado
    * de nuevo para esta generación antes que copiado del previo.
    */
   protected int indiceNuevo;

   /**
    * Cache para la eficiencia de los operadores en la lista de arriba,
    * junto con detalles de sus pesos para permitir que uno sea seleccionado.
    */
   protected ItemOp [] operadores;
   protected double [] pesoAcumulado;
   protected double pesoTotal;

   public Poblacion()
   {
   }

   public Poblacion( int tamPob, int tamGene, int minGenes,int maxGenes )
   {
      this( tamPob, tamGene, minGenes, maxGenes, 0 );
   }

   /**
    * Constructor. Si se da una tasa de mutación distinta de cero, los
    * operadores de mutación por defecto (en la tasa dada) y de
    * cruzamiento de dos puntos se asocian con la población automáticamente.
    */
   public Poblacion( int tamPob, int tamGene, int minGenes,
      int maxGenes, double tasaMut )
   {
      crearPob( tamPob, tamGene, minGenes, maxGenes );

      indiceNuevo = 0;
      operadores = null;
```

```
    pesoAcumulado = null;
    pesoTotal = 0.0;
    listaOp = new ListaOrdenada();

    // Crea los operadores genéticos por defecto, el de mutación y
    // el de cruzamiento de dos puntos

    if (tasaMut != 0.0)
    {
      OpGen mutador = new OpMut( tasaMut );
      OpGen cruzamiento = new OpCruzPt( 2 ); // cruzamiento de 2-pt

      // 1 es la "prioridad" por defecto

      listaOp.add( new ItemOp( 1, mutador ) );
      listaOp.add( new ItemOp( 1, cruzamiento ) );
    }
}

/**
 * Vaciar la población.
 */
public void clear()
{
    // Sobreescrito para restaurar los nuevos elementos.

    super.clear();
    indiceNuevo = 0;
}

/**
 * Devuelve un iterador sobre el total de la población.
 */
public ListIterator iteradorTodo()
{
    return iteradorLista();
}

/**
 * Devuelve un iterardor sobre la parte de la población que es
 * nueva para esta generación. Esto es, la parte cuya capacidad no
 * ha sido evaluado aún.
 */
public ListIterator iteradorSoloNuevos()
{
    return iteradorLista( indiceNuevo );
}

/**
 * Provee un operador modificador genético para usarlo en esta
 * población, y la ponderación que este tiene como probabilidad de ser
 * seleccionado.
 */
public void añadirOp( OpGen genOp, double peso )
{
    listaOp.add( new ItemOp( peso, genOp ) );
```

```
    operadores = null;
}

/**
 * Define la proporción de la población que se lleva hacia adelante
 * sin cambio entre generaciones.
 */
public void establecerTasaElitismo( double porc )
{
    if (porc < 0.0)
        porcentajeElite = 0.0;
    else if (porc > 1.0)
        porcentajeElite = 1.0;
    else
        porcentajeElite = porc;
}

/**
 * Define la propoción de la población estimada que se considera
 * incapacitado para la reproducción entre generaciones.
 */
public void establecerTasaIncapacidad( double porc )
{
    if (porc < 0.0)
        porcentajeIncapaz = 0.0;
    else if (porc > 1.0)
        porcentajeIncapaz = 1.0;
    else
        porcentajeIncapaz = porc;
}

/**
 * Una nueva generación se crea al llevar sobre una parte de la elite
 * sin cambio y luego generando el resto de la reproducción,
 * excluyendo lo peor de la generación actual de la reproducción a
 * una tasa definida por establecerTasaIncapacidad.
 */
public void nuevaGeneracion()
{
    // Crea la nueva población al copiar la elite y luego genera el
    // resto requerido.

    ListaOrdenada pobOrdenada = new ListaOrdenada( this );
    ItemCrom [] pobAntigua = new ItemCrom[pobOrdenada.size()];
    int pobTotal = pobAntigua.length;
    int elite = (int) (pobTotal * porcentajeElite);
    int incapacitado = (int) (pobTotal * porcentajeIncapaz);

    // System.out.println( "Población Total: " + pobTotal );

    pobAntigua = (ItemCrom []) pobOrdenada.toArray( pobAntigua );
    clear();

    // Cromosomas para la nueva generación se crean por duplicación
    // más que solamente siendo tranferido para asegurar que estos
    // permanecen sin cambio para el programa de reproducción llevado
```

```
      // a cabo después.

      for (int i = pobTotal - elite; i < pobTotal; ++i)
         add( new ItemCrom( pobAntigua[i] ) );
      indiceNuevo = elite;

      for (int i = elite; i < pobTotal; ++i)
      {
         Crom nuevoCrom = new Crom( seleccionarCromosoma( pobAntigua,
           incapacitado,pobTotal - incapacitado ) );
         OpGen operador = seleccionarOperador();

         // Aplica el operador genético al cromosoma elegido

         // System.out.println( "Modificando " + nuevoCrom + " con " +
         //                     operador );

         if (!operador.esOpBinario())
            operador.aplicar( nuevoCrom );
         else
         {
            // elegir otro gene aleatoriamente

            Crom pareja = new Crom( seleccionarCromosoma( pobAntigua,
               incapacitado,pobTotal - incapacitado ) );

            // System.out.println( "...y " + pareja );

            operador.aplicar( nuevoCrom, pareja );
         }

         // System.out.println( "Resultado: " + nuevoCrom );

         add( new ItemCrom( 0, nuevoCrom ) );
      }
   }

   /**
    * Elimina la población actual y crea una nueva
    */
   public void reset( int tamGene, int minGenes, int maxGenes )
   {
      int pobTotal = size();

      clear();
      crearPob( pobTotal, tamGene, minGenes, maxGenes );
   }

   /**
    * Crea nueva población
    */
   protected void crearPob( int tamPob, int tamGene, int minGenes,
      int maxGenes )
   {
      for (int i = 0; i < tamPob; ++i)
      {
```

```
      int lon = minGenes + AleatorioInt.proximo( maxGenes - minGenes + 1 );
      Crom nuevoCrom = new Crom( lon * tamGene );

      add( new ItemCrom( 0, nuevoCrom ) );
    }
}

/**
 * Selecciona aleatoriamente un cromosoma con quien reproducirse. La
 * rutina selecciona un cromosoma de <code>croms</code> desde dentro
 * del rango definido por <code>primero</code> y <code>longitud</code>.
 * Esta implementación simplemente selecciona aleatoriamente un
 * cromosoma valido, sin embargo la rutina pude ser sobreescrita por
 * las subclases, por ejemplo, para discriminar la selección de
 * acuerdo a la ponderación del cromosoma, varie el algoritmo de
 * acuerdo a la maduración de la población y así por el estilo.
 *
 * @param croms array de ElemCroms desde donde seelccionar el
 *          cromosoma. Se garantiza que el array esta ordenado
 *          desde el que tiene peor a mejor capacidad.
 * @param primero el primer elemento valido en el array
 * @param longitud el número de elementos validos en el array
 * @return un cromosma elegido de los elementos validos
 */
protected Crom seleccionarCromosoma( ItemCrom [] croms, int primero,
  int longitud )
{
  return croms[AleatorioInt.proximo( longitud ) + primero].obtenerCrom();
}

/**
 * Selecciona aleatoriamente un operador genético de aquellos
 * asociados con esta población.
 */
protected OpGen seleccionarOperador()
{
  if (operadores == null || operadores.length != listaOp.size())
  {
    // Crea el cache si este es actualmente invalido
    if (operadores == null)
      operadores = new ItemOp[listaOp.size()];
    operadores = (ItemOp []) listaOp.toArray( operadores );
    pesoAcumulado = new double[operadores.length];
    pesoTotal = 0;

    for (int i = 0; i < operadores.length; ++i)
    {
      double pesoOp = operadores[i].obtenerPeso();

      pesoAcumulado[i] = pesoOp + pesoTotal;
      pesoTotal += pesoOp;
    }
  }

  // genera la elección del operador aleatoriamente basandose en
  // los pesos de cada operador
```

```
        double pesoElegido = Math.random() * pesoTotal;
        int min = 0;
        int max = operadores.length - 1;
        int elegido = (int) ((pesoElegido / pesoTotal) * max);

        do
        {
           if (pesoElegido > pesoAcumulado[elegido])
           {
              min = elegido + 1;
              elegido += (max - elegido - 1)/2 + 1;
           }
           else if (elegido != 0 &&
              pesoElegido <= pesoAcumulado[elegido-1])
           {
              max = elegido - 1;
              elegido -= (elegido - min - 1)/2 + 1;
           }
           else break;
        }
        while (true);

        return operadores[elegido].obtenerOp();
     }
  }
```

C. CODIFICACIÓN DE CLASES DEL RESORTE

```
*****************************************************************************

//ResorteHelicoidalCompresion

/**
 * Representa un Resorte Helicoidal de Compresión con sus
 * atributos y métodos que permiten caracterizar el
 * resorte.
 */
package jgm.RHC;

import java.util.Arrays;

public class ResorteHelicoidalCompresion
{
    private int material = 2;
    private int tipo_servicio = 2;
    private int tipo_extremo = 4;
    private int sujecion = 1;

    private double G = 11.85E6; //Módulo del alambre del resorte
    private double Lf;          //Longitud libre
    private double OD;          //Diámetro exterior
    private double ID;          //Diámetro interior
    private double Dw;          //Díámetro del alambre
    private double Dm;          //Díámetro medio
    private double Fs;          //Fuerza en longitud comprimido
    private double Li;          //Longitud instalado;
    private double Fi;          //Fuerza en longitud instalado
    private double Ff;          //Fuerza en longitud libre
    private double k;           //Razón de resorte
    private double C;           //Indice de resorte
    private int N;            //Número total de bobinas
    private int Na;             //Número de bobinas activas
    private double Namax;       //Número máximo de bobinas activas
    private double p;           //Espaciamiento
    private double ae;          //Angulo de espaciamiento
    private double ODs;         //Diámetro exterior en longitud comprimido
    private double cc;          //Margen de bobina
    private double K;           //Factor de Wahl
```

```
private double Fo;          //Fuerza en longitud de operación
private double So;          //Esfuerzo o tensión en carga de operación
private double fo;          //Deflexión en longitud de operación
private double Lo;          //Longitud de operación
private double Ls;          //Longitud comprimido
private double Ss;          //Tensión o esfuerzo en longitud comprimido;
private double Sd;          //Tensión de diseño
private double Smax;        //Tensión máxima permisible
private double Do = 1E6; //Diámetro del orificio de instalación
private double Dv = 0;      //Diámetro de la varilla de instalación
private double RC;          //Razón Crítica
private double vol;         //Volúmen
private double pen;             //Penalización

private boolean C_OK = false;
private boolean OD_OK = false;
private boolean ID_OK = false;
private boolean No_Pandeo = false;
private boolean So_OK = false;
private boolean Ls_OK = false;
private boolean Ss_OK = false;
private boolean cc_OK = false;
private boolean Na_OK = false;

/**
 * Asigna el Tipo de Material
 * Para ASTM A227 Clase II   asignar 1
 * Para ASTM A228            asignar 2
 * Para ASTM A229 Clase II   asignar 3
 * Para ASTM A231            asignar 4
 * Para ASTM A401            asignar 5
 * Para ASTM A313 (A 302)    asignar 6
 * Para ASTM A304            asignar 7
 * Para ASTM A316            asignar 8
 */
public void asignarMaterial(int id_mat){

    if (id_mat >= 1 && id_mat <= 8)
        material = id_mat;
    else
        material = 2;

    asignarG(material);

    if (Dw > 0) {
        asignarDw(Dw);
        asignarSd(Dw,material,tipo_servicio);
        asignarSmax(Dw,material);
    }

}

/**
 * Devuelve la identificación del tipo de material.
 */
public int obtenerMaterial() {
```

```
    return material;
}

/**
 * Asigna el tipo de servicio del resorte.
 * Para Servicio Ligero    asignar 1
 *      Cargas estáticas hasta 10000 ciclos de carga, tasa baja de carga
 * Para Servicio Promedio  asignar 2
 *      Resorte para máquinarias, tasa moderada de carga y hasta 1000000 de
 *      ciclos
 * Para Servicio Severo    asignar 3
 *      Ciclaje rápido para más de un millón de ciclos; posibilidad de
 *      carga por impacto o choque
 */
public void asignarTipoServicio(int id_ser){
    if (id_ser >= 1 && id_ser <= 3)
        tipo_servicio = id_ser;
    else
        tipo_servicio = 2;

    if (Dw > 0) {
        asignarDw(Dw);
        asignarSd(Dw,material,tipo_servicio);
        asignarSmax(Dw,material);
    }
}

/**
 * Devuelve la identificación del tipo de servicio
 */
public int obtenerTipoServicio(){
    return tipo_servicio;
}


/**
 *  Asigna el tipo de extremo del resorte.
 *  Para extremos a escuadra y lijados asignar 1
 *  Para solo extremos a escuadra      asignar 2
 *  Para extremos en bruto y lijados   asignar 3
 *  Para extremos en bruto             asignar 4
 */
public void asignarTipoExtremo(int id_ext){
    if (id_ext >= 1 && id_ext <= 4)
        tipo_extremo = id_ext;
    else
        tipo_extremo = 4;
}

/**
 * Devuelve la identificación del tipo de extremo
 */
public int obtenerTipoExtremo() {
    return tipo_extremo;
}
```

```
/**
 * Asigna el tipo de sujeción en los extremos del resorte
 *  Para extremos fijos asignar                          1
 *  Para un extremo fijo y un extremo atornillado asignar 2
 *  Para ambos extremos atornillado asignar              3
 */
public void asignarSujecion(int id_suj){
    if (id_suj >= 1 && id_suj <= 3)
        sujecion = id_suj;
    else
        sujecion = 1;
}

/**
 * Devuelve la identificación del tipo de sujeción
 */
public int obtenerSujecion(){
    return sujecion;
}


//G =  Modulo de elasticidad del material
private void asignarG (int i){
    if (i==1)
        G = 11.5E6;
    else if (i==2)
        G = 11.85E6;
    else if (i==3)
        G = 11.2E6;
    else if (i==4)
        G = 11.2E6;
    else if (i==5)
        G = 11.2E6;
    else if (i==6)
        G = 10.0E6;
    else if (i==7)
        G = 11.2E6;
    else
        G = 11.2E6;
}

/**
 * Devuelve el módulo de elasticidad de acuerdo
 * al tipo de material asignado
 */
public double obtenerG(){
    return G;
}

/**
 * Asigna la fuerza en longitud de operación
 *
 * Fo = Fuerza en longitud de operación
 */
public void asignarFo(double fueOpe){
    Fo = fueOpe;
```

```
}

/**
 * Devuelve la fuerza en longitud de operación
 */
public double obtenerFo() {
    return Fo;
}

/**
 * Asigna la longitud de operación
 *
 * Lo = longitud de operación
 */
public void asignarLo(double lonOpe){
    Lo = lonOpe;
}

/**
 * Calcula la Longitud de operación en función de la
 * longitud libre y la deflexión
 */
private void calcularLo(){
    Lo = Lf - fo;
}

/**
 * Devuelve la Longitud libre
 */
public double obtenerLo() {
    return Lo;
}

/**
 * Asigna la Fuerza en longitud instalado
 *
 *  Fi = Fuerza en longitud de instalado
 */
public void asignarFi(double fueIns){
    Fi = fueIns;
}

/**
 * Devuelve la Fuerza en longitud instalado
 */
public double obtenerFi() {
    return Fi;
}

/**
 * Asigna la Longitud instalado
 *
 *  Li = Longitud de instalado
 */
public void asignarLi(double longIns){
```

```
        Li = longIns;
    }

    /**
     * Devuelve la Longitud instalado
     */
    public double obtenerLi() {
        return Li;
    }

    /**
     * Asigna el Diámetro del orificio donde
     * se instalará el resorte
     *
     * Do = Diámetro del orificio
     */
    public void asignarDo(double diaOri){
        Do = diaOri;
    }

    /**
     * Devuelve el Diámetro del Orificio
     */
    public double obtenerDo(){
        return Do;
    }

    /**
     * Asigna el Diámetro de la varilla donde
     * se instalará el resorte
     *
     * Dv = Diámetro de la varilla
     */
    public void asignarDv(double diaVar){
        Dv = diaVar;
    }

    /**
     * Devuelve el Diámetro de la varilla
     */
    public double obtenerDv(){
        return Dv;
    }

    //k = razón de resorte
    private void calculark(){
        k = (Fo-Fi)/(Li-Lo);
    }

    /**
     * Devuelve la razón del resorte
     *
     * k = razón de resorte
     */
    public double obtenerk(){
        return k;
```

```
}

//Lf = Longitud libre
private void calcularLf(){
    Lf = Li + Fi/k;
}

/**
 * Devuelve la Longitud libre
 *
 * Lf = Longitud libre
 */
public double obtenerLf() {
    return Lf;
}

//fo = deflexión en longitud de operación
private void calcularfo(){
    fo = Lf - Lo;
}

/**
 * Devuelve la deflexión en longitud de operación
 *
 * fo = deflexión en longitud de operación
 */
public double obtenerfo() {
    return fo;
}

/**
 * Asigna el Diámetro del alambre del resorte
 *
 * Dw = Diámetro del alambre
 */

public void asignarDw(double diaAla){

    if ( material == 1 )
    {
        Arrays.sort(DatosRHC.diaA227pulg);
        buscarAsignar(DatosRHC.diaA227pulg,diaAla);
    }
    else if (material == 2 )
    {
        Arrays.sort(DatosRHC.diaA228pulg);
        buscarAsignar(DatosRHC.diaA228pulg,diaAla);
    }
    else if (material == 3 )
    {
        Arrays.sort(DatosRHC.diaA229pulg);
        buscarAsignar(DatosRHC.diaA229pulg,diaAla);
    }
    else if (material == 4 )
    {
        Arrays.sort(DatosRHC.diaA231pulg);
```

```
            buscarAsignar(DatosRHC.diaA231pulg,diaAla);
        }
        else if (material == 5 )
        {
            Arrays.sort(DatosRHC.diaA401pulg);
            buscarAsignar(DatosRHC.diaA401pulg,diaAla);
        }
        else if (material == 6 )
        {
            Arrays.sort(DatosRHC.diaA313pulg);
            buscarAsignar(DatosRHC.diaA313pulg,diaAla);
        }
        else if (material == 7 )
        {
            Arrays.sort(DatosRHC.diaA304pulg);
            buscarAsignar(DatosRHC.diaA304pulg,diaAla);
        }
        else if (material == 8 )
        {
            Arrays.sort(DatosRHC.diaA316pulg);
            buscarAsignar(DatosRHC.diaA316pulg,diaAla);
        }

        asignarSd(Dw,material,tipo_servicio);
        asignarSmax(Dw,material);
    }

    private void buscarAsignar(double[] datos, double diaAla){

        int bajo = 0 ;
        int alto = datos.length - 1 ;
        int medio = ( bajo + alto + 1 ) / 2 ;
        int ubicacion = -1;

        do
        {

            if ( diaAla == datos[ medio ] )
                ubicacion = medio;
            else if ( diaAla < datos[ medio ] )
                alto = medio - 1 ;
            else
                bajo = medio + 1 ;

            medio = ( bajo + alto + 1 ) / 2 ;
      } while ( ( bajo <= alto ) && ( ubicacion == -1 ) );

        if (ubicacion == -1 )
        {
            if ( diaAla < datos[0] )
                Dw = datos[0];
            else if ( diaAla > datos[datos.length - 1] )
                Dw = datos[datos.length - 1];
            else
                Dw = datos[medio];
        }
```

```
    else
        Dw = datos[ubicacion];

}

/**
 * Devuelve el diámetro del alambre del resortee
 *
 */
public double obtenerDw(){
    return Dw;
}

//Sd = Tensión de diseño
private void asignarSd(double diaAla,int mat, int tipSer){
    if (mat==1 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA227pulgKsiLig,diaAla)*1000.0;
    else if (mat==1 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA227pulgKsiPro,diaAla)*1000.0;
    else if (mat==1 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA227pulgKsiSev,diaAla)*1000.0;
    else if (mat==2 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA228pulgKsiLig,diaAla)*1000.0;
    else if (mat==2 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA228pulgKsiPro,diaAla)*1000.0;
    else if (mat==2 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA228pulgKsiSev,diaAla)*1000.0;
    else if (mat==3 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA229pulgKsiLig,diaAla)*1000.0;
    else if (mat==3 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA229pulgKsiPro,diaAla)*1000.0;
    else if (mat==3 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA229pulgKsiSev,diaAla)*1000.0;
    else if (mat==4 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA231pulgKsiLig,diaAla)*1000.0;
    else if (mat==4 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA231pulgKsiPro,diaAla)*1000.0;
    else if (mat==4 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA231pulgKsiSev,diaAla)*1000.0;
    else if (mat==5 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA401pulgKsiLig,diaAla)*1000.0;
    else if (mat==5 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA401pulgKsiPro,diaAla)*1000.0;
    else if (mat==5 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA401pulgKsiSev,diaAla)*1000.0;
    else if (mat==6 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA313pulgKsiLig,diaAla)*1000.0;
    else if (mat==6 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA313pulgKsiPro,diaAla)*1000.0;
    else if (mat==6 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA313pulgKsiSev,diaAla)*1000.0;

    else if (mat==7 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA304pulgKsiLig,diaAla)*1000.0;
    else if (mat==7 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA304pulgKsiPro,diaAla)*1000.0;
```

```
    else if (mat==7 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA304pulgKsiSev,diaAla)*1000.0;
    else if (mat==8 && tipSer==1)
        Sd = lagrange(DatosRHC.datosA316pulgKsiLig,diaAla)*1000.0;
    else if (mat==8 && tipSer==2)
        Sd = lagrange(DatosRHC.datosA316pulgKsiPro,diaAla)*1000.0;
    else if (mat==8 && tipSer==3)
        Sd = lagrange(DatosRHC.datosA316pulgKsiSev,diaAla)*1000.0;


}

/**
 * Devuelve la tensión de diseño
 *
 */
public double obtenerSd(){
    return Sd;
}


//Smax = Tensión máxima permisible
private void asignarSmax(double diaAla,int mat){
    if (mat==1)
        Smax = lagrange(DatosRHC.datosA227pulgKsiLig,diaAla)*1000.0;
    else if (mat==2)
        Smax = lagrange(DatosRHC.datosA228pulgKsiLig,diaAla)*1000.0;
    else if (mat==3)
        Smax = lagrange(DatosRHC.datosA229pulgKsiLig,diaAla)*1000.0;
    else if (mat==4)
        Smax = lagrange(DatosRHC.datosA231pulgKsiLig,diaAla)*1000.0;
    else if (mat==5)
        Smax = lagrange(DatosRHC.datosA401pulgKsiLig,diaAla)*1000.0;
    else if (mat==6)
        Smax = lagrange(DatosRHC.datosA313pulgKsiLig,diaAla)*1000.0;
    else if (mat==7)
        Smax = lagrange(DatosRHC.datosA304pulgKsiLig,diaAla)*1000.0;
    else if (mat==8)
        Smax = lagrange(DatosRHC.datosA316pulgKsiLig,diaAla)*1000.0;

}

/**
 * Devuelve la Tensión Máxima permisible
 *
 */
public double obtenerSmax(){
    return Smax;
}


//Na = Número de bobinas activas
private void calcularNa(){
    if(tipo_extremo == 1 || tipo_extremo == 2)
        Namax = (Lo - 2*Dw)/Dw;
    else if (tipo_extremo == 3)
```

```
            Namax = (Lo - Dw)/Dw;
        else if (tipo_extremo == 4)
            Namax = Lo/Dw;

        Na = (int) Math.floor(Namax);

        if(Na > 1)
            Na_OK = true;
    }


    /**
     * Asigna el Número de bobinas activas
     *
     * Na = Número de bobinas activas
     */
    public void asignarNa(int valNa){
        Na = valNa;

        if(Na > 1)
            Na_OK = true;
    }

    /**
     * Devuelve el Número de bobinas activas
     *
     */
    public int obtenerNa(){
        return Na;
    }

    //C = Indice de resorte
    private void calcularC() {
        C = Math.cbrt(G*Dw/(8*k*Na));

        if ( C > 5)
            C_OK = true;
    }

    /**
     * Devuelve el Índice del resorte
     *
     * C = Indice de resorte
     */
    public double obtenerC() {
        return C;
    }

    //Dm = Diámetro medio
    private void calcularDm(){
        Dm = C*Dw;
    }

    /**
     * Asigna el Diámetro medio del resorte
     *
```

```
 * Dm = Diámetro medio
 */
public void asignarDm(double valDm){
    Dm = valDm;
}

/**
 * Devuelve el Diámetro medio del resorte
 *
 * Dm = Diámetro medio
 */
public double obtenerDm(){
    return Dm;
}

//OD = Diámetro exterior
private void calcularOD(){
    OD = Dm + Dw;

    if (Do > (OD + Dw/10))
        OD_OK = true;
}

/**
 * Devuelve el Diámetro exterior
 *
 * OD = Diámetro exterior
 */
public double obtenerOD(){
    return OD;
}

//ID = Diametro interior
private void calcularID(){
    ID = Dm - Dw;

    if(Dv < (ID - Dw/10))
        ID_OK = true;
}

/**
 * Devuelve el Diámetro interior
 *
 * ID = Diametro interior
 */
public double obtenerID(){
    return ID;
}


//RC = Razón Crítica
private void calcularRC(){
    if (sujecion == 1){
        if( Lf/Dm < DatosRHC.RCExtremosFijos[0][0]){
            RC = 1E100;
        }
```

```
            else {
                RC = lagrange(DatosRHC.RCExtremosFijos,Lf/Dm);
            }
        }
        else if (sujecion == 2){
            if( Lf/Dm < DatosRHC.RCUnExtremoFijo[0][0]){
                RC = 1E100;
            }
            else {
                RC = lagrange(DatosRHC.RCUnExtremoFijo,Lf/Dm);
            }
        }
        else if (sujecion == 3){
            if( Lf/Dm < DatosRHC.RCExtremosAtornillados[0][0]){
                RC = 1E100;
            }
            else {
                RC = lagrange(DatosRHC.RCExtremosAtornillados,Lf/Dm);
            }
        }
        if( fo/Lf <= RC )
            No_Pandeo = true;
    }

    /**
     * Devuelve la Razón Crítica del Resorte
     *
     * RC = Razón Crítica
     */
    public double obtenerRC(){
        return RC;
    }

    //K = Factor de Wahl
    private void calcularK(){
        K = (4*C - 1)/(4*C - 4) + 0.615/C;
    }

    /**
     * Devuelve el Factor de Wahl
     *
     * K = Factor de Wahl
     */
    public double obtenerK() {
        return K;
    }

    //So = Tensión de operación
    private void calcularSo(){
        So = 8*K*Fo*Dm/(Math.PI*Math.pow(Dw,3.0));

        if (So < Sd)
            So_OK = true;

    }
```

```
/**
 * Devuelve la Tensión de operación
 *
 * So = Tensión de operación
 */
public double obtenerSo(){
    return So;
}

//N = Número total de bobinas
private void calcularN(){
    if(tipo_extremo == 1 || tipo_extremo == 2)
        N = Na + 2;
    else if (tipo_extremo == 3)
        N = Na + 1;
    else if (tipo_extremo == 4)
        N = Na;
}

/**
 * Devuelve el Número total de bobinas
 *
 * N = Número total de bobinas
 */
public int obtenerN(){
    return N;
}


//Ls = longitud comprimido
private void calcularLs(){
    Ls = Dw*N;

    if(Ls < Lo)
        Ls_OK = true;
}

/**
 * Devuelve la longitud comprimido
 *
 * Ls = longitud comprimido
 */
public double obtenerLs(){
    return Ls;
}

//Fs = Fuerza en longitud comprimido
private void calcularFs(){
    Fs = k*(Lf - Ls);
}

/**
 * Devuelve la Fuerza en longitud comprimido
 *
 * Fs = Fuerza en longitud comprimido
 */
```

```
public double obtenerFs(){
    return Fs;
}

//Ss = Tensión en longitud comprimido
private void calcularSs(){
    Ss = So*Fs/Fo;

    if(Ss < Smax)
        Ss_OK = true;
}

/**
 * Devuelve la Tensión en longitud comprimido
 *
 * Ss = Tensión en longitud comprimido
 */
public double obtenerSs(){
    return Ss;
}

//cc = Margen de bobinas
private void calcularcc(){
    cc = (Lo - Ls)/Na;

    if(cc > Dw/10)
        cc_OK = true;
}

/**
 * Devuelve el Margen de bobinas
 *
 * cc = Margen de bobinas
 */
public double obtenercc(){
    return cc;
}

//p = Espaciamiento
private void calcularp(){
    if (tipo_extremo == 1)
        p = (Lf - 2*Dw)/Na;
    else if (tipo_extremo == 2)
        p = (Lf - 3*Dw)/Na;
    else if (tipo_extremo == 3)
        p = Lf/(Na+1);
    else if (tipo_extremo == 4)
        p = (Lf-Dw)/Na;
}

/**
 * Devuelve el Espaciamiento
 *
 * p = Espaciamiento
 */
public double obtenerp(){
```

```
    return p;
}

//ODs = Diámetro exterior en longitud comprimido
private void calcularODs(){
    ODs = Math.sqrt(Math.pow(Dm,2.0)+
          (Math.pow(p,2.0)-Math.pow(Dw,2.0))/Math.pow(Math.PI,2.0)) + Dw;
}

/**
 * Devuelve el Diámetro exterior en longitud comprimido
 *
 * ODs = Diámetro exterior en longitud comprimido
 */
public double obtenerODs(){
    return ODs;
}

//ae = angulo de espaciamiento
private void calcularae(){
    ae = Math.atan(p/(Math.PI*Dm));
}

/**
 * Devuelve el ángulo de espaciamiento
 *
 * ae = ángulo de espaciamiento
 */
public double obtenerae(){
    return ae;
}

//vol = volumen del resorte
private void calcularvol(){
    vol = 0.25*Math.pow(Math.PI,2.0)*Math.pow(Dw,2.0)*Dm*N;
}

/**
 * Devuelve el volumen del resorte
 *
 * vol = volumen del resorte
 */
public double obtenervol(){
    return vol;
}

//penalizacion
private void calcularpen(){
    if ( C_OK == false )
        pen = pen + 1E6;
    if ( OD_OK == false )
        pen = pen + 1E6;
    if ( ID_OK == false )
        pen = pen + 1E6;
    if ( No_Pandeo == false )
        pen = pen + 1E6;
```

```
    if ( So_OK == false )
        pen = pen + 1E6;
    if ( Ls_OK == false )
        pen = pen + 1E6;
    if ( Ss_OK == false )
        pen = pen + 1E6;
    if ( cc_OK == false )
        pen = pen + 1E6;
    if ( Na_OK == false )
        pen = pen + 1E6;
}


/**
 * Devuelve la penalización del resorte, por cada
 * restricción de diseño que no se cumpla se penaliza
 * 1E6
 *
 * pen = penalización del resorte
 */
public double obtenerpen(){
    return pen;
}


private double lagrange(double[][] datos,double xint){

    int u=0,t=0;
    boolean hallado=false;

    int n = datos.length;
    double l[] = new double[n];
    double prod1=1;
    double prod2=1;
    double y=0;

    for(u=0; u <= n-1; u++){
        if (datos[u][0] > xint)
        {
            hallado = true;
            if(u==0){
                u=0;
                t=u+2;
                break;
            }
            else if (u==n-1) {
                u=u-2;
                t=u+2;
                break;
            } else {
                u=u-1;
                t=u+2;
                break;
            }

        }
```

```
        }

        if (hallado == false){
            t=n-1;
            u=t-2;
        }

        //con tres puntos
        for (int j=u; j<=t; j++)
        {
            for (int i=u; i<=t; i++)
                if (i!=j){
                    prod1 = prod1*(xint-datos[i][0]);
                prod2 = prod2*(datos[j][0]-datos[i][0]);
                }
            l[j] = prod1/prod2;
            prod1 = 1;
            prod2 = 1;
            y = y + datos[j][1]*l[j];
        }
        return y;
    }

    /**
     * Calcula las otras características del resorte
     * cuando se tiene determinado:
     *
     *  Material
     *  Tipo de Extremo
     * Tipo de Servicio
     * Tipo de Sujeción en los Extremos
     * Longitud de Operación
     * Fuerza en Longitud de Operación
     * Longitud de Instalado
     * Fuerza en Longitud de Instalado
     * Diámetro del Alambre
     */
    public void analizar(){
        calculark();
        calcularLf();
        calcularfo();
        calcularNa();
        calcularC();
        calcularDm();
        calcularOD();
        calcularID();
        calcularRC();
        calcularK();
        calcularSo();
        calcularN();
        calcularLs();
        calcularFs();
        calcularSs();
        calcularcc();
        calcularvol();
        calcularp();
```

```
        calcularODs();
        calcularae();
        calcularpen();
    }

    /**
     * Calcula las otras características del resorte
     * cuando se tiene determinado:
     *
     *   Material
     *   Tipo de Extremo
     *  Tipo de Servicio
     *  Tipo de Sujeción en los Extremos
     *  Longitud de Operación
     *  Fuerza en Longitud de Operación
     *  Longitud de Instalado
     *  Fuerza en Longitud de Instalado
     *  Diámetro del Alambre
     *   Diámetro Medio
     *  Numero de Bobinas Activas
     *
     * Este método resulta útil en el caso de la
     * aplicación de los algoritmos genéticos
     */
    public void analizarAG(){
        calculark();
        calcularLf();
        calcularfo();
        calcularC();
        calcularOD();
        calcularID();
        calcularRC();
        calcularK();
        calcularSo();
        calcularN();
        calcularLs();
        calcularFs();
        calcularSs();
        calcularcc();
        calcularvol();
        calcularp();
        calcularODs();
        calcularae();
        calcularpen();
    }

    /**
     * Inicializa las variables que determinan
     * las restricicones de diseño en los
     * Resortes Helicoidales de Compresión
     * para hacer un nuevo análisis.
     */
    public void inicializar(){
        pen = 0.0;
        C_OK = false;
        OD_OK = false;
```

```
        ID_OK = false;
        No_Pandeo = false;
        So_OK = false;
        Ls_OK = false;
        Ss_OK = false;
        cc_OK = false;
        Na_OK = false;
    }

    /**
     * Devuelve true si el valor del
     * Índice de resorte está permitido
     */
    public boolean obtenerC_OK(){
        return C_OK;
    }

    /**
     * Devuelve true si el valor del
     * Diámetro externo está permitido
     */
    public boolean obtenerOD_OK(){
        return OD_OK;
    }

    /**
     * Devuelve true si el valor del
     * Diámetro interno está permitido
     */
    public boolean obtenerID_OK(){
        return ID_OK;
    }

    /**
     * Devuelve true si el resorte
     * no se pandeará
     */
    public boolean obtenerNo_Pandeo(){
        return No_Pandeo;
    }

    /**
     * Devuelve true si la Tensión de
     * Operación tiene un valor permitido
     */
    public boolean obtenerSo_OK(){
        return So_OK;
    }

    /**
     * Devuelve true si la Longitud de
     * Comprimido tiene un valor permitido
     */
    public boolean obtenerLs_OK(){
        return Ls_OK;
    }
```

```
    /**
     * Devuelve true si la Tensión de
     * Comprimido tiene un valor permitido
     */
    public boolean obtenerSs_OK(){
        return Ss_OK;
    }

    /**
     * Devuelve true si el Margen de las
     * Bobinas tiene un valor permitido
     */
    public boolean obtenercc_OK(){
        return cc_OK;
    }

    /**
     * Devuelve true si el Número de Bobinas
     * Activas tiene un valor permitido
     */
    public boolean obtenerNa_OK(){
        return Na_OK;
    }
}
```

****

```
package jgm.RHC;

public class DatosRHC {

    //Diámetros de alambres para resortes helicoidales de compresión

    protected static double diaA227pulg[] = {
    0.4900,0.4615,0.4305,0.3938,0.3625,0.3310,0.3065,0.2830,0.2625,0.2437,
    0.2253,0.2070,0.1920,0.1770,0.1620,0.1483,0.1350,0.1205,0.1055,0.0915,
    0.0800,0.0720,0.0625,0.0540,0.0475,0.0410,0.0348,0.0317,0.0286,0.0258,
    0.0230,0.0204 };

    protected static double diaA227mm[] = {
    13.0,12.0,11.0,10.0,9.0,8.5,8.0,7.0,6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,3.5,
    3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60,0.55,0.50,
    0.55,0.45,0.40 };

    protected static double diaA228pulg[] = {
    0.004,0.005,0.006,0.007,0.008,0.009,0.010,0.011,0.012,0.013,0.014,0.016,
    0.018,0.020,0.022,0.024,0.026,0.029,0.031,0.033,0.035,0.037,0.039,0.041,
    0.043,0.045,0.047,0.049,0.051,0.055,0.059,0.063,0.067,0.071,0.075,0.080,
    0.085,0.090,0.095,0.100,0.106,0.112,0.118,0.124,0.130,0.138};


    protected static double diaA228mm[] = {
    6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,3.5,
    3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60, 0.55,0.50,
```

```
0.55,0.45,0.40,0.35,0.30,0.28,0.25,0.22,0.20,0.18 };

protected static double diaA229pulg[] = {
0.4900,0.4615,0.4305,0.3938,0.3625,0.3310,0.3065,0.2830,0.2625,0.2437,
0.2253,0.2070,0.1920,0.1770,0.1620,0.1483,0.1350,0.1205,0.1055,0.0915,
0.0800,0.0720,0.0625,0.0540,0.0475,0.0410,0.0348,0.0317,0.0286,0.0258,
0.0230,0.0204 };

protected static double diaA229mm[] = {
13.0,12.0,11.0,10.0,9.0,8.5,8.0,7.0,6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,3.5,
3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60,0.55,0.50,
0.55,0.45,0.40 };

protected static double diaA231pulg[] = {
0.4900,0.4615,0.4305,0.3938,0.3625,0.3310,0.3065,0.2830,0.2625,0.2437,
0.2253,0.2070,0.1920,0.1770,0.1620,0.1483,0.1350,0.1205,0.1055,0.0915,
0.0800,0.0720,0.0625,0.0540,0.0475,0.0410,0.0348,0.0317,0.0286,0.0258,
0.0230,0.0204 };

protected static double diaA231mm[] = {
13.0,12.0,11.0,10.0,9.0,8.5,8.0,7.0,6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,3.5,
3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60,0.55,0.50,
0.55,0.45,0.40 };

protected static double diaA401pulg[] = {
0.4900,0.4615,0.4305,0.3938,0.3625,0.3310,0.3065,0.2830,0.2625,0.2437,
0.2253,0.2070,0.1920,0.1770,0.1620,0.1483,0.1350,0.1205,0.1055,0.0915,
0.0800,0.0720,0.0625,0.0540,0.0475,0.0410,0.0348,0.0317,0.0286,0.0258,
0.0230,0.0204 };

protected static double diaA401mm[] = {
13.0,12.0,11.0,10.0,9.0,8.5,8.0,7.0,6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,3.5,
3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60,0.55,0.50,
0.55,0.45,0.40 };

protected static double diaA313pulg[] = {
0.4900,0.4615,0.4305,0.3938,0.3625,0.3310,0.3065,0.2830,0.2625,0.2437,
0.2253,0.2070,0.1920,0.1770,0.1620,0.1483,0.1350,0.1205,0.1055,0.0915,
0.0800,0.0720,0.0625,0.0540,0.0475,0.0410,0.0348,0.0317,0.0286,0.0258,
0.0230,0.0204,0.0181,0.0173,0.0162,0.0150,0.0140,0.0132,0.0128,0.0118,
0.0104 };

protected static double diaA313mm[] = {
13.0,12.0,11.0,10.0,9.0,8.5,8.0,7.0,6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,
3.5,3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60,
0.55,0.50, 0.55,0.45,0.40,0.35,0.30,0.28,0.25,0.22,0.20,0.18 };

protected static double diaA304pulg[] = {
0.4900,0.4615,0.4305,0.3938,0.3625,0.3310,0.3065,0.2830,0.2625,0.2437,
0.2253,0.2070,0.1920,0.1770,0.1620,0.1483,0.1350,0.1205,0.1055,0.0915,
0.0800,0.0720,0.0625,0.0540,0.0475,0.0410,0.0348,0.0317,0.0286,0.0258,
0.0230,0.0204,0.0181,0.0173,0.0162,0.0150,0.0140,0.0132,0.0128,0.0118,
0.0104 };

protected static double diaA304mm[] = {
13.0,12.0,11.0,10.0,9.0,8.5,8.0,7.0,6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,
```

```
3.5,3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60,
0.55,0.50, 0.55,0.45,0.40,0.35,0.30,0.28,0.25,0.22,0.20,0.18 };

protected static double diaA316pulg[] = {
0.4900,0.4615,0.4305,0.3938,0.3625,0.3310,0.3065,0.2830,0.2625,0.2437,
0.2253,0.2070,0.1920,0.1770,0.1620,0.1483,0.1350,0.1205,0.1055,0.0915,
0.0800,0.0720,0.0625,0.0540,0.0475,0.0410,0.0348,0.0317,0.0286,0.0258,
0.0230,0.0204,0.0181,0.0173,0.0162,0.0150,0.0140,0.0132,0.0128,0.0118,
0.0104 };

protected static double diaA316mm[] = {
13.0,12.0,11.0,10.0,9.0,8.5,8.0,7.0,6.5,6.0,5.5,5.0,4.8,4.5,4.0,3.8,
3.5,3.0,2.8,2.5,2.0,1.8,1.6,1.4,1.2,1.0,0.90,0.80,0.70,0.65,0.60,
0.55,0.50, 0.55,0.45,0.40,0.35,0.30,0.28,0.25,0.22,0.20,0.18 };

// Tablas de tensiones de diseño para resortes helicoidales de
// compresión por material y según tipo de servicio

protected static double datosA227pulgKsiSev[][] = {
{0.020,120},{0.040,111},{0.060,105},{0.080,98},{0.100,95},{0.120,92},
{0.140,87.5},{0.160,85},{0.180,83},{0.200,82.5},{0.220,80},{0.240,78},
{0.260,77.5},{0.280,77},{0.300,75},{0.320,74.5},{0.340,74},{0.360,73.5},
{0.380,72.5},{0.400,72},{0.420,71.5},{0.440,70}
};

protected static double datosA227pulgKsiPro[][] = {
{0.020,142.5},{0.040,132},{0.060,124},{0.080,118},{0.100,113},
{0.120,109.5},{0.140,105.5},{0.160,103},{0.180,100},{0.200,97.5},
{0.220,96},{0.240,93.5},{0.260,92.5},{0.280,91},{0.300,89.5},
{0.320,88},{0.340,87},{0.360,86},{0.380,85},{0.400,84},{0.420,83},
{0.440,82.5},{0.460,82},{0.480,81.5},{0.500,81}
};

protected static double datosA227pulgKsiLig[][] = {
{0.020,160},{0.040,145},{0.060,138},{0.080,131},{0.100,125},{0.120,121},
{0.140,117},{0.160,114.5},{0.180,111.5},{0.200,108.5},{0.220,107},
{0.240,104.5},{0.260,102.5},{0.280,101.5},{0.300,100},{0.320,98},
{0.340,97},{0.360,96},{0.380,95},{0.400,94},{0.420,93},{0.440,92.5},
{0.460,92},{0.480,91.5},{0.500,91}
};

protected static double datosA228pulgKsiSev[][] = {
{0.005,160},{0.010,145},{0.020,131},{0.030,125},{0.040,119},
{0.050,113},{0.060,110},{0.070,107},{0.080,102},{0.090,101},{0.100,99},
{0.110,98},{0.120,97},{0.130,96},{0.140,95},{0.150,94},{0.160,93},
{0.170,92},{0.180,91},{0.190,90.5},{0.200,90},{0.210,89.5},{0.220,88.5},
{0.230,88},{0.240,87.5},{0.250,87}
};

protected static double datosA228pulgKsiPro[][] = {
{0.005,190},{0.010,172},{0.020,158},{0.030,147},{0.040,141},{0.050,135},
{0.060,131},{0.070,128},{0.080,125},{0.090,123},{0.100,121},{0.110,119.5},
{0.120,118},{0.130,117.5},{0.140,115},{0.150,113.5},{0.160,112},
{0.170,110},{0.180,109},{0.190,108},{0.200,107},{0.210,106},
{0.220,105},{0.230,104},{0.240,103},{0.250,102}
};
```

```
protected static double datosA228pulgKsiLig[][] = {
{0.005,220},{0.010,192},{0.020,174},{0.030,163},{0.040,157.5},{0.050,151},
{0.060,146},{0.070,142},{0.080,140},{0.090,138},{0.100,135},{0.110,132},
{0.120,130},{0.130,129},{0.140,128},{0.150,126.5},{0.160,125},
{0.170,122.5},{0.180,121.5},{0.190,120.5},{0.200,119.5},{0.210,118.5},
{0.220,117.5},{0.230,116.5},{0.240,115.5},{0.250,114.5}
};

protected static double datosA229pulgKsiSev[][] = {
{0.020,120},{0.040,112},{0.060,106},{0.080,100},{0.100,96},{0.120,92},
{0.140,89},{0.160,87},{0.180,84},{0.200,82.5},{0.220,81},{0.240,80},
{0.260,79},{0.280,78},{0.300,77.5},{0.320,77},{0.340,76},{0.360,75},
{0.380,74.5},{0.400,74.25},{0.420,74},{0.440,73.75},{0.460,73.5},
{0.480,73.25},{0.500,73}
};

protected static double datosA229pulgKsiPro[][] = {
{0.020,143},{0.040,134},{0.060,126},{0.080,119},{0.100,114},{0.120,110},
{0.140,107},{0.160,103},{0.180,100},{0.200,99},{0.220,97},{0.240,95},
{0.260,94},{0.280,93},{0.300,92.5},{0.320,91.5},{0.340,91},{0.360,90.5},
{0.380,90},{0.400,89.8},{0.420,89.5},{0.440,89.3},{0.460,89},{0.480,88},
{0.500,87.5}
};

protected static double datosA229pulgKsiLig[][] = {
{0.020,158},{0.040,148},{0.060,139},{0.080,133},{0.100,127},{0.120,123},
{0.140,118},{0.160,115},{0.180,112},{0.200,110},{0.220,108},{0.240,107},
{0.260,105},{0.280,104},{0.300,103},{0.320,102},{0.340,101.5},{0.360,100},
{0.380,99.5},{0.400,99.4},{0.420,99.3},{0.440,99.2},{0.460,99.1},
{0.480,99.15},{0.500,99}
};

protected static double datosA231pulgKsiSev[][] = {
{0.020,135},{0.040,127.5},{0.060,121},{0.080,115},{0.100,111},
{0.120,107.5},{0.140,103},{0.160,100},{0.180,97.5},{0.200,96.5},
{0.220,95},{0.240,94},{0.260,92.5},{0.280,91},{0.300,90},{0.320,89.5},
{0.340,89},{0.360,88.5},{0.380,88},{0.400,87.5},{0.420,87},{0.440,86.5},
{0.460,86},{0.480,85.5},{0.500,85}
};

protected static double datosA231pulgKsiPro[][] = {
{0.020,166},{0.040,157.5},{0.060,147.5},{0.080,141},{0.100,136},
{0.120,132},{0.140,127.5},{0.160,125},{0.180,122},{0.200,120},
{0.220,117.5},{0.240,116},{0.260,114},{0.280,112.5},{0.300,112},
{0.320,110.5},{0.340,110},{0.360,109.5},{0.380,109},{0.400,108.5},
{0.420,108},{0.440,107.5},{0.460,107},{0.480,106.5},{0.500,106}
};

protected static double datosA231pulgKsiLig[][] = {
{0.020,159},{0.040,147.5},{0.060,140},{0.080,133},{0.100,127.5},
{0.120,122.5},{0.140,117.5},{0.160,115},{0.180,112.5},{0.200,110},
{0.220,107.5},{0.240,106},{0.260,105},{0.280,103},{0.300,102.5},
{0.320,102},{0.340,101},{0.360,100},{0.380,99.8},{0.400,99.6},
{0.420,99.4},{0.440,99.2},{0.460,99},{0.480,98.8},{0.500,98.6}
};
```

```
protected static double datosA401pulgKsiSev[][] = {
{0.020,137.5},{0.040,134},{0.060,131},{0.080,127.5},{0.100,125},
{0.120,122.5},{0.140,120},{0.160,118},{0.180,117},{0.200,116},
{0.220,115},{0.240,113.5},{0.260,112.5},{0.280,112},{0.300,110},
{0.320,109},{0.340,108},{0.360,107.5},{0.380,107},{0.400,106},
{0.420,105.5},{0.440,105},{0.460,104.5},{0.480,104},{0.500,103.5}
};

protected static double datosA401pulgKsiPro[][] = {
{0.020,167.5},{0.040,164},{0.060,160},{0.080,158},{0.100,154},
{0.120,151},{0.140,148},{0.160,146},{0.180,143},{0.200,142.5},
{0.220,140},{0.240,138},{0.260,137.5},{0.280,136},{0.300,135},
{0.320,134},{0.340,132.5},{0.360,132},{0.380,131},{0.400,130},
{0.420,129.5},{0.440,129},{0.460,128.5},{0.480,128},{0.500,127.5}
};

protected static double datosA401pulgKsiLig[][] = {
{0.020,198},{0.040,194},{0.060,189},{0.080,185},{0.100,182},
{0.120,177.5},{0.140,175},{0.160,172.5},{0.180,169},{0.200,167.5},
{0.220,165},{0.240,163},{0.260,162},{0.280,160},{0.300,158},
{0.320,157.5},{0.340,156},{0.360,155},{0.380,154},{0.400,153},
{0.420,152},{0.440,151.5},{0.460,151},{0.480,150.5},{0.500,150}
};

protected static double datosA313pulgKsiSev[][]={
{0.012,120},{0.020,115},{0.040,105},{0.060,98},{0.080,92},{0.100,87},
{0.120,82.5},{0.140,79},{0.160,76},{0.180,73},{0.200,71.5},{0.220,69},
{0.240,67},{0.260,64},{0.280,62.5},{0.300,61},{0.320,59},{0.340,57},
{0.360,56},{0.380,55},{0.400,54},{0.420,52.5},{0.440,51},{0.457,41}
};


protected static double datosA313pulgKsiPro[][]={
{0.012,142.5},{0.020,137.5},{0.040,127},{0.060,119},{0.080,112},
{0.100,106},{0.120,101},{0.140,97},{0.160,92.5},{0.180,88},{0.200,85},
{0.220,82.5},{0.240,80},{0.260,77.5},{0.280,75},{0.300,73},{0.320,72},
{0.340,89},{0.360,67.5},{0.380,65},{0.400,62.5},{0.420,61.5},{0.440,60},
{0.460,58.5},{0.480,57.5},{0.500,56}
};

protected static double datosA313pulgKsiLig[][]={
{0.010,160},{0.020,152},{0.040,141},{0.060,132},{0.080,124},{0.100,117.5},
{0.120,112.5},{0.140,107.5},{0.160,103},{0.180,99},{0.200,95},{0.220,92.5},
{0.240,90},{0.260,87},{0.280,84},{0.300,82},{0.320,80},{0.340,77.5},
{0.360,74},{0.380,72.5},{0.400,71},{0.420,69.5},{0.440,67.5},{0.460,65},
{0.480,64},{0.500,63}
};

protected static double datosA304pulgKsiSev[][]={
{0.012,120*0.95},{0.020,115*0.95},{0.040,105*0.95},{0.060,98*0.95},
{0.080,92*0.95},{0.100,87*0.95},{0.120,82.5*0.95},{0.140,79*0.95},
{0.160,76*0.95},{0.180,73*0.95},{0.200,71.5*0.95},{0.220,69*0.95},
{0.240,67*0.95},{0.260,64*0.95},{0.280,62.5*0.95},{0.300,61*0.95},
{0.320,59*0.95},{0.340,57*0.95},{0.360,56*0.95},{0.380,55*0.95},
{0.400,54*0.95},{0.420,52.5*0.95},{0.440,51*0.95},{0.457,41*0.95}
```

```
};

protected static double datosA304pulgKsiPro[][]={
{0.012,142.5*0.95},{0.020,137.5*0.95},{0.040,127*0.95},{0.060,119*0.95},
{0.080,112*0.95},{0.100,106*0.95},{0.120,101*0.95},{0.140,97*0.95},
{0.160,92.5*0.95},{0.180,88*0.95},{0.200,85*0.95},{0.220,82.5*0.95},
{0.240,80*0.95},{0.260,77.5*0.95},{0.280,75*0.95},{0.300,73*0.95},
{0.320,72*0.95},{0.340,89*0.95},{0.360,67.5*0.95},{0.380,65*0.95},
{0.400,62.5*0.95},{0.420,61.5*0.95},{0.440,60*0.95},{0.460,58.5*0.95},
{0.480,57.5*0.95},{0.500,56*0.95}
};

protected static double datosA304pulgKsiLig[][]={
{0.010,160*0.95},{0.020,152*0.95},{0.040,141*0.95},{0.060,132*0.95},
{0.080,124*0.95},{0.100,117.5*0.95},{0.120,112.5*0.95},{0.140,107.5*0.95},
{0.160,103*0.95},{0.180,99*0.95},{0.200,95*0.95},{0.220,92.5*0.95},
{0.240,90*0.95},{0.260,87*0.95},{0.280,84*0.95},{0.300,82*0.95},
{0.320,80*0.95},{0.340,77.5*0.95},{0.360,74*0.95},{0.380,72.5*0.95},
{0.400,71*0.95},{0.420,69.5*0.95},{0.440,67.5*0.95},{0.460,65*0.95},
{0.480,64*0.95},{0.500,63*0.95}
};

protected static double datosA316pulgKsiSev[][]={
{0.012,120*0.85},{0.020,115*0.85},{0.040,105*0.85},{0.060,98*0.85},
{0.080,92*0.85},{0.100,87*0.85},{0.120,82.5*0.85},{0.140,79*0.85},
{0.160,76*0.85},{0.180,73*0.85},{0.200,71.5*0.85},{0.220,69*0.85},
{0.240,67*0.85},{0.260,64*0.85},{0.280,62.5*0.85},{0.300,61*0.85},
{0.320,59*0.85},{0.340,57*0.85},{0.360,56*0.85},{0.380,55*0.85},
{0.400,54*0.85},{0.420,52.5*0.85},{0.440,51*0.85},{0.457,41*0.85}
};

protected static double datosA316pulgKsiPro[][]={
{0.012,142.5*0.85},{0.020,137.5*0.85},{0.040,127*0.85},{0.060,119*0.85},
{0.080,112*0.85},{0.100,106*0.85},{0.120,101*0.85},{0.140,97*0.85},
{0.160,92.5*0.85},{0.180,88*0.85},{0.200,85*0.85},{0.220,82.5*0.85},
{0.240,80*0.85},{0.260,77.5*0.85},{0.280,75*0.85},{0.300,73*0.85},
{0.320,72*0.85},{0.340,89*0.85},{0.360,67.5*0.85},{0.380,65*0.85},
{0.400,62.5*0.85},{0.420,61.5*0.85},{0.440,60*0.85},{0.460,58.5*0.85},
{0.480,57.5*0.85},{0.500,56*0.85}
};

protected static double datosA316pulgKsiLig[][]={
{0.010,160*0.85},{0.020,152*0.85},{0.040,141*0.85},{0.060,132*0.85},
{0.080,124*0.85},{0.100,117.5*0.85},{0.120,112.5*0.85},{0.140,107.5*0.85},
{0.160,103*0.85},{0.180,99*0.85},{0.200,95*0.85},{0.220,92.5*0.85},
{0.240,90*0.85},{0.260,87*0.85},{0.280,84*0.85},{0.300,82*0.85},
{0.320,80*0.85},{0.340,77.5*0.85},{0.360,74*0.85},{0.380,72.5*0.85},
{0.400,71*0.85},{0.420,69.5*0.85},{0.440,67.5*0.85},{0.460,65*0.85},
{0.480,64*0.85},{0.500,63*0.85}
};

// Lf/Dm versus Razón crítica: deflexión/longitud libre, fo/Lf
// Se considera el tipo de sujeción en los extremos

protected static double RCExtremosFijos[][] = {
```

```
{5.25,0.75},{5.25,0.70},{5.255,0.65},{5.3,0.60},{5.4,0.55},{5.55,0.50},
{5.7,0.45},{5.9,0.40},{6.12,0.35},{6.55,0.30},{7.2,0.25},{8.1,0.20},
{9.15,0.15},{10,0.13}
};
protected static double RCUnExtremoFijo[][] = {
{3.75,0.75},{3.75,0.70},{3.76,0.65},{3.8,0.60},{3.9,0.55},{4.0,0.50},
{4.05,0.45},{4.20,0.40},{4.4,0.35},{4.6,0.30},{5,0.25},{5.7,0.20},
{6.7,0.15},{8.25,0.10},{10,0.007}
};
protected static double RCExtremosAtornillados[][] = {
{2.5,0.75},{2.5,0.70},{2.5,0.65},{2.55,0.60},{2.6,0.55},{2.7,0.50},
{2.75,0.45},{2.9,0.40},{3.1,0.35},{3.25,0.30},{3.6,0.25},{4,0.20},
{4.6,0.15},{5.45,0.10},{7.6,0.05}
};


}
```

D. CODIFICACIÓN DEL ALGORITMO DE OPTIMIZACIÓN

```
import agjava.ag.*;
import agjava.util.*;
import jgm.RHC.DatosRHC;
import jgm.RHC.ResorteHelicoidalCompresion;

class AGResHel{
    static final int numGenes = 3;
    static final int tamGenes = 13;
    static final int generaciones = 1000;
    static final int numCromosomas = 300;

    public static void main(String[] args){

        ResorteHelicoidalCompresion miRes = new ResorteHelicoidalCompresion();

        miRes.asignarMaterial(6);
        System.out.println("Tipo material = " + miRes.obtenerMaterial());
        System.out.println("G = " + miRes.obtenerG());

        miRes.asignarTipoExtremo(1);
        System.out.println("Tipo extremo = " + miRes.obtenerTipoExtremo());

        miRes.asignarTipoServicio(2);
        System.out.println("Tipo servicio:" + miRes.obtenerTipoServicio());

        miRes.asignarSujecion(1);
        System.out.println("Sujecion:" + miRes.obtenerSujecion());

        miRes.asignarFo(14);
        System.out.println("Fo:" + miRes.obtenerFo());

        miRes.asignarLo(1.25);
        System.out.println("Lo:" + miRes.obtenerLo());

        miRes.asignarFi(1.50);
        System.out.println("Fi:" + miRes.obtenerFi());

        miRes.asignarLi(2);
        System.out.println("Li:" + miRes.obtenerLi());

        //miRes.asignarDv(3);
```

```
Poblacion resortes = new Poblacion(numCromosomas,tamGenes,numGenes,
    numGenes, 0.3);

Vista visDw = new VistaFija(13,0.004,0.496);
Vista visDm = new VistaFija(13,0.2,49.99);
Vista visNa = new VistaInt(13,false);

resortes.establecerTasaElitismo(0.10);
resortes.establecerTasaIncapacidad(0.4);

for (Iterator censo = resortes.iterator();censo.hasNext(); )
    System.out.println(censo.next());

for (int i = 0; i < generaciones; ++i){

    Iterator censo;
    if(i % 10 == 0){
        censo = resortes.iteradorTodo();
    }
    else {
        censo = resortes.iteradorSoloNuevos();
    }

    while (censo.hasNext()){
        double val = 0.0;
        ItemCrom tmp = (ItemCrom) censo.next();
        Crom crom = tmp.obtenerCrom();

        miRes.inicializar();

        miRes.asignarDw(
            ((Double)visDw.obtenerGene(crom,0)).doubleValue() );
        miRes.asignarDm(
            ((Double)visDm.obtenerGene(crom,13)).doubleValue() );
        miRes.asignarNa(
            ((Long)visNa.obtenerGene(crom,26)).intValue() );

        miRes.analizarAG();

        val = - miRes.obtenervol() - miRes.obtenerpen();

        tmp.establecerCapacidad(val);
    }

    if ( i < generaciones - 1 )
        resortes.nuevaGeneracion();
}

ListaOrdenada resultados = new ListaOrdenada( resortes );
ListIterator iter = resultados.iteradorLista( resultados.size() );

while( iter.hasPrevious() ){
```

```
                ItemCrom item = (ItemCrom) iter.previous();
                Crom crom = item.obtenerCrom();

                System.out.print(visDw.obtenerGene(crom,0) + " ");
                System.out.print(visDm.obtenerGene(crom,13) + " ");
                System.out.print(visNa.obtenerGene(crom,26) + " ");
                System.out.println(" = " + item.obtenerDobleCapacidad() );
            }

        }

    }
```

E. APLICACIÓN DE OPTIMIZACIÓN CON INTERFAZ GRÁFICA

```
*****************************************************************************

package agrhc;

import java.awt.Toolkit;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import java.awt.Dimension;

public class AGRHC {
   boolean packFrame = false;

   /**
    * Construct and show the application.
    */
   public AGRHC() {
      FrameAGRHC frame = new FrameAGRHC();
      // Validate frames that have preset sizes
      // Pack frames that have useful preferred size info, e.g. from their layout
      if (packFrame) {
         frame.pack();
      } else {
         frame.validate();
      }

      // Center the window
      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = frame.getSize();
      if (frameSize.height > screenSize.height) {
         frameSize.height = screenSize.height;
      }
      if (frameSize.width > screenSize.width) {
         frameSize.width = screenSize.width;
      }
      frame.setLocation((screenSize.width - frameSize.width) / 2,
                        (screenSize.height - frameSize.height) / 2);
      frame.setVisible(true);
   }

   /**
    * Application entry point.
```

```
     *
     * @param args String[]
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    UIManager.setLookAndFeel(UIManager.
                                getSystemLookAndFeelClassName());
                } catch (Exception exception) {
                    exception.printStackTrace();
                }

                new AGRHC();
            }
        });
    }
}
```

****

```
package agrhc;

import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import java.awt.FlowLayout;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JOptionPane;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import agjava.ag.*;
import agjava.util.*;
import jgm.RHC.ResorteHelicoidalCompresion;
import jgm.RHC.DatosRHC;


public class FrameAGRHC extends JFrame {

    public ResorteHelicoidalCompresion resAG[] =
          new ResorteHelicoidalCompresion[300];

    public ResorteHelicoidalCompresion resLi =
          new ResorteHelicoidalCompresion();

    private String[] mat = { "ASTM A227 Clase II","ASTM A228","ASTM A229",
                    "ASTM A231","ASTM A401","ASTM A313 (A302)",
                    "ASTM A304","ASTM A316"};
    private String[] ext = {"Extremos a escuadra y lijados",
                    "Solo extremos a escuadra",
                    "Extremos en bruto y lijados",
```

```
                "Extremos en bruto"};
private String[] ser = {"Servicio Ligero","Servicio Promedio",
                "Servicio Severo"};
private String[] suj = {"Extremos fijos",
                "Un extremo fijo y un extremo atornillado",
                "Ambos extremos atornillados"};

JPanel contentPane;
BorderLayout borderLayout1 = new BorderLayout();
JTabbedPane jTabbedRHC = new JTabbedPane();
JPanel jPanelDisseno = new JPanel();
JPanel jPanelResultados = new JPanel();
JPanel jPanelLibre = new JPanel();
JPanel jPanelEntradas02 = new JPanel();
FlowLayout flowLayout1 = new FlowLayout();
JPanel jPanelEntradas01 = new JPanel();
FlowLayout flowLayout2 = new FlowLayout();
JLabel jLabelMaterial = new JLabel();
JComboBox jComboBoxMaterial = new JComboBox(mat);
BorderLayout borderLayout2 = new BorderLayout();
JLabel jLabelFo = new JLabel();
JTextField jTextFieldFo = new JTextField();
JLabel jLabelExtremo = new JLabel();
JComboBox jComboBoxExtremo = new JComboBox(ext);
JLabel jLabelServicio = new JLabel();
JComboBox jComboBoxServicio = new JComboBox(ser);
JLabel jLabelSujecion = new JLabel();
JComboBox jComboBoxSujecion = new JComboBox(suj);
JLabel jLabelLo = new JLabel();
JTextField jTextFieldLo = new JTextField();
JLabel jLabelFi = new JLabel();
JTextField jTextFieldFi = new JTextField();
JLabel jLabelLi = new JLabel();
JTextField jTextFieldLi = new JTextField();
JButton jButtonAplicar = new JButton();
JPanel jPanelSalida = new JPanel();
JScrollPane jScrollPaneSalida = new JScrollPane();
JTextArea jTextAreaSalida = new JTextArea();
BorderLayout borderLayout3 = new BorderLayout();
JPanel jPanel1 = new JPanel();
JLabel jLabel1 = new JLabel();
JTextField jTextFieldFo1 = new JTextField();
JLabel jLabel2 = new JLabel();
JTextField jTextFieldLo1 = new JTextField();
JLabel jLabel3 = new JLabel();
JTextField jTextFieldFi1 = new JTextField();
JLabel jLabel4 = new JLabel();
JTextField jTextFieldLi1 = new JTextField();
JPanel jPanel2 = new JPanel();
JScrollPane jScrollPane1 = new JScrollPane();
JTextArea jTextAreaLibre = new JTextArea();
JPanel jPanel3 = new JPanel();
JLabel jLabel5 = new JLabel();
JComboBox jComboBoxMaterial1 = new JComboBox(mat);
JLabel jLabel6 = new JLabel();
JComboBox jComboBoxExtremo1 = new JComboBox(ext);
```

```
    JLabel jLabel7 = new JLabel();
    JComboBox jComboBoxServicio1 = new JComboBox(ser);
    JLabel jLabel8 = new JLabel();
    JComboBox jComboBoxSujecion1 = new JComboBox(suj);
    BorderLayout borderLayout4 = new BorderLayout();
    JLabel jLabel9 = new JLabel();
    JTextField jTextFieldDw1 = new JTextField();
    JLabel jLabel10 = new JLabel();
    JTextField jTextFieldDm1 = new JTextField();
    JLabel jLabel11 = new JLabel();
    JTextField jTextFieldNa1 = new JTextField();
    JButton jButtonCalcular1 = new JButton();
    JLabel jLabelResultado = new JLabel();
    JSpinner jSpinnerAnalisis = new JSpinner();
    JPanel jPanel4 = new JPanel();
    FlowLayout flowLayout3 = new FlowLayout();
    JScrollPane jScrollPane2 = new JScrollPane();
    JTextArea jTextAreaAnalisis = new JTextArea();
    JButton jButtonAnalizar = new JButton();
    JLabel jLabel12 = new JLabel();
    JTextField jTextFieldDo = new JTextField();
    JLabel jLabel13 = new JLabel();
    JTextField jTextFieldDv = new JTextField();
    JLabel jLabel14 = new JLabel();
    JTextField jTextFieldDo1 = new JTextField();
    JLabel jLabel15 = new JLabel();
    JTextField jTextFieldDv1 = new JTextField();
    public FrameAGRHC() {
        try {
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            jbInit();
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }

    /**
     * Component initialization.
     *
     * @throws java.lang.Exception
     */
    private void jbInit() throws Exception {
        contentPane = (JPanel) getContentPane();
        contentPane.setLayout(borderLayout1);
        setSize(new Dimension(1000,600));
        setTitle(
                "Diseño de Resortes Helicoidales de Compresión con Algoritmos Genéti-
cos");
        jPanelEntradas02.setLayout(flowLayout1);
        jPanelEntradas02.setMaximumSize(new Dimension(490, 25));
        jPanelEntradas02.setMinimumSize(new Dimension(490, 25));
        jPanelEntradas02.setPreferredSize(new Dimension(490, 25));
        jPanelEntradas02.setSize(new Dimension(495,590));
        jPanelEntradas01.setLayout(flowLayout2);
        jPanelEntradas01.setMaximumSize(new Dimension(490, 25));
        jPanelEntradas01.setMinimumSize(new Dimension(490, 25));
```

```
jPanelEntradas01.setPreferredSize(new Dimension(490, 25));
jPanelEntradas01.setSize(new Dimension(495,590));
jLabelMaterial.setMaximumSize(new Dimension(200, 14));
jLabelMaterial.setMinimumSize(new Dimension(200, 14));
jLabelMaterial.setPreferredSize(new Dimension(200, 14));
jLabelMaterial.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelMaterial.setText("Tipo de Material");
jComboBoxMaterial.setMaximumSize(new Dimension(200, 22));
jComboBoxMaterial.setMinimumSize(new Dimension(200, 22));
jComboBoxMaterial.setPreferredSize(new Dimension(200, 22));
jComboBoxMaterial.setSelectedIndex(1);
jPanelDisseno.setLayout(borderLayout2);
jPanelDisseno.setMaximumSize(new Dimension(980, 550));
jPanelDisseno.setMinimumSize(new Dimension(980, 550));
jPanelDisseno.setPreferredSize(new Dimension(980, 550));
jLabelFo.setMaximumSize(new Dimension(250, 14));
jLabelFo.setMinimumSize(new Dimension(250, 14));
jLabelFo.setPreferredSize(new Dimension(250, 14));
jLabelFo.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelFo.setText("Fuerza en longitud de operación (Fo) (libras)");
jTextFieldFo.setMaximumSize(new Dimension(200, 19));
jTextFieldFo.setMinimumSize(new Dimension(200, 19));
jTextFieldFo.setPreferredSize(new Dimension(200, 19));
jLabelExtremo.setMaximumSize(new Dimension(200, 14));
jLabelExtremo.setMinimumSize(new Dimension(200, 14));
jLabelExtremo.setPreferredSize(new Dimension(200, 14));
jLabelExtremo.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelExtremo.setText("Tipo de Extremo");
jTabbedRHC.setMinimumSize(new Dimension(1090, 250));
jTabbedRHC.setPreferredSize(new Dimension(1090, 250));
jComboBoxExtremo.setMinimumSize(new Dimension(200, 22));
jComboBoxExtremo.setPreferredSize(new Dimension(200, 22));
jComboBoxExtremo.setSelectedIndex(3);
jLabelServicio.setMaximumSize(new Dimension(200, 14));
jLabelServicio.setMinimumSize(new Dimension(200, 14));
jLabelServicio.setPreferredSize(new Dimension(200, 14));
jLabelServicio.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelServicio.setText("Tipo de Servicio");
jComboBoxServicio.setMinimumSize(new Dimension(200, 22));
jComboBoxServicio.setPreferredSize(new Dimension(200, 22));
jComboBoxServicio.setSelectedIndex(1);
jLabelSujecion.setMaximumSize(new Dimension(200, 14));
jLabelSujecion.setMinimumSize(new Dimension(200, 14));
jLabelSujecion.setPreferredSize(new Dimension(200, 14));
jLabelSujecion.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelSujecion.setText("Tipo de Sujeción en los Extremos");
jComboBoxSujecion.setMinimumSize(new Dimension(200, 22));
jComboBoxSujecion.setPreferredSize(new Dimension(200, 22));
jLabelLo.setMaximumSize(new Dimension(250, 14));
jLabelLo.setMinimumSize(new Dimension(250, 14));
jLabelLo.setPreferredSize(new Dimension(250, 14));
jLabelLo.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelLo.setText("Longitud de operación (Lo) (pulgadas)");
jTextFieldLo.setMinimumSize(new Dimension(200, 19));
jTextFieldLo.setPreferredSize(new Dimension(200, 19));
jLabelFi.setMaximumSize(new Dimension(250, 14));
```

```
jLabelFi.setMinimumSize(new Dimension(250, 14));
jLabelFi.setPreferredSize(new Dimension(250, 14));
jLabelFi.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelFi.setText("Fuerza en longitud instalado (Fi)(libras)");
jTextFieldFi.setMinimumSize(new Dimension(200, 19));
jTextFieldFi.setPreferredSize(new Dimension(200, 19));
jLabelLi.setMaximumSize(new Dimension(250, 14));
jLabelLi.setMinimumSize(new Dimension(250, 14));
jLabelLi.setPreferredSize(new Dimension(250, 14));
jLabelLi.setHorizontalAlignment(SwingConstants.RIGHT);
jLabelLi.setText("Longitud instalado (Li)(pulgadas)");
jTextFieldLi.setMinimumSize(new Dimension(200, 19));
jTextFieldLi.setPreferredSize(new Dimension(200, 19));
jButtonAplicar.setText("Aplicar Algoritmos Genéticos");
jButtonAplicar.addActionListener(new
      FrameAGRHC_jButtonAplicar_actionAdapter(this));
jPanelSalida.setLayout(borderLayout3);
jTextAreaSalida.setMinimumSize(new Dimension(1000, 10000));
jTextAreaSalida.setPreferredSize(new Dimension(10000, 10000));
jScrollPaneSalida.setHorizontalScrollBarPolicy(JScrollPane.
                              HORIZONTAL_SCROLLBAR_ALWAYS);
jScrollPaneSalida.setVerticalScrollBarPolicy(JScrollPane.
                            VERTICAL_SCROLLBAR_ALWAYS);
jScrollPaneSalida.setMaximumSize(new Dimension(100000, 380));
jScrollPaneSalida.setMinimumSize(new Dimension(10000, 380));
jScrollPaneSalida.setPreferredSize(new Dimension(10000, 380));
jPanel1.setMaximumSize(new Dimension(490, 25));
jPanel1.setMinimumSize(new Dimension(490, 25));
jPanel1.setPreferredSize(new Dimension(490, 25));
jPanel1.setSize(new Dimension(490, 192));
jLabel1.setMaximumSize(new Dimension(250, 14));
jLabel1.setMinimumSize(new Dimension(250, 14));
jLabel1.setPreferredSize(new Dimension(250, 14));
jLabel1.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel1.setText("Fuerza en longitud de operación (Fo) (libras)");
jTextFieldFo1.setMaximumSize(new Dimension(200, 19));
jTextFieldFo1.setMinimumSize(new Dimension(200, 19));
jTextFieldFo1.setPreferredSize(new Dimension(200, 19));
jLabel2.setMaximumSize(new Dimension(250, 14));
jLabel2.setMinimumSize(new Dimension(250, 14));
jLabel2.setPreferredSize(new Dimension(250, 14));
jLabel2.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel2.setText("Longitud de operación (Lo) (pulgadas)");
jTextFieldLo1.setMinimumSize(new Dimension(200, 19));
jTextFieldLo1.setPreferredSize(new Dimension(200, 19));
jLabel3.setMaximumSize(new Dimension(250, 14));
jLabel3.setMinimumSize(new Dimension(250, 14));
jLabel3.setPreferredSize(new Dimension(250, 14));
jLabel3.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel3.setText("Fuerza en longitud instalado (Fi)(libras)");
jTextFieldFi1.setMinimumSize(new Dimension(200, 19));
jTextFieldFi1.setPreferredSize(new Dimension(200, 19));
jLabel4.setMaximumSize(new Dimension(250, 14));
jLabel4.setMinimumSize(new Dimension(250, 14));
jLabel4.setPreferredSize(new Dimension(250, 14));
jLabel4.setHorizontalAlignment(SwingConstants.RIGHT);
```

```
jLabel4.setText("Longitud instalado (Li)(pulgadas)");
jTextFieldLi1.setMinimumSize(new Dimension(200, 19));
jTextFieldLi1.setPreferredSize(new Dimension(200, 19));
jScrollPane1.setHorizontalScrollBarPolicy(JScrollPane.
                                HORIZONTAL_SCROLLBAR_ALWAYS);
jScrollPane1.setVerticalScrollBarPolicy(JScrollPane.
                              VERTICAL_SCROLLBAR_ALWAYS);
jScrollPane1.setMaximumSize(new Dimension(10000, 290));
jScrollPane1.setMinimumSize(new Dimension(990, 290));
jScrollPane1.setPreferredSize(new Dimension(990, 290));
jTextAreaLibre.setMinimumSize(new Dimension(10000, 10000));
jTextAreaLibre.setPreferredSize(new Dimension(10000, 100000));
jPanel3.setMaximumSize(new Dimension(490, 25));
jPanel3.setMinimumSize(new Dimension(490, 25));
jPanel3.setPreferredSize(new Dimension(490, 25));
jPanel3.setSize(new Dimension(490, 192));
jLabel5.setMaximumSize(new Dimension(200, 14));
jLabel5.setMinimumSize(new Dimension(200, 14));
jLabel5.setPreferredSize(new Dimension(200, 14));
jLabel5.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel5.setText("Tipo de Material");
jComboBoxMaterial1.setMaximumSize(new Dimension(200, 22));
jComboBoxMaterial1.setMinimumSize(new Dimension(200, 22));
jComboBoxMaterial1.setPreferredSize(new Dimension(200, 22));
jComboBoxMaterial1.setSelectedIndex(1);
jLabel6.setMaximumSize(new Dimension(200, 14));
jLabel6.setMinimumSize(new Dimension(200, 14));
jLabel6.setPreferredSize(new Dimension(200, 14));
jLabel6.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel6.setText("Tipo de Extremo");
jComboBoxExtremo1.setMinimumSize(new Dimension(200, 22));
jComboBoxExtremo1.setPreferredSize(new Dimension(200, 22));
jComboBoxExtremo1.setSelectedIndex(3);
jLabel7.setMaximumSize(new Dimension(200, 14));
jLabel7.setMinimumSize(new Dimension(200, 14));
jLabel7.setPreferredSize(new Dimension(200, 14));
jLabel7.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel7.setText("Tipo de Servicio");
jComboBoxServicio1.setMinimumSize(new Dimension(200, 22));
jComboBoxServicio1.setPreferredSize(new Dimension(200, 22));
jComboBoxServicio1.setSelectedIndex(1);
jLabel8.setMaximumSize(new Dimension(200, 14));
jLabel8.setMinimumSize(new Dimension(200, 14));
jLabel8.setPreferredSize(new Dimension(200, 14));
jLabel8.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel8.setText("Tipo de Sujeción en los Extremos");
jComboBoxSujecion1.setMinimumSize(new Dimension(200, 22));
jComboBoxSujecion1.setPreferredSize(new Dimension(200, 22));
jPanelLibre.setLayout(borderLayout4);
jLabel9.setMaximumSize(new Dimension(250, 14));
jLabel9.setMinimumSize(new Dimension(250, 14));
jLabel9.setPreferredSize(new Dimension(250, 14));
jLabel9.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel9.setText("Diámetro del alambre de resorte (Dw)(pulgadas)");
jTextFieldDw1.setMinimumSize(new Dimension(200, 19));
jTextFieldDw1.setPreferredSize(new Dimension(200, 19));
```

```
        jLabel10.setMaximumSize(new Dimension(250, 14));
        jLabel10.setMinimumSize(new Dimension(250, 14));
        jLabel10.setPreferredSize(new Dimension(250, 14));
        jLabel10.setHorizontalAlignment(SwingConstants.RIGHT);
        jLabel10.setText("Diámetro medio del resorte (Dm)(pulgadas)");
        jTextFieldDm1.setMinimumSize(new Dimension(200, 19));
        jTextFieldDm1.setPreferredSize(new Dimension(200, 19));
        jLabel11.setMaximumSize(new Dimension(250, 14));
        jLabel11.setMinimumSize(new Dimension(250, 14));
        jLabel11.setPreferredSize(new Dimension(250, 14));
        jLabel11.setHorizontalAlignment(SwingConstants.RIGHT);
        jLabel11.setText("Número de bobinas activas (Na)");
        jTextFieldNa1.setMinimumSize(new Dimension(200, 19));
        jTextFieldNa1.setPreferredSize(new Dimension(200, 19));
        jPanel2.setMinimumSize(new Dimension(990, 290));
        jPanel2.setPreferredSize(new Dimension(990, 290));
        jButtonCalcular1.setText("Calcular valores");
        jButtonCalcular1.addActionListener(new                                  Fra-
meAGRHC_jButton1_actionAdapter(this));
        jLabelResultado.setText("Elija la solución que desea analizar:");
        jSpinnerAnalisis.setMinimumSize(new Dimension(60, 18));
        jSpinnerAnalisis.setPreferredSize(new Dimension(60, 18));
        SpinnerNumberModel model = new SpinnerNumberModel(
          new Integer(1), // Dato visualizado al inicio en el spinner
          new Integer(1), // Límite inferior
          new Integer(300), // Límite superior
          new Integer(1) // incremento-decremento
        );
        jSpinnerAnalisis.setModel(model);
        jPanel4.setMinimumSize(new Dimension(990, 500));
        jPanel4.setPreferredSize(new Dimension(990, 500));
        jPanel4.setLayout(flowLayout3);
        jTextAreaAnalisis.setMinimumSize(new Dimension(1100, 2000));
        jTextAreaAnalisis.setPreferredSize(new Dimension(1100, 2000));
        jScrollPane2.setHorizontalScrollBarPolicy(JScrollPane.
                                    HORIZONTAL_SCROLLBAR_ALWAYS);
        jScrollPane2.setVerticalScrollBarPolicy(JScrollPane.
                                  VERTICAL_SCROLLBAR_ALWAYS);
        jScrollPane2.setMinimumSize(new Dimension(900, 490));
        jScrollPane2.setPreferredSize(new Dimension(900, 490));
        jButtonAnalizar.setText("Analizar");
        jButtonAnalizar.addActionListener(new
            FrameAGRHC_jButtonAnalizar_actionAdapter(this));
        jLabel12.setMaximumSize(new Dimension(250, 14));
        jLabel12.setMinimumSize(new Dimension(250, 14));
        jLabel12.setPreferredSize(new Dimension(250, 14));
        jLabel12.setHorizontalAlignment(SwingConstants.RIGHT);
        jLabel12.setText("Diámetro del orificio de instalación (Do)(pulgadas)");
        jTextFieldDo.setMinimumSize(new Dimension(200, 19));
        jTextFieldDo.setPreferredSize(new Dimension(200, 19));
        jTextFieldDo.setText("Sin orificio");
        jLabel13.setMaximumSize(new Dimension(250, 14));
        jLabel13.setMinimumSize(new Dimension(250, 14));
        jLabel13.setPreferredSize(new Dimension(250, 14));
        jLabel13.setHorizontalAlignment(SwingConstants.RIGHT);
        jLabel13.setText("Diámetro de la varilla de instalación (Dv)(pulgadas)");
```

```
jTextFieldDv.setMinimumSize(new Dimension(200, 19));
jTextFieldDv.setPreferredSize(new Dimension(200, 19));
jTextFieldDv.setText("Sin varilla");
jPanelSalida.setMinimumSize(new Dimension(990, 350));
jPanelSalida.setPreferredSize(new Dimension(990, 350));
jLabel14.setMaximumSize(new Dimension(250, 14));
jLabel14.setMinimumSize(new Dimension(250, 14));
jLabel14.setPreferredSize(new Dimension(250, 14));
jLabel14.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel14.setText("Diámetro del orificio de instalación (Do)(pulgadas)");
jTextFieldDo1.setMinimumSize(new Dimension(200, 19));
jTextFieldDo1.setPreferredSize(new Dimension(200, 19));
jTextFieldDo1.setText("Sin orificio");
jLabel15.setMaximumSize(new Dimension(250, 14));
jLabel15.setMinimumSize(new Dimension(250, 14));
jLabel15.setPreferredSize(new Dimension(250, 14));
jLabel15.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel15.setText("Diámetro de la varilla de instalación (Dv)(pulgadas)");
jTextFieldDv1.setMinimumSize(new Dimension(200, 19));
jTextFieldDv1.setPreferredSize(new Dimension(200, 19));
jTextFieldDv1.setText("Sin varilla");
jPanelEntradas02.add(jLabelMaterial);
jPanelEntradas02.add(jComboBoxMaterial);
jPanelEntradas02.add(jLabelExtremo);
jPanelEntradas02.add(jComboBoxExtremo);
jPanelEntradas02.add(jLabelServicio);
jPanelEntradas02.add(jComboBoxServicio);
jPanelEntradas02.add(jLabelSujecion);
jPanelEntradas02.add(jComboBoxSujecion);
jPanelDisseno.add(jPanelEntradas01, java.awt.BorderLayout.EAST);
jTabbedRHC.add(jPanelDisseno, "Diseño de Resortes");
jTabbedRHC.add(jPanelResultados, "Análisis de Resultados");
jPanelResultados.add(jLabelResultado);
jPanelResultados.add(jSpinnerAnalisis);
jPanelResultados.add(jButtonAnalizar);
jPanelResultados.add(jPanel4);
jPanel4.add(jScrollPane2);
jScrollPane2.getViewport().add(jTextAreaAnalisis);
jTabbedRHC.add(jPanelLibre, "Análisis Libre");
jPanel3.add(jLabel5);
jPanel3.add(jComboBoxMaterial1);
jPanel3.add(jLabel6);
jPanel3.add(jComboBoxExtremo1);
jPanel3.add(jLabel7);
jPanel3.add(jComboBoxServicio1);
jPanel3.add(jLabel8);
jPanel3.add(jComboBoxSujecion1);
jPanel2.add(jScrollPane1);
jScrollPane1.getViewport().add(jTextAreaLibre);
jPanel1.add(jLabel1);
jPanel1.add(jTextFieldFo1);
jPanel1.add(jLabel2);
jPanel1.add(jTextFieldLo1);
jPanel1.add(jLabel3);
jPanel1.add(jTextFieldFi1);
jPanel1.add(jLabel4);
```

```
        jPanel1.add(jTextFieldLi1);
        jPanel1.add(jLabel9);
        jPanel1.add(jTextFieldDw1);
        jPanel1.add(jLabel10);
        jPanel1.add(jTextFieldDm1);
        jPanel1.add(jLabel11);
        jPanel1.add(jTextFieldNa1);
        jPanel1.add(jLabel14);
        jPanel1.add(jTextFieldDo1);
        jPanel1.add(jLabel15);
        jPanel1.add(jTextFieldDv1);
        jPanel1.add(jButtonCalcular1);
        jPanelEntradas01.add(jLabelFo);
        jPanelEntradas01.add(jTextFieldFo);
        jPanelEntradas01.add(jLabelLo);
        jPanelEntradas01.add(jTextFieldLo);
        jPanelEntradas01.add(jLabelFi);
        jPanelEntradas01.add(jTextFieldFi);
        jPanelEntradas01.add(jLabelLi);
        jPanelEntradas01.add(jTextFieldLi);
        jPanelEntradas01.add(jLabel12);
        jPanelEntradas01.add(jTextFieldDo);
        jPanelEntradas01.add(jLabel13);
        jPanelEntradas01.add(jTextFieldDv);
        jPanelEntradas01.add(jButtonAplicar);
        jPanelDisseno.add(jPanelSalida, java.awt.BorderLayout.SOUTH);
        jPanelDisseno.add(jPanelEntradas02, java.awt.BorderLayout.WEST);
        contentPane.add(jTabbedRHC, java.awt.BorderLayout.CENTER);
        jPanelSalida.add(jScrollPaneSalida, java.awt.BorderLayout.CENTER);
        jScrollPaneSalida.getViewport().add(jTextAreaSalida);
        jPanelLibre.add(jPanel2, java.awt.BorderLayout.SOUTH);
        jPanelLibre.add(jPanel1, java.awt.BorderLayout.EAST);
        jPanelLibre.add(jPanel3, java.awt.BorderLayout.WEST);
    }

    public void jButton1_actionPerformed(ActionEvent e) {


        resLi.inicializar();

        int indMat = jComboBoxMaterial1.getSelectedIndex();
        resLi.asignarMaterial(indMat+1);

        int indExt = jComboBoxExtremo1.getSelectedIndex();
        resLi.asignarTipoExtremo(indExt+1);

        int indSer = jComboBoxServicio1.getSelectedIndex();
        resLi.asignarTipoServicio(indSer+1);

        int indSuj = jComboBoxSujecion1.getSelectedIndex();
        resLi.asignarSujecion(indSuj+1);

        double valFo = Double.parseDouble(jTextFieldFo1.getText());
        resLi.asignarFo(valFo);

        double valLo = Double.parseDouble(jTextFieldLo1.getText());
```

```
resLi.asignarLo(valLo);

double valFi = Double.parseDouble(jTextFieldFi1.getText());
resLi.asignarFi(valFi);

double valLi = Double.parseDouble(jTextFieldLi1.getText());
resLi.asignarLi(valLi);

double valDw = Double.parseDouble(jTextFieldDw1.getText());
resLi.asignarDw(valDw);

double valDm = Double.parseDouble(jTextFieldDm1.getText());
resLi.asignarDm(valDm);

int valNa = Integer.parseInt(jTextFieldNa1.getText());
resLi.asignarNa(valNa);

double valDo;
if(jTextFieldDo1.getText().compareTo("Sin orificio") != 0){
   valDo = Double.parseDouble(jTextFieldDo1.getText());
   resLi.asignarDo(valDo);
}
else valDo = 1E6;

double valDv;
if(jTextFieldDv1.getText().compareTo("Sin varilla") != 0){
   valDv = Double.parseDouble(jTextFieldDv1.getText());
   resLi.asignarDv(valDv);
}
else valDv = 0;

resLi.analizarAG();

String salida = "Análisis de la Solución \n\n";

int mat = resLi.obtenerMaterial();
if (mat == 1)
   salida = salida + "Material: ASTM A227 Clase II\n";
else if (mat == 2)
   salida = salida + "Material: ASTM A228\n";
else if (mat == 3)
   salida = salida + "Material: ASTM A229 Clase II\n";
else if (mat == 4)
   salida = salida + "Material: ASTM A231\n";
else if (mat == 5)
   salida = salida + "Material: ASTM A401\n";
else if (mat == 6)
   salida = salida + "Material: ASTM A313 (A302)\n";
else if (mat == 7)
   salida = salida + "Material: ASTM A304\n";
else salida = salida + "Material: ASTM A316\n";

salida = salida +
      "Módulo de elasticidad de alambre para resorte en corte (G)" +
      ": " + resLi.obtenerG() + " psi\n";
```

```
    int ext = resLi.obtenerTipoExtremo();
    if (ext == 1)
       salida = salida + "Extremos del resorte: " +
             "Extremos a escuadra y lijados\n";
    else if (ext == 2)
       salida = salida + "Extremos del resorte: " +
             "Solo extremos a escuadra\n";
    else if (ext == 3)
       salida = salida + "Extremos del resorte: " +
             "Extremos en bruto y lijados\n";
    else salida = salida + "Extremos del resorte: " +
              "Extremos en bruto\n";

    int ser = resLi.obtenerTipoServicio();
    if (ser == 1)
       salida = salida + "Tipo de servicio: " +
             "Servicio Ligero\n" +
             "Cargas estáticas hasta 10000 ciclos de carga, tasa baja de carga\n";
    else if (ser == 2)
       salida = salida + "Tipo de servicio: " +
             "Servicio Promedio\n" +
             "\tResorte para máquinarias, tasa moderada de carga y hasta 1000000 de
ciclos\n";
    else salida = salida + "Tipo de servicio: " +
              "Servicio Severo\n" +
              "\tCiclaje rápido para más de un millón de ciclos; posibilidad de carga por
impacto o choque\n";

    int suj = resLi.obtenerSujecion();
    if (suj == 1)
       salida = salida + "Tipo de sujeción en los extremos del resorte: " +
             "Extremos fijos\n";
    else if (suj == 2)
       salida = salida + "Tipo de sujeción en los extremos del resorte: " +
             "Un extremo fijo y un extremo atornillado\n";
    else salida = salida + "Tipo de sujeción en los extremos del resorte: " +
              "Ambos extremos atornillados\n";

    salida = salida + "Fuerza en longitud de operación (Fo): " +
          resLi.obtenerFo() + " libras\n";

    salida = salida + "Longitud de operación (Lo): " +
          resLi.obtenerLo() + " pulgadas\n";

    salida = salida + "Fuerza en longitud instalado (Fi): " +
          resLi.obtenerFi() + " libras\n";

    salida = salida + "Longitud instalado (Li): " +
          resLi.obtenerLi() + " pulgadas\n";

    salida = salida + "Diámetro del orificio de instalación (Do): ";
    if(resLi.obtenerDo() == 1E6)
       salida = salida + "Sin orificio\n";
    else
       salida = salida + resLi.obtenerDo() + " pulgadas\n";
```

```
salida = salida + "Diámetro de la varilla de instalación (Dv): ";
if(resLi.obtenerDv() == 0)
   salida = salida + "Sin varilla\n";
else
   salida = salida + resLi.obtenerDv() + " pulgadas\n";

salida = salida + "Razón de resorte (k): " +
       resLi.obtenerk() + "\n";

salida = salida + "Longitud libre (Lf): " +
       resLi.obtenerLf() + " pulgadas\n";

salida = salida + "Deflexión en longitud de operación (fo): " +
       resLi.obtenerfo() + " pulgadas\n";

salida = salida + "Diámetro del alambre del resorte (Dw): " +
       resLi.obtenerDw() + " pulgadas\n";

salida = salida + "Tensión de diseño (Sd): " +
       resLi.obtenerSd() + " psi\n";

 salida = salida + "Tensión máxima permisible (Smax): " +
       resLi.obtenerSmax() + " psi\n";

salida = salida + "Número de bobinas activas (Na): " +
       resLi.obtenerNa() + " \n";

salida = salida + "Índice de resorte (C): " +
       resLi.obtenerC() + " \n";

salida = salida + "Diámetro medio (Dm): " +
       resLi.obtenerDm() + " pulgadas\n";

salida = salida + "Diámetro exterior (OD): " +
       resLi.obtenerOD() + " pulgadas\n";

salida = salida + "Diámetro interior (ID): " +
       resLi.obtenerID() + " pulgadas\n";

salida = salida + "Razón Crítica (RC): " +
       resLi.obtenerRC() + " \n";

salida = salida + "Factor de Wahl (K): " +
       resLi.obtenerK() + " \n";

salida = salida + "Esfuerzo o tensión en carga de operación (So): " +
       resLi.obtenerSo() + " psi\n";

salida = salida + "Número total de bobinas (N): " +
       resLi.obtenerN() + " \n";

salida = salida + "Longitud comprimido (Ls): " +
       resLi.obtenerLs() + " pulgadas\n";

salida = salida + "Fuerza en longitud comprimido (Fs): " +
       resLi.obtenerFs() + " libras\n";
```

```
salida = salida + "Tensión o esfuerzo en longitud comprimido (Ss): " +
      resLi.obtenerSs() + " psi\n";

salida = salida + "Margen de bobina (cc): " +
      resLi.obtenercc() + " pulgadas\n";

salida = salida + "Volumen (vol): " +
      resLi.obtenervol() + " pulgadas cúbicas\n";

salida = salida + "\nPenalización acumulada (pen): " +
      resLi.obtenerpen() + " \n";

salida = salida + "Factor de Wahl: ";
if(resLi.obtenerC_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Índice de resorte: ";
if(resLi.obtenerC_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Diámetro exterior: ";
if(resLi.obtenerOD_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Diámetro interior: ";
if(resLi.obtenerID_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Pandeo: ";
if(resLi.obtenerNo_Pandeo() == true)
   salida = salida + "Sin pandeo\n";
else
   salida = salida + "Con pandeo (¡cuidado!)\n";

salida = salida + "Esfuerzo o tensión en carga de operación: ";
if(resLi.obtenerSo_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Longitud comprimido: ";
if(resLi.obtenerLs_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Tensión o esfuerzo en longitud comprimido: ";
```

```
    if(resLi.obtenerSs_OK() == true)
       salida = salida + "Correcto\n";
    else
       salida = salida + "Incorrecto\n";

    salida = salida + "Margen de bobina: ";
    if(resLi.obtenercc_OK() == true)
       salida = salida + "Correcto\n";
    else
       salida = salida + "Incorrecto\n";

    salida = salida + "Número de espiras activas: ";
    if(resLi.obtenerNa_OK() == true)
       salida = salida + "Correcto\n";
    else
       salida = salida + "Incorrecto\n";

    jTextAreaLibre.setText(salida);
}

public void jButtonAplicar_actionPerformed(ActionEvent e) {

    String salida="Sol.\tDiámetro del alambre\tDiámetro Medio\t"+
              "\tBobinas Activas\t  Volumen\n";

    int numGenes = 3;
    int tamGenes = 13;
    int generaciones = 1000;
    int numCromosomas = 300;

    ResorteHelicoidalCompresion miRes =
       new ResorteHelicoidalCompresion();

    int indMat = jComboBoxMaterial.getSelectedIndex();
    miRes.asignarMaterial(indMat+1);

    int indExt = jComboBoxExtremo.getSelectedIndex();
    miRes.asignarTipoExtremo(indExt+1);

    int indSer = jComboBoxServicio.getSelectedIndex();
    miRes.asignarTipoServicio(indSer+1);

    int indSuj = jComboBoxSujecion.getSelectedIndex();
    miRes.asignarSujecion(indSuj+1);

    double valFo = Double.parseDouble(jTextFieldFo.getText());
    miRes.asignarFo(valFo);

    double valLo = Double.parseDouble(jTextFieldLo.getText());
    miRes.asignarLo(valLo);

    double valFi = Double.parseDouble(jTextFieldFi.getText());
    miRes.asignarFi(valFi);

    double valLi = Double.parseDouble(jTextFieldLi.getText());
    miRes.asignarLi(valLi);
```

```
double valDo;
if(jTextFieldDo.getText().compareTo("Sin orificio") != 0){
   valDo = Double.parseDouble(jTextFieldDo.getText());
   miRes.asignarDo(valDo);
}
else valDo = 1E6;

double valDv;
if(jTextFieldDv.getText().compareTo("Sin varilla") != 0){
   valDv = Double.parseDouble(jTextFieldDv.getText());
   miRes.asignarDv(valDv);
}
else valDv = 0;


Poblacion resortes = new Poblacion(numCromosomas,tamGenes,numGenes,
                        numGenes, 0.3);

Vista visDw = new VistaFija(13,0.004,0.496);
Vista visDm = new VistaFija(13,0.2,49.99);
Vista visNa = new VistaInt(13,false);

resortes.establecerTasaElitismo(0.10);
resortes.establecerTasaIncapacidad(0.4);

for (int i = 0; i < generaciones; ++i){
   Iterator censo;
   if(i % 10 == 0){
      censo = resortes.iteradorTodo();
   }
   else{
      censo = resortes.iteradorSoloNuevos();
   }

   while (censo.hasNext()){
      double val = 0.0;
      ItemCrom tmp = (ItemCrom) censo.next();
      Crom crom = tmp.obtenerCrom();

      miRes.inicializar();

      miRes.asignarDw(((Double)visDw.obtenerGene(crom,0)).doubleValue());
      miRes.asignarDm(((Double)visDm.obtenerGene(crom,13)).doubleValue());
      miRes.asignarNa(((Long)visNa.obtenerGene(crom,26)).intValue());

      miRes.analizarAG();

      val = - miRes.obtenervol() - miRes.obtenerpen();

      tmp.establecerCapacidad(val);
   }

   if ( i < generaciones - 1 )
      resortes.nuevaGeneracion();
}
```

```
        ListaOrdenada resultados = new ListaOrdenada(resortes);
        ListIterator iter = resultados.iteradorLista(resultados.size());

        int i=0;
        while( iter.hasPrevious()){

           ItemCrom item = (ItemCrom) iter.previous();
           Crom crom = item.obtenerCrom();

           salida = salida + String.valueOf(i+1) + "\t" +
                 visDw.obtenerGene(crom,0) + "\t" +
                 visDm.obtenerGene(crom,13) + "\t" +
                 visNa.obtenerGene(crom,26) + "\t" +
                 "\t= " + item.obtenerDobleCapacidad()+"\n";

           resAG[i] = new ResorteHelicoidalCompresion();
           resAG[i].inicializar();
           resAG[i].asignarMaterial(indMat+1);
           resAG[i].asignarTipoExtremo(indExt+1);
           resAG[i].asignarTipoServicio(indSer+1);
           resAG[i].asignarSujecion(indSuj+1);
           resAG[i].asignarFo(valFo);
           resAG[i].asignarLo(valLo);
           resAG[i].asignarFi(valFi);
           resAG[i].asignarLi(valLi);
           resAG[i].asignarDo(valDo);
           resAG[i].asignarDv(valDv);



           resAG[i].asignarDw(
                ((Double)visDw.obtenerGene(crom,0)).doubleValue() );
           resAG[i].asignarDm(
                ((Double)visDm.obtenerGene(crom,13)).doubleValue() );
           resAG[i].asignarNa(
                ((Long)visNa.obtenerGene(crom,26)).intValue() );

           resAG[i].analizarAG();

           i=i+1;
        }
        jTextAreaSalida.setText(salida);
     }

     public void jButtonAnalizar_actionPerformed(ActionEvent e) {
        int i = ((SpinnerNumberMod-
el)jSpinnerAnalisis.getModel()).getNumber().intValue();
        String salida = "Análisis de la Solución " + i + "\n\n";

        int mat = resAG[i-1].obtenerMaterial();
        if (mat == 1)
           salida = salida + "Material: ASTM A227 Clase II\n";
        else if (mat == 2)
           salida = salida + "Material: ASTM A228\n";
        else if (mat == 3)
```

```
            salida = salida + "Material: ASTM A229 Clase II\n";
         else if (mat == 4)
            salida = salida + "Material: ASTM A231\n";
         else if (mat == 5)
            salida = salida + "Material: ASTM A401\n";
         else if (mat == 6)
            salida = salida + "Material: ASTM A313 (A302)\n";
         else if (mat == 7)
            salida = salida + "Material: ASTM A304\n";
         else salida = salida + "Material: ASTM A316\n";

         salida = salida +
               "Módulo de elasticidad de alambre para resorte en corte (G)" +
               ": " + resAG[i-1].obtenerG() + " psi\n";

         int ext = resAG[i-1].obtenerTipoExtremo();
         if (ext == 1)
            salida = salida + "Extremos del resorte: " +
                  "Extremos a escuadra y lijados\n";
         else if (ext == 2)
            salida = salida + "Extremos del resorte: " +
                  "Solo extremos a escuadra\n";
         else if (ext == 3)
            salida = salida + "Extremos del resorte: " +
                  "Extremos en bruto y lijados\n";
         else salida = salida + "Extremos del resorte: " +
                   "Extremos en bruto\n";

         int ser = resAG[i-1].obtenerTipoServicio();
         if (ser == 1)
            salida = salida + "Tipo de servicio: " +
                  "Servicio Ligero\n" +
                  "Cargas estáticas hasta 10000 ciclos de carga, tasa baja de carga\n";
         else if (ser == 2)
            salida = salida + "Tipo de servicio: " +
                  "Servicio Promedio\n" +
                  "\tResorte para máquinarias, tasa moderada de carga y hasta 1000000
de ciclos\n";
         else salida = salida + "Tipo de servicio: " +
                   "Servicio Severo\n" +
                   "\tCiclaje rápido para más de un millón de ciclos; posibilidad de carga
por impacto o choque\n";

         int suj = resAG[i-1].obtenerSujecion();
         if (suj == 1)
            salida = salida + "Tipo de sujeción en los extremos del resorte: " +
                  "Extremos fijos\n";
         else if (suj == 2)
            salida = salida + "Tipo de sujeción en los extremos del resorte: " +
                  "Un extremo fijo y un extremo atornillado\n";
          else salida = salida + "Tipo de sujeción en los extremos del resorte: " +
                  "Ambos extremos atornillados\n";

         salida = salida + "Fuerza en longitud de operación (Fo): " +
               resAG[i-1].obtenerFo() + " libras\n";
```

```
salida = salida + "Longitud de operación (Lo): " +
      resAG[i-1].obtenerLo() + " pulgadas\n";

salida = salida + "Fuerza en longitud instalado (Fi): " +
      resAG[i-1].obtenerFi() + " libras\n";

salida = salida + "Longitud instalado (Li): " +
      resAG[i-1].obtenerLi() + " pulgadas\n";

salida = salida + "Diámetro del orificio de instalación (Do): ";
if(resAG[i-1].obtenerDo() == 1E6)
   salida = salida + "Sin orificio\n";
else
   salida = salida + resAG[i-1].obtenerDo() + " pulgadas\n";

salida = salida + "Diámetro de la varilla de instalación (Dv): ";
if(resAG[i-1].obtenerDv() == 0)
   salida = salida + "Sin varilla\n";
else
   salida = salida + resAG[i-1].obtenerDv() + " pulgadas\n";

salida = salida + "Razón de resorte (k): " +
      resAG[i-1].obtenerk() + "\n";

salida = salida + "Longitud libre (Lf): " +
      resAG[i-1].obtenerLf() + " pulgadas\n";

salida = salida + "Deflexión en longitud de operación (fo): " +
      resAG[i-1].obtenerfo() + " pulgadas\n";

salida = salida + "Diámetro del alambre del resorte (Dw): " +
      resAG[i-1].obtenerDw() + " pulgadas\n";

salida = salida + "Tensión de diseño (Sd): " +
      resAG[i-1].obtenerSd() + " psi\n";

salida = salida + "Tensión máxima permisible (Smax): " +
      resAG[i-1].obtenerSmax() + " psi\n";

salida = salida + "Número de bobinas activas (Na): " +
      resAG[i-1].obtenerNa() + " \n";

salida = salida + "Índice de resorte (C): " +
      resAG[i-1].obtenerC() + " \n";

salida = salida + "Diámetro medio (Dm):" +
      resAG[i-1].obtenerDm() + " pulgadas\n";

salida = salida + "Diámetro exterior (OD): " +
      resAG[i-1].obtenerOD() + " pulgadas\n";

salida = salida + "Diámetro interior (ID): " +
      resAG[i-1].obtenerID() + " pulgadas\n";

salida = salida + "Razón Crítica (RC): " +
      resAG[i-1].obtenerRC() + " \n";
```

```
salida = salida + "Factor de Wahl (K): " +
      resAG[i-1].obtenerK() + " \n";

salida = salida + "Esfuerzo o tensión en carga de operación (So): " +
      resAG[i-1].obtenerSo() + " psi\n";

salida = salida + "Número total de bobinas (N): " +
      resAG[i-1].obtenerN() + " \n";

salida = salida + "Longitud comprimido (Ls): " +
      resAG[i-1].obtenerLs() + " pulgadas\n";

salida = salida + "Fuerza en longitud comprimido (Fs): " +
      resAG[i-1].obtenerFs() + " libras\n";

salida = salida + "Tensión o esfuerzo en longitud comprimido (Ss): " +
      resAG[i-1].obtenerSs() + " psi\n";

salida = salida + "Margen de bobina (cc): " +
      resAG[i-1].obtenercc() + " pulgadas\n";

salida = salida + "Volumen (vol): " +
      resAG[i-1].obtenervol() + " pulgadas cúbicas\n";

salida = salida + "\nPenalización acumulada (pen): " +
      resAG[i-1].obtenerpen() + " \n";

salida = salida + "Factor de Wahl: ";
if(resAG[i-1].obtenerC_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Índice de resorte: ";
if(resAG[i-1].obtenerC_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Diámetro exterior: ";
if(resAG[i-1].obtenerOD_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Diámetro interior: ";
if(resAG[i-1].obtenerID_OK() == true)
   salida = salida + "Correcto\n";
else
   salida = salida + "Incorrecto\n";

salida = salida + "Pandeo: ";
if(resAG[i-1].obtenerNo_Pandeo() == true)
   salida = salida + "Sin pandeo\n";
else
```

```
            salida = salida + "Con pandeo (¡cuidado!)\n";

        salida = salida + "Esfuerzo o tensión en carga de operación: ";
        if(resAG[i-1].obtenerSo_OK() == true)
            salida = salida + "Correcto\n";
        else
            salida = salida + "Incorrecto\n";

        salida = salida + "Longitud comprimido: ";
        if(resAG[i-1].obtenerLs_OK() == true)
            salida = salida + "Correcto\n";
        else
            salida = salida + "Incorrecto\n";

        salida = salida + "Tensión o esfuerzo en longitud comprimido: ";
        if(resAG[i-1].obtenerSs_OK() == true)
            salida = salida + "Correcto\n";
        else
            salida = salida + "Incorrecto\n";

        salida = salida + "Margen de bobina: ";
        if(resAG[i-1].obtenercc_OK() == true)
            salida = salida + "Correcto\n";
        else
            salida = salida + "Incorrecto\n";

        salida = salida + "Número de espiras activas: ";
        if(resAG[i-1].obtenerNa_OK() == true)
            salida = salida + "Correcto\n";
        else
            salida = salida + "Incorrecto\n";

        jTextAreaAnalisis.setText(salida);
    }
}


class FrameAGRHC_jButtonAnalizar_actionAdapter implements ActionListener {
    private FrameAGRHC adaptee;
    FrameAGRHC_jButtonAnalizar_actionAdapter(FrameAGRHC adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e) {
        adaptee.jButtonAnalizar_actionPerformed(e);
    }
}


class FrameAGRHC_jButtonAplicar_actionAdapter implements ActionListener {
    private FrameAGRHC adaptee;
    FrameAGRHC_jButtonAplicar_actionAdapter(FrameAGRHC adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e) {
```

```
        adaptee.jButtonAplicar_actionPerformed(e);
    }
}


class FrameAGRHC_jButton1_actionAdapter implements ActionListener {
    private FrameAGRHC adaptee;
    FrameAGRHC_jButton1_actionAdapter(FrameAGRHC adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e) {
        adaptee.jButton1_actionPerformed(e);
    }
}
```

F. PROGRAMA MÉTODO 1 PARA EL DISEÑO DE RESORTES

```
import javax.swing.*;

public class Metodo1
{
    public static void main(String[] args)
    {
        int opcion=1;
        String introduccion, opciones;
        String entrada;

        double G, Fo, Lo, Fi, Li, k, Lf, Dm, Sd, Dw, Sm, C, K, So,
            Na, OD, ID, Ls, DLs, Fs, Ss;

        do
        {
            introduccion = "Programa para asistir al Diseño de Resortes"
                + "\nde Compresión Helicoidal con un Diámetro"
                + "\nMedio Específico\n"
                + "Los datos de entrada requeridos son:\n"
                + " - Modulo de elasticidad ante esfuerzo de corte del\n"
                + "   material (G) en PSI\n"
                + " - Fuerza de operación máxima (Fo) en libras\n"
                + " - Longitud de operación (Lo) en pulgadas\n"
                + " - Fuerza de instalado (Fi) en libras\n"
                + " - Longitud de instalado (Li) en pulgadas\n";

                                ;

            JOptionPane.showMessageDialog(null,introduccion,
                "Diseño de Resortes Helicoidales",
                JOptionPane.INFORMATION_MESSAGE);

            entrada = JOptionPane.showInputDialog
                ("Módulo de elasticidad ante esfuerzo\n"+
                "de corte del material (G)");
            G = Double.parseDouble(entrada);

            entrada = JOptionPane.showInputDialog
                ("Fuerza en longitud de operación (Fo)\n"+
                 "(la fuerza máxima que observa el resorte\n"+
                 " en operación normal)");
```

```
Fo = Double.parseDouble(entrada);

entrada = JOptionPane.showInputDialog
    ("Longitud de operación (Lo)");
Lo = Double.parseDouble(entrada);

entrada = JOptionPane.showInputDialog
    ("Fuerza en longitud de instalado (Fi)");
Fi = Double.parseDouble(entrada);

entrada = JOptionPane.showInputDialog
    ("Longitud de instalado (Li)");
Li = Double.parseDouble(entrada);

//Razón de resorte
k = (Fo-Fi)/(Li-Lo);

JOptionPane.showMessageDialog(null,
    "La razón de resorte k es " + k +" lb/pulg",
    "Razón de resorte",
    JOptionPane.INFORMATION_MESSAGE);

//Longitud Libre

Lf = Li + Fi/k;

JOptionPane.showMessageDialog(null,
    "La Longitud libre, Lf es " + Lf +" pulg",
    "Longitud libre",
    JOptionPane.INFORMATION_MESSAGE);
do
{
    entrada = JOptionPane.showInputDialog
        ("Especifique un estimado inicial para\n"+
         "el diametro medio (Dm)");
    Dm = Double.parseDouble(entrada);

    entrada = JOptionPane.showInputDialog
        ("Especifique una tensión de diseño inicial (Sd)\n");
    Sd = Double.parseDouble(entrada);

    // Diametro del alambre Dw para la prueba, considerando el
    // factor de Wahl K = 1.2

    Dw = Math.pow(8.0*1.2*Fo*Dm/(Math.PI*Sd),1.0/3.0);

    JOptionPane.showMessageDialog(null,
        "Diametro del alambre (Dw) para la prueba: " + Dw +
        "pulg\n considerando Factor de Wahl K = 1.2",
        "Diámetro del alambre",
        JOptionPane.INFORMATION_MESSAGE);

    do
    {
        entrada = JOptionPane.showInputDialog
            ("Seleccione un diámetro de alambre estándar(Dw)"+
```

```
        "de las tablas");

    Dw = Double.parseDouble(entrada);

    entrada = JOptionPane.showInputDialog
        ("Especifique la tensión de diseño(Sd) para\n"+
         "el material con ese diámetro");

    Sd = Double.parseDouble(entrada);

    entrada = JOptionPane.showInputDialog
        ("Especifique la tensión máxima tolerable(Sm)\n"+
         "para el material con ese diámetro");

    Sm = Double.parseDouble(entrada);

    C = Dm/Dw;

    K = (4*C - 1.0)/(4*C - 4.0) + 0.615/C;

    So = 8.0*K*Fo*Dm/(Math.PI*Math.pow(Dw,3.0));

    JOptionPane.showMessageDialog( null,
        "La Tensión de operación (So) = " + So + "PSI",
        "Tensión de Operación",
        JOptionPane.INFORMATION_MESSAGE);


    if (So<Sd)
        JOptionPane.showMessageDialog( null,
            "La Tensión de operación (So) es Aceptable",
            "Tensión de Operación",
            JOptionPane.INFORMATION_MESSAGE);
    else
        JOptionPane.showMessageDialog( null,
            "La Tensión de operación (So) execede la" +
            "Tensión de Diseño (Sd)",
            "Tensión de Operación",
            JOptionPane.INFORMATION_MESSAGE);

    //Número de bobinas activas, Na
    Na = G*Dw/(8*k*Math.pow(C,3.0));

    //Diámetro exterior
    OD = Dm + Dw;

    //Diámetro Interior
    ID = Dm - Dw;

    JOptionPane.showMessageDialog(null,
        "Diámetro medio (Dm) = " + Dm + " pulg\n"+
        "Diámetro Exterior (OD) = " + OD + " pulg\n"+
        "Diámetro Interior (ID) = " + ID + " pulg\n\n" +
        "Índice del Resorte (C) = " + C,
        "Para el diámetro medio especificado",
        JOptionPane.INFORMATION_MESSAGE);
```

```
if (C>5)
    JOptionPane.showMessageDialog(null,
        "Índice de Resorte Aceptable",
        "Índice de Resorte",
        JOptionPane.INFORMATION_MESSAGE);

else
    JOptionPane.showMessageDialog(null,
        "CUIDADO: Índice de Resorte muy bajo",
        "Índice de Resorte",
        JOptionPane.INFORMATION_MESSAGE);


JOptionPane.showMessageDialog(null,
    "Número de bobinas activas = "+Na+" bobinas",
    "Bobinas activas",
    JOptionPane.INFORMATION_MESSAGE);

//Longitud de comprimido, Ls
Ls = Dw*(Na+2);

if (Ls<Lo)
{
    JOptionPane.showMessageDialog(null,
        "Longitud de Comprimido menor\n"+
        "a la Longitud de Operación\n",
        "Mensaje",
        JOptionPane.INFORMATION_MESSAGE);

    //Deflexión en longitud de comprimido
    DLs = Lf - Ls;

    //Fuerza en longitud de comprimido
    Fs = k*(Lf-Ls);

    //Tensión en longitud de comprimido
    Ss = So*(Fs/Fo);

    JOptionPane.showMessageDialog(null,
        "Longitud de Comprimido (Ls) = " + Ls + " pulg\n"+
        "Fuerza en longitud de comprimido (Fs) = " + Fs +
        " libras\n"+
        "Tensión en longitud de comprimido (Ss) = " + Ss +
        " PSI\n\n",
        "Mensaje",
        JOptionPane.INFORMATION_MESSAGE);

    if (Ss < Sm)
        JOptionPane.showMessageDialog(null,
            "Tensión en longitud de comprimido (Ss)\n"+
            "ACEPTABLE\n",
            "Tensión en longitud de comprimido",
            JOptionPane.INFORMATION_MESSAGE);
    else
        JOptionPane.showMessageDialog(null,
```

```
                        "CUIDADO\n" +
                        "Tensión en longitud de comprimido (Ss)\n" +
                        "muy alta",
                        "Tensión en longitud de comprimido",
                        JOptionPane.INFORMATION_MESSAGE);

                    opciones =
                        "1 para Salir\n" +
                        "2 para un nuevo problema\n" +
                        "3 para cambiar el diámetro medio (Dm)\n" +
                        "4 para cambiar el diámetro del alambre (Dw)\n";

                    entrada = JOptionPane.showInputDialog(opciones);

                    opcion = Integer.parseInt(entrada);
                }
                else
                    JOptionPane.showMessageDialog(null,
                        "CUIDADO: Longitud de Comprimido mayor o igual\n"+
                        "       a la Longitud de Operación \n" +
                        "SITUACIÓN IMPOSIBLE",
                        "Advertencia",
                        JOptionPane.INFORMATION_MESSAGE);

            } while (opcion == 4);

        } while (opcion == 3);

    } while(opcion == 2);

    System.exit(0);
  }

}
```

G. PROGRAMA MÉTODO 2 PARA EL DISEÑO DE RESORTES

```
import javax.swing.*;

public class Metodo2
{
   static double Datos[] = {0.0204,0.0230,0.0258,0.0286,0.0317,0.0348,0.0410,
                            0.0475,0.0540,0.0625,0.0720,0.0800,0.0915,0.1055,
                     0.1205,0.1350,0.1483,0.1620,0.1770,0.1920,0.2070,
                     0.2253,0.2437,0.2625,0.2830,0.3065,0.3310,0.3625,
                     0.3938,0.4305,0.4615,0.4900,0.0000,0.0000,0.0000};

   static int opcion, servicio, i;
   static String introduccion, opciones, A;
   static String entrada;

   static double G, Fo, Lo, Fi, Li, k, Lf, Dm, Sd, Dw, Sm, C, K, So,
      Na, OD, ID, Ls, DLs, Fs, Ss, DLo, Nam, Da, R, cc, ccR;

   public static void main(String[] args)
   {
      do
      {
         // Para el alambre A231,Modulo de elasticidad ante esfuerzo de corte
         // G=11 200 000 PSI
         G=1.12E07;

         introduccion = "Programa para asistir al Diseño de Resortes"
            + "\nde Compresión Helicoidal con un alambre"
            + "\nASTM A231\n\n"
            + "Los extremos son a escuadra y lijados\n\n"
            + "Los datos de entrada requeridos son:\n"
            + " - Fuerza de operación máxima (Fo) en libras\n"
            + " - Longitud de operación (Lo) en pulgadas\n"
            + " - Fuerza de instalado (Fi) en libras\n"
            + " - Longitud de instalado (Li) en pulgadas\n\n";

         JOptionPane.showMessageDialog(null,introduccion,
            "Diseño de Resortes Helicoidales",
            JOptionPane.INFORMATION_MESSAGE);

         entrada = JOptionPane.showInputDialog
            ("Fuerza en longitud de operación (Fo)\n"+
```

```
    "(la fuerza máxima que observa el resorte\n"+
    " en operación normal)");

Fo = Double.parseDouble(entrada);

entrada = JOptionPane.showInputDialog
    ("Longitud de operación (Lo)");

Lo = Double.parseDouble(entrada);

entrada = JOptionPane.showInputDialog
    ("Fuerza en longitud de instalado (Fi)");

Fi = Double.parseDouble(entrada);

entrada = JOptionPane.showInputDialog
    ("Longitud de instalado (Li)");

Li = Double.parseDouble(entrada);

//Razón de resorte
k = (Fo-Fi)/(Li-Lo);

//Longitud Libre
Lf = Li + Fi/k;

//Deflexión en Longitud de Operación
DLo = Lf - Lo;


entrada = JOptionPane.showInputDialog
    ("Ingrese un estimado inicial del diámetro del alambre (Dw)"+
     "\nen pulg\n");

Dw = Double.parseDouble(entrada);

entrada = JOptionPane.showInputDialog
    ( "Ingrese el tipo de servicio\n"
     + "- 1 Servicio Ligero\n"
     + "- 2 Servicio Promedio\n"
     + "- 3 Servicio Severo\n");

servicio = Integer.parseInt(entrada);

tensiones();

JOptionPane.showMessageDialog(null,
    "Tensión de diseño inicial (Sd) = " + Sd +" PSI\n" +
    "Tensión máxima permisible (Sm) = " + Sm + "PSI\n",
    "Tensiones Iniciales",
    JOptionPane.INFORMATION_MESSAGE);

//Cálculo del Nuevo diámetro de alambre de prueba
Dw = Math.sqrt(21.4*Fo/Sd);

JOptionPane.showMessageDialog(null,
```

```
    "RESULTADOS DE PRUEBA\n\n" +
    "Longitud Libre (Lf) = " + Lf +" Pulg\n" +
    "Deflexión en longitud de operación (DLo) = " + DLo +" Pulg\n"+
    "Razón de resorte (k) = " + k +" lib/pulg\n\n" +
    "Diámetro del alambre de prueba (Dw) = " + Dw,
    "Resultados de Prueba",
    JOptionPane.INFORMATION_MESSAGE);

buscar();

do
{
    entrada = JOptionPane.showInputDialog
        ("Los diámetros de alambre estándar más\n" +
         "cercanos son:\n"+
         Datos[i-1] + "  " + Datos[i] + "  " + Datos[i+1] + "\n" +
         Datos[i+2] + "  " + Datos[i+3] + "  " + Datos[i+4] +
         "\n\n" + "Ingrese el diámetro de alambre seleccionado\n");

    Dw = Double.parseDouble(entrada);

    tensiones();

    //Número de bobinas máximo (Nam)

    Nam = (Lo-2*Dw)/Dw;

    do
    {
        entrada = JOptionPane.showInputDialog
            ("Especifique un Número de bobinas activas (Na) menor" +
             "\nque " + Nam +"\n");
        Na = Double.parseDouble(entrada);

        //índice del resorte (C)
        C = Math.pow(G*Dw/(8.0*k*Na),1.0/3.0);

        if (C>5)
            JOptionPane.showMessageDialog(null,
                "Índice de resorte (C) = " + C + "\n" +
                "ACEPTABLE",
                "Índice de Resorte (C)",
                JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(null,
                "Índice de resorte (C) = " + C + "\n" +
                "MUY BAJO",
                "ADEVERTENCIA: Índice de Resorte (C)",
                JOptionPane.INFORMATION_MESSAGE);

        //Diámetro medio (Dm)
        Dm = C*Dw;

        //Diámetro exterior (OD)
        OD = Dm + Dw;
```

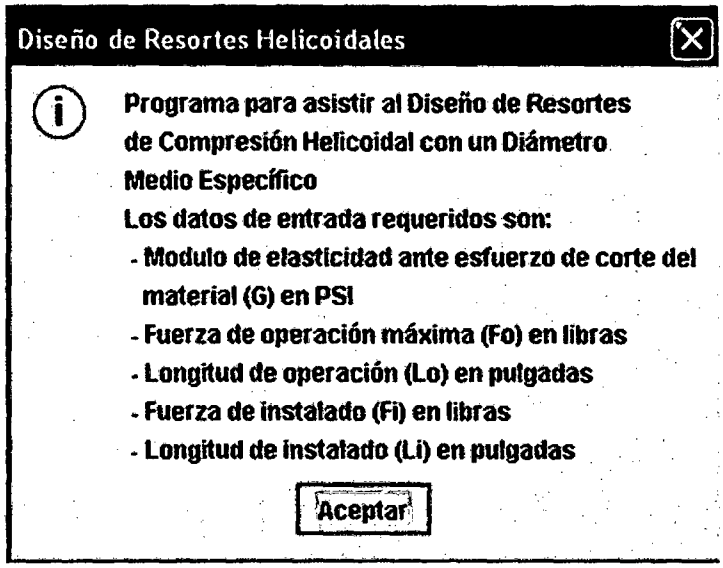
```
//Diámetro interior (OD)
ID = Dm - Dw;

//Diámetro del agujero (Da)
Da = OD + Dw/2.0;

JOptionPane.showMessageDialog(null,
   "Diámetro del alambre (Dw) = " + Dw + " pulg\n" +
   "Diámetro medio (Dm) = " + Dm + " pulg\n" +
   "Diámetro Exterior (OD) = " + OD + " pulg\n" +
   "Diámetro Interior (ID) = " + ID + " pulg\n\n" +
   "NOTA: El Diámetro del Agujero debe\n"+
   "      ser mayor a " + Da + " pulg\n\n" +
   "Índice del resorte = " + C + " \n",
   "Geometría de Prueba",
   JOptionPane.INFORMATION_MESSAGE);

JOptionPane.showMessageDialog(null,
   "Longitud Libre / Diámetro Medio = Lf/Dm\n" +
   (Lf/Dm),
   "Longitud Libre / Diámetro Medio",
   JOptionPane.INFORMATION_MESSAGE);

if (Lf/Dm > 5.4)
   {
      R = 14.2748/Math.pow((Lf/Dm),2.03558);

      JOptionPane.showMessageDialog(null,
         "EL PANDEO PUEDE SER UN PROBLEMA\n" +
         "Razón: Deflexión de Operación/Longitud "+
         "Libre =\n DLo/Lf = " + (DLo/Lf) + "\n" +
         "Razón Máxima = " + R + "\n\n" +
         "Prueba con aumentar el Diámetro Medio o " +
         "disminuir la Longitud Libre",
         "Pandeo",
         JOptionPane.INFORMATION_MESSAGE);
   }
else
   JOptionPane.showMessageDialog(null,
      "EL PANDEO NO SERÁ UN PROBLEMA",
      "Pandeo",
      JOptionPane.INFORMATION_MESSAGE);

JOptionPane.showMessageDialog(null,
   "Tensión de Diseño (Sd) = " + Sd + " PSI\n" +
   "Tensión Máxima Permitida (Sm) = " + Sm + " PSI",
   "Geometría de Prueba",
   JOptionPane.INFORMATION_MESSAGE);

entrada = JOptionPane.showInputDialog(
   "¿Está bien la geometría?\n" +
   "(Escriba SI o NO)\n");
A = entrada.toUpperCase();
if (A.equals("SI"))
{
   JOptionPane.showMessageDialog(null,
```

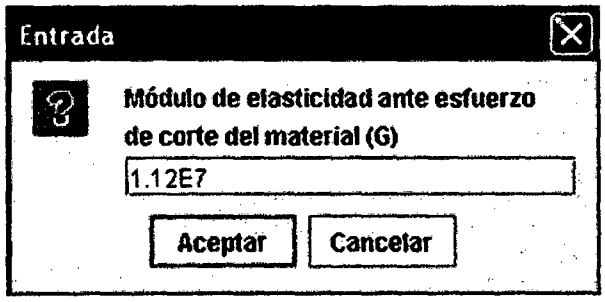
```
    "GEOMETRÍA ¡OK!\nREVISE LAS TENSIONES\n\n",
    "Geometría",
    JOptionPane.INFORMATION_MESSAGE);

//Factor de Wahl
K = (4*C-1)/(4*C-4)+0.615/C;

//Tensión Actual esperada en Fuerza de Operación
So = 2.546*K*C*Fo/Math.pow(Dw,2.0);

//Longitud de comprimido
Ls = Dw*(Na+2);

//Deflexión en Longitud de Comprimido
DLs = Lf - Ls;

//Fuerza en Longitud de Comprimido
Fs = k*DLs;

//Tensión en Longitud de Comprimido
Ss = So*Fs/Fo;

JOptionPane.showMessageDialog(null,
    "Tensión debida a la Fuerza de Operación:\n" +
    So + "Psi",
    "Tensión en Fuerza de Operación",
    JOptionPane.INFORMATION_MESSAGE);

if(So > Sd)
    JOptionPane.showMessageDialog(null,
        "Tensión en Longitud de " +
        "Operación MUY ALTA\n"  +
        "Tensión de Diseño: " + Sd + " PSI",
        "Tensión en Fuerza de Operación",
        JOptionPane.INFORMATION_MESSAGE);
else
    JOptionPane.showMessageDialog(null,
        "Tensión en Longitud de " +
        "Operación ACEPTABLE",
        "Tensión en Fuerza de Operación",
        JOptionPane.INFORMATION_MESSAGE);
if (Ss > Sm)
    JOptionPane.showMessageDialog(null,
        "Longitud de Comprimido: " + Ls + " pulg\n" +
        "Tensión en Longitud de " +
        "Comprimido: " + Ss + " PSI MUY ALTA\n" +
        "Tensión de Compresión máxima: " +
         Sm + " PSI\n" +
        "Una elongación ocurrirá en " +
        "Longitud de Comprimido ",
        "Tensión en Longitud de Compresión",
        JOptionPane.INFORMATION_MESSAGE);
else
    JOptionPane.showMessageDialog(null,
        "Longitud de Comprimido: " + Ls + " pulg\n" +
        "Tensión en Longitud de " +
```

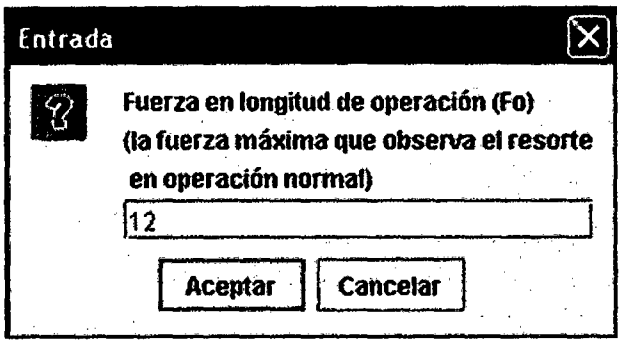
```
                    "Comprimido: " + Ss + " PSI ACEPTABLE",
                    "Tensión en Longitud de Comprimido",
                    JOptionPane.INFORMATION_MESSAGE);

                //Margen de las bobinas en Longitud de
                //Operación
                cc = (Lo-Ls)/Na;

                //Margen de las bobinas recomendado
                ccR = 0.1*Dw;

                if (cc < ccR)
                    JOptionPane.showMessageDialog(null,
                        "Margen de bobinas = " + cc + " pulg\n" +
                        "0.1 x Dw = "  + ccR + " pulg\n\n" +
                        "Disminuya el número de bobinas o \n" +
                        "aumente el diámetro del alambre",
                        "Margen de Bobinas",
                        JOptionPane.INFORMATION_MESSAGE);
                else
                    JOptionPane.showMessageDialog(null,
                        "Margen de bobinas = " + cc +
                        " pulg\n ACEPTABLE",
                        "Margen de Bobinas",
                        JOptionPane.INFORMATION_MESSAGE);
            }
            opciones = "1 para Salir\n" +
                "2 para un nuevo problema\n" +
                "3 para cambiar el número de bobinas (Na)\n" +
                "4 para cambiar el diámetro del alambre (Dw)\n";

            entrada = JOptionPane.showInputDialog(opciones);

            opcion = Integer.parseInt(entrada);

        } while (opcion == 3);

    } while (opcion == 4);

} while(opcion == 2);

System.exit(0);
}

public static void tensiones()
{
    Sm = 110843/Math.pow(Dw,0.1486);

    switch (servicio)
    {
        case 1:
            Sd = Sm;
            break;
        case 2:
            Sd = 94487/Math.pow(Dw,0.1484);
            break;
```

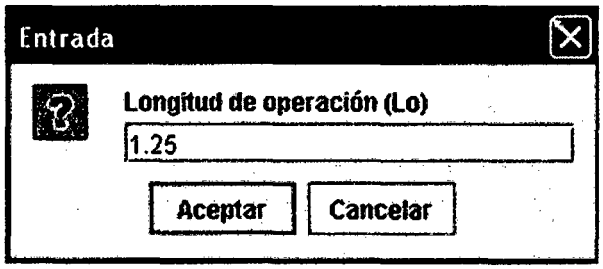
```
                case 3:
                    Sd = 75942/Math.pow(Dw,0.1532);
                    break;
                default:
                    Sd = 94487/Math.pow(Dw,0.1484);
            }
        }

        public static void buscar()
        {
            for ( i=0; i<32 ; i++ )
            {
                if (Datos[i] > Dw)
                {
                    if (i==0)
                        i=i+1;
                    break;
                }
            }
        }
}
```

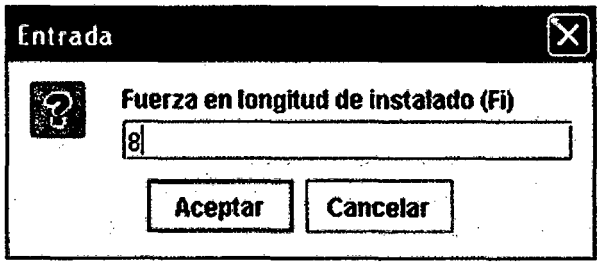
H. EJEMPLO DE EJECUCIÓN DEL PROGRAMA MÉTODO 1

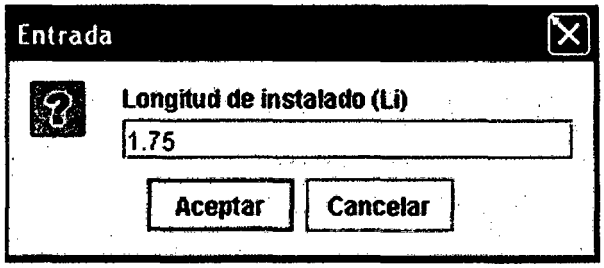


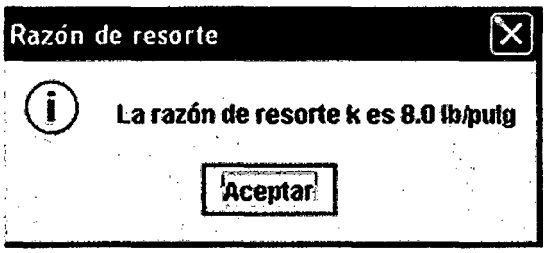


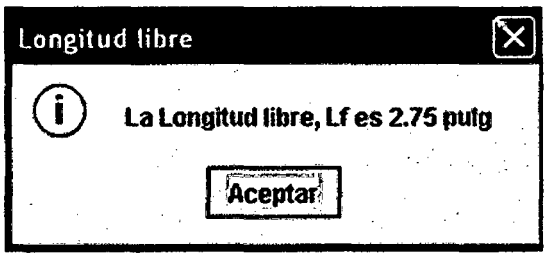


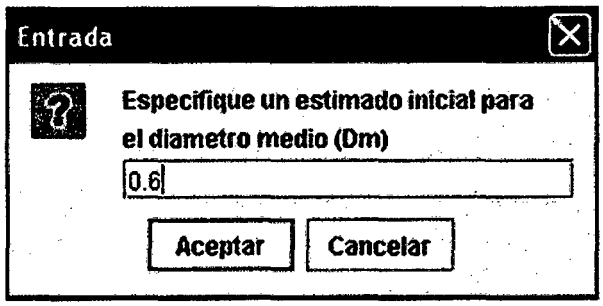

Entrada
Longitud de operación (Lo)
1.25
Aceptar
Cancelar

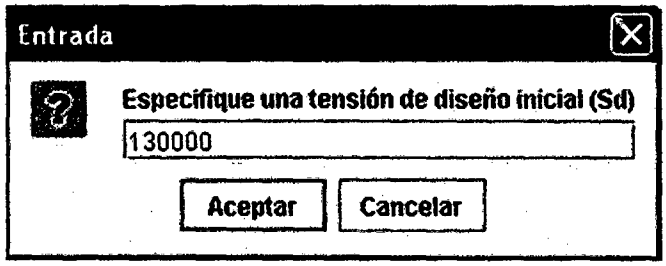

Entrada
Fuerza en longitud de instalado (Fi)
8
Aceptar
Cancelar

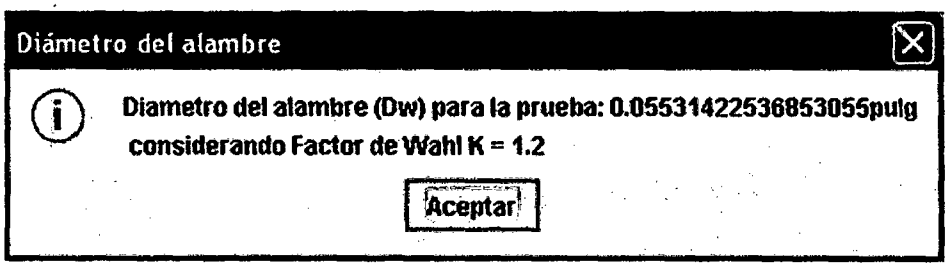

Entrada
Longitud de instalado (Li)
1.75
Aceptar
Cancelar

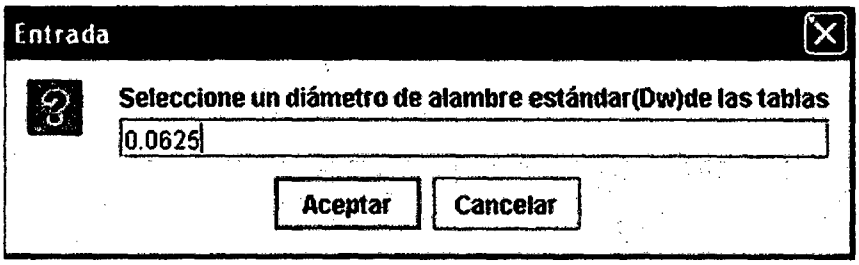

Razón de resorte
La razón de resorte k es 8.0 lb/pulg
Aceptar

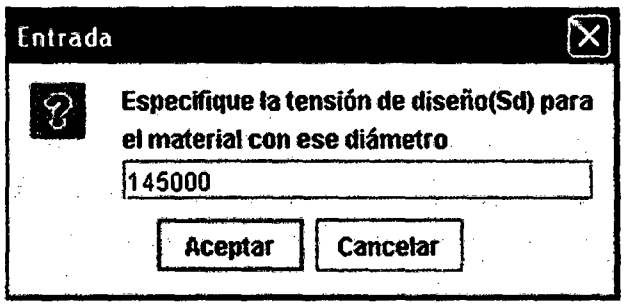

Longitud libre
La Longitud libre, Lf es 2.75 pulg
Aceptar

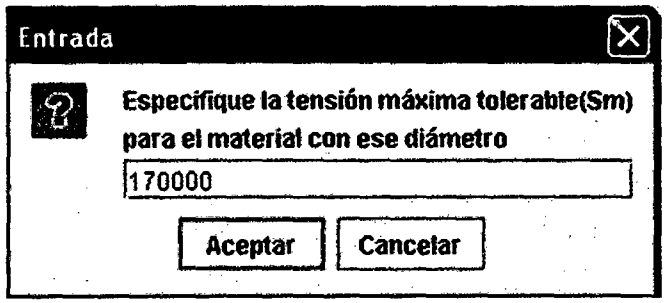

Entrada
Especifique un estimado inicial para
el diametro medio (Dm)
0.6
Aceptar
Cancelar

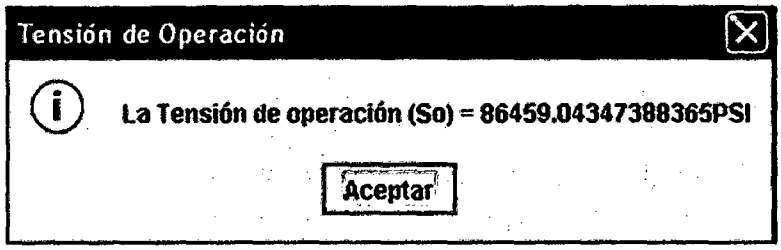

Entrada
Especifique una tensión de diseño inicial (Sd)
130000
Aceptar
Cancelar

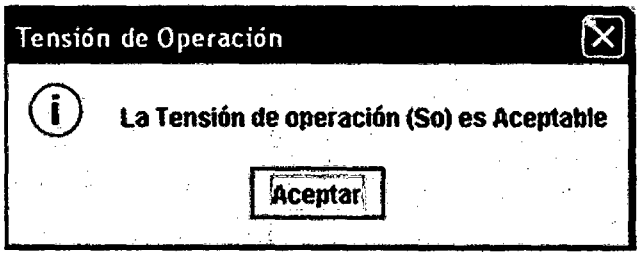

Diámetro del alambre
Diametro del alambre (Dw) para la prueba: 0.05531422536853055pulg
considerando Factor de Wahl K = 1.2
Aceptar

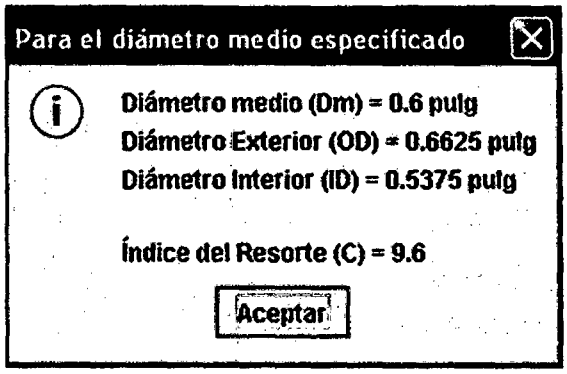

Entrada
Seleccione un diámetro de alambre estándar(Dw)de las tablas
0.0625
Aceptar
Cancelar

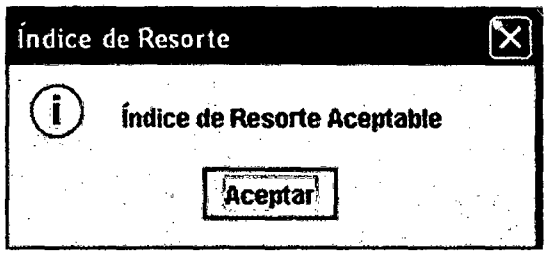

Entrada
Especifique la tensión de diseño(Sd) para
el material con ese diámetro
145000
Aceptar
Cancelar

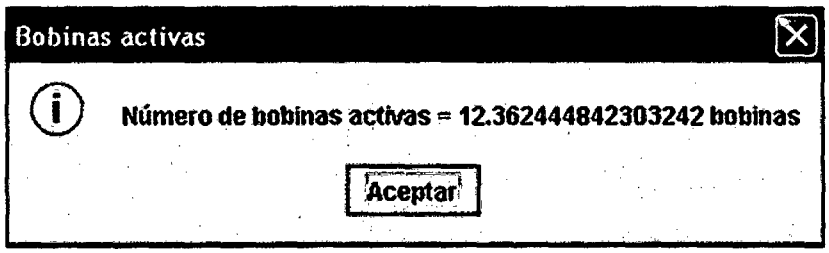

Entrada
Especifique la tensión máxima tolerable(Sm)
para el material con ese diámetro
170000
Aceptar
Cancelar

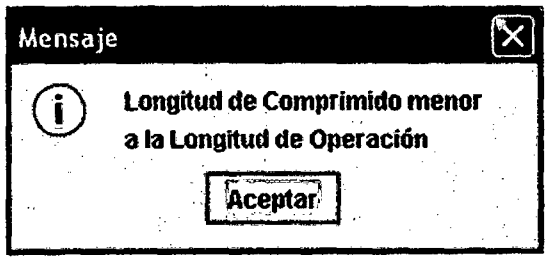

Tensión de Operación
La Tensión de operación (So) = 86459.04347388365PSI
Aceptar

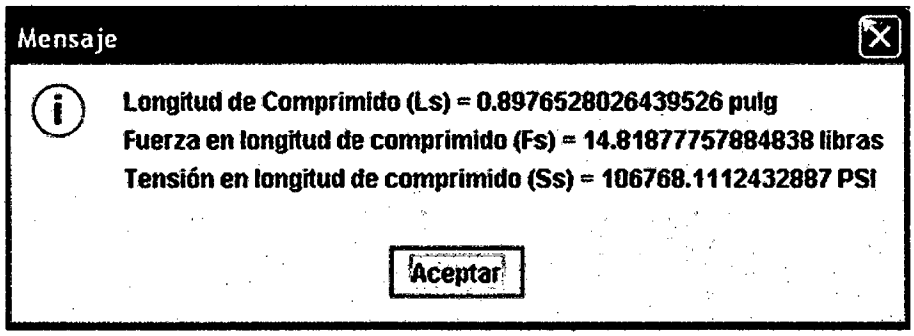

Tensión de Operación
La Tensión de operación (So) es Aceptable
Aceptar

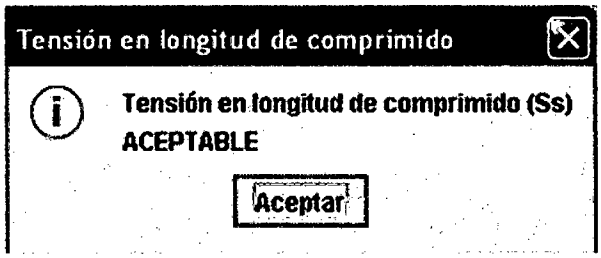

Para el diámetro medio especificado
Diámetro medio (Dm) = 0.6 pulg
Diámetro Exterior (OD) = 0.6625 pulg
Diámetro Interior (ID) = 0.5375 pulg
Índice del Resorte (C) = 9.6
Aceptar

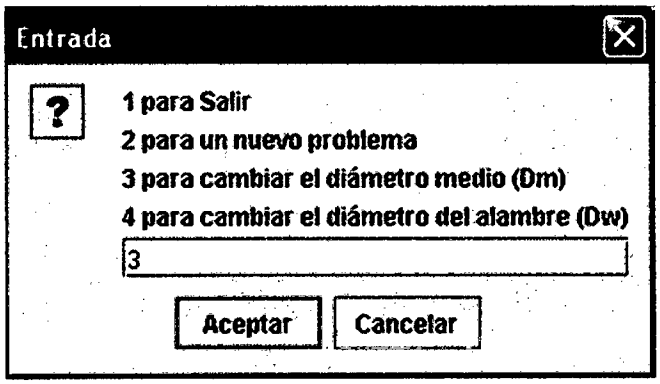

Índice de Resorte
Índice de Resorte Aceptable
Aceptar

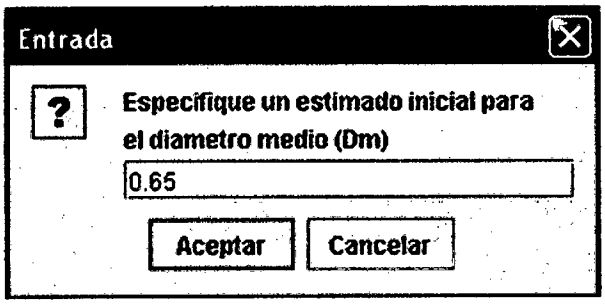

Bobinas activas
Número de bobinas activas = 12.362444842303242 bobinas
Aceptar

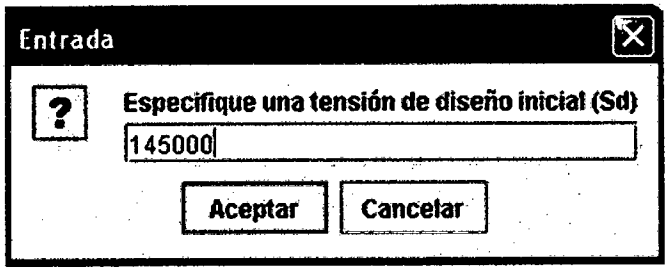

Mensaje
Longitud de Comprimido menor
a la Longitud de Operación
Aceptar

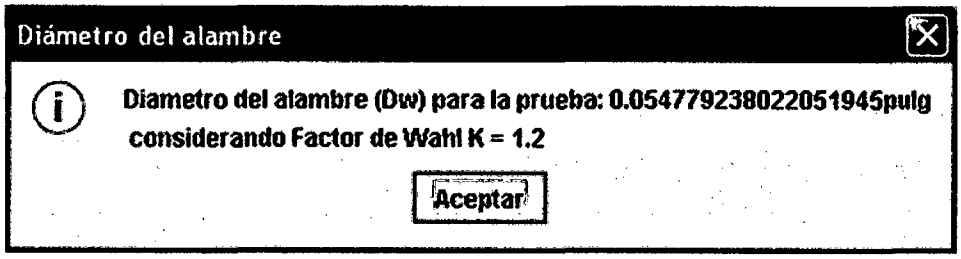

Mensaje
Longitud de Comprimido (Ls) = 0.8976528026439526 pulg
Fuerza en longitud de comprimido (Fs) = 14.81877757884838 libras
Tensión en longitud de comprimido (Ss) = 106768.1112432887 PSI
Aceptar

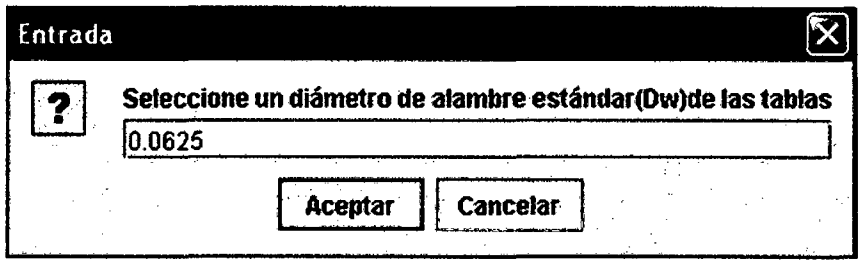

Tensión en longitud de comprimido
Tensión en longitud de comprimido (Ss)
ACEPTABLE
Aceptar

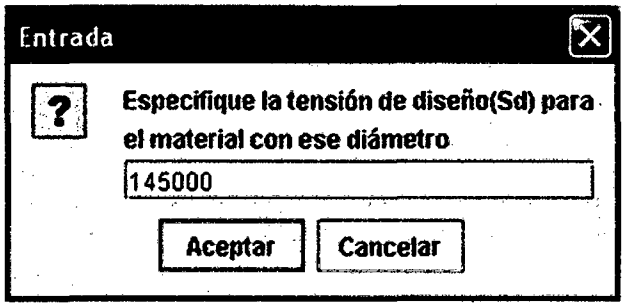

Entrada
1 para Salir
2 para un nuevo problema
3 para cambiar el diámetro medio (Dm)
4 para cambiar el diámetro del alambre (Dw)
3
Aceptar
Cancelar

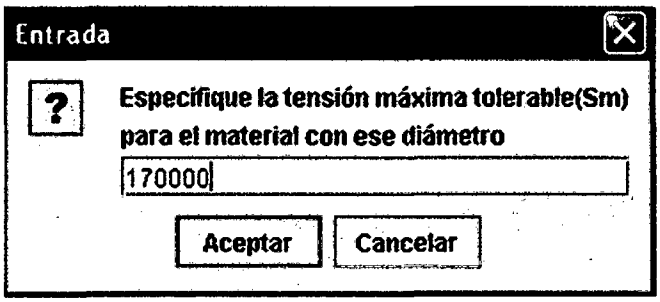

Entrada
Especifique un estimado inicial para
el diametro medio (Dm)
0.65
Aceptar
Cancelar

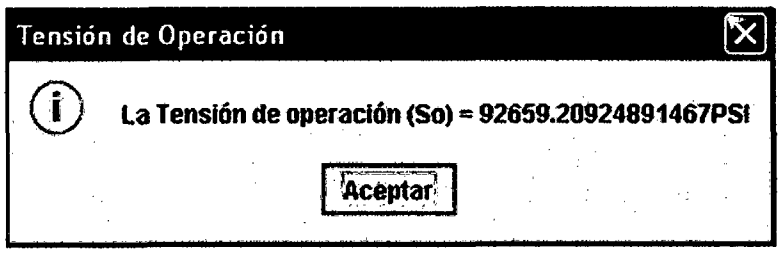

Entrada
Especifique una tensión de diseño inicial (Sd)
145000
Aceptar
Cancelar

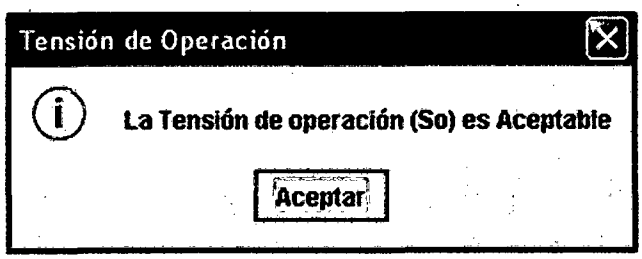

Diámetro del alambre
Diametro del alambre (Dw) para la prueba: 0.054779238022051945pulg
considerando Factor de Wahl K = 1.2
Aceptar

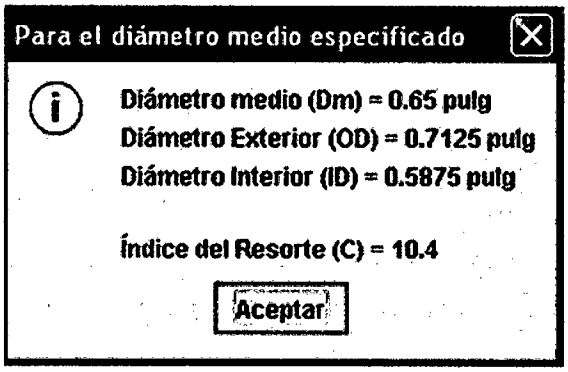

Entrada
Seleccione un diámetro de alambre estándar(Dw)de las tablas
0.0625
Aceptar
Cancelar

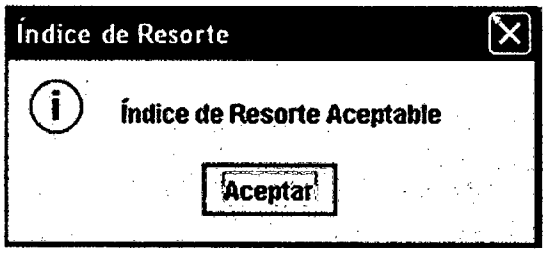

Entrada
Especifique la tensión de diseño(Sd) para
el material con ese diámetro
145000
Aceptar
Cancelar

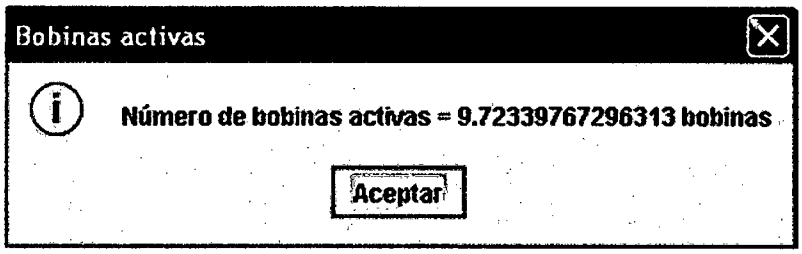

Entrada
Especifique la tensión máxima tolerable(Sm)
para el material con ese diámetro
170000
Aceptar
Cancelar

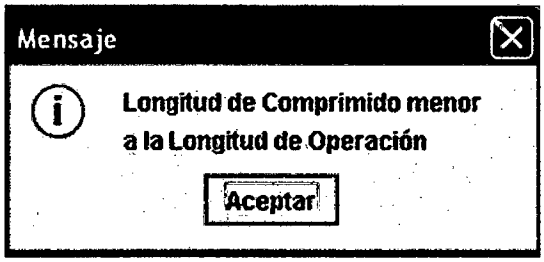

Tensión de Operación
La Tensión de operación (So) = 92659.20924891467PSI
Aceptar

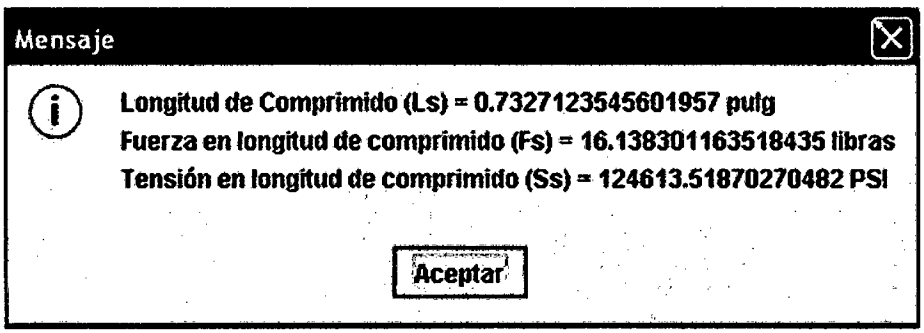

Tensión de Operación
La Tensión de operación (So) es Aceptable
Aceptar

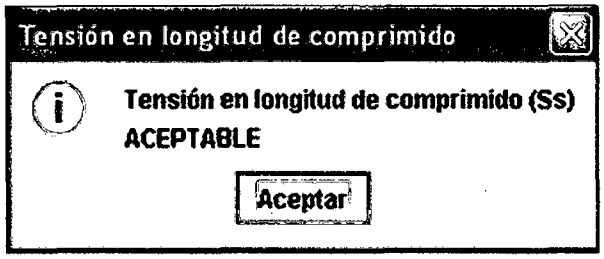

Para el diámetro medio especificado
Diámetro medio (Dm) = 0.65 pulg
Diámetro Exterior (OD) = 0.7125 pulg
Diámetro Interior (ID) = 0.5875 pulg
Índice del Resorte (C) = 10.4
Aceptar

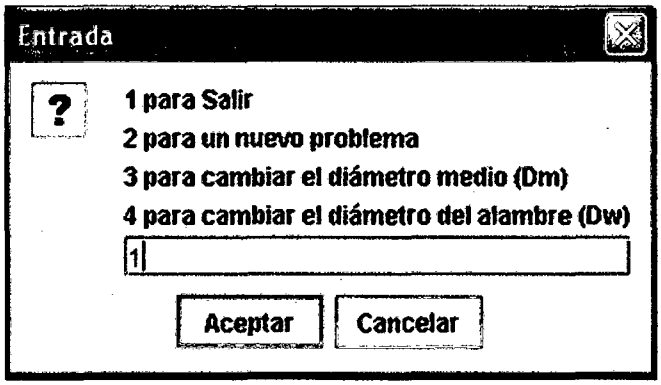

Índice de Resorte
Índice de Resorte Aceptable
Aceptar


Bobinas activas
Número de bobinas activas = 9.72339767296313 bobinas
Aceptar


Mensaje
Longitud de Comprimido menor
a la Longitud de Operación
Aceptar


Mensaje
Longitud de Comprimido (Ls) = 0.7327123545601957 pulg
Fuerza en longitud de comprimido (Fs) = 16.138301163518435 libras
Tensión en longitud de comprimido (Ss) = 124613.51870270482 PSI
Aceptar


Tensión en longitud de comprimido
Tensión en longitud de comprimido (Ss)
ACEPTABLE
Aceptar


Entrada
1 para Salir
2 para un nuevo problema
3 para cambiar el diámetro medio (Dm)
4 para cambiar el diámetro del alambre (Dw)
1
Aceptar
Cancelar

I. EJECUCIÓN DEL CUASI EXPERIMENTO

CASO 1

Un resorte helicoidal de compresión debe ejercer una fuerza de 8.0 lb cuando se comprime a una longitud de 1.75". A una longitud de 1.25", la fuerza debe ser de 12.0 lb. El resorte se instalará en una máquina que cumple ciclos con lentitud, y aproximadamente se esperan 200,00 ciclos en total. La temperatura no excederá los 200° F. Se contempla instalar el resorte en un orificio cuyo diámetro es de 0.75". Especifique un material adecuado para esta aplicación, diámetro del alambre, diámetro medio, OD, ID, longitud libre, longitud comprimido, número de bobinas y tipo de condición en los extremos. Verifique la tensión en la carga máxima de operación y la condición de longitud comprimido.

Consideraciones:

Por la temperatura el material puede ser ASTM A231
El tipo de servicio es promedio

Sin algoritmos genéticos (Después de 30 iteraciones)

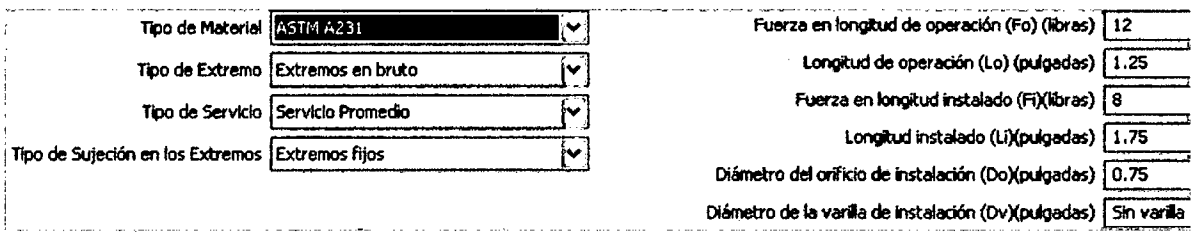
Tipo de Material	ASTM A231	Fuerza en longitud de operación (Fo) (libras)	12
Tipo de Extremo	Extremos en bruto	Longitud de operación (Lo) (pulgadas)	1.25
Tipo de Servicio	Servicio Promedio	Fuerza en longitud instalado (Fi)(libras)	8
Tipo de Sujeción en los Extremos	Extremos fijos	Longitud instalado (Li)(pulgadas)	1.75
		Diámetro del alambre de resorte (Dw)(pulgadas)	0.072
		Diámetro medio del resorte (Dm)(pulgadas)	0.6
		Número de bobinas activas (Na)	6
		Diámetro del orificio de instalación (Do)(pulgadas)	0.75
		Diámetro de la varilla de instalación (Dv)(pulgadas)	Sin varilla

Material: ASTM A231
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi
Extremos del resorte: Extremos en bruto
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 12.0 libras
Longitud de operación (Lo): 1.25 pulgadas
Fuerza en longitud instalado (Fi): 8.0 libras
Longitud instalado (Li): 1.75 pulgadas

Diámetro del orificio de instalación (Do): 0.75 pulgadas
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 8.0
Longitud libre (Lf): 2.75 pulgadas
Deflexión en longitud de operación (fo): 1.5 pulgadas
Diámetro del alambre del resorte (Dw): 0.072 pulgadas
Tensión de diseño (Sd): 143420.00000000003 psi
Tensión máxima permisible (Smax): 135620.0 psi
Número de bobinas activas (Na): 6
Índice de resorte (C): 12.805791649874942
Diámetro medio (Dm): 0.6 pulgadas
Diámetro exterior (OD): 0.6719999999999999 pulgadas
Diámetro interior (ID): 0.528 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.1115532862278132
Esfuerzo o tensión en carga de operación (So): 54601.60494234473 psi
Número total de bobinas (N): 6
Longitud comprimido (Ls): 0.43199999999999994 pulgadas
Fuerza en longitud comprimido (Fs): 18.544 libras
Tensión o esfuerzo en longitud comprimido (Ss): 84377.6801709034 psi
Margen de bobina (cc): 0.13633333333333333 pulgadas
Volumen (vol): 0.0460476262937225 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Aplicando algoritmos genéticos:



Material: ASTM A231
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi
Extremos del resorte: Extremos en bruto
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 12.0 libras
Longitud de operación (Lo): 1.25 pulgadas
Fuerza en longitud instalado (Fi): 8.0 libras
Longitud instalado (Li): 1.75 pulgadas
Diámetro del orificio de instalación (Do): 0.75 pulgadas
Diámetro de la varilla de instalación (Dv): Sin varilla

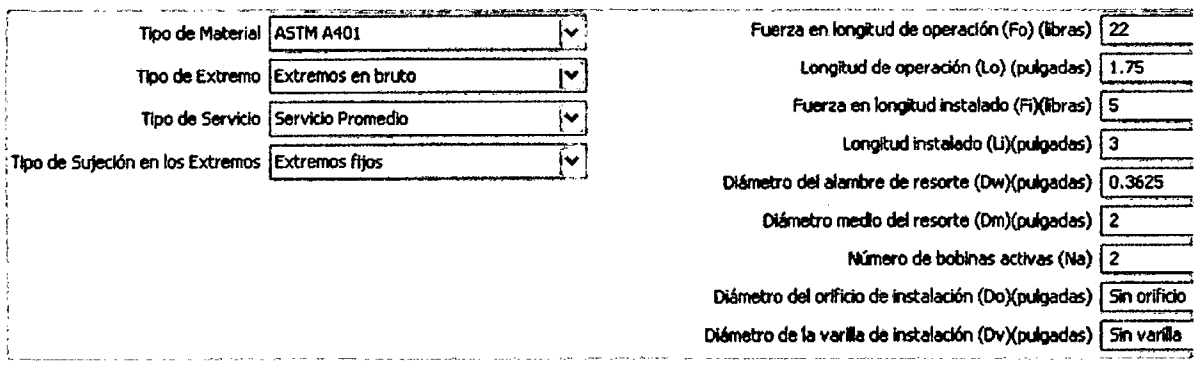
Razón de resorte (k): 8.0
Longitud libre (Lf): 2.75 pulgadas
Deflexión en longitud de operación (fo): 1.5 pulgadas
Diámetro del alambre del resorte (Dw): 0.0625 pulgadas
Tensión de diseño (Sd): 146605.46875 psi
Tensión máxima permisible (Smax): 139042.96875 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 17.61824683016223
Diámetro medio (Dm):0.5112550360151387 pulgadas
Diámetro exterior (OD): 0.5737550360151387 pulgadas
Diámetro interior (ID): 0.44875503601513866 pulgadas
Razón Crítica (RC): 0.559430855265544
Factor de Wahl (K): 1.0800381064055085
Esfuerzo o tensión en carga de operación (So): 69112.72009856043 psi
Número total de bobinas (N): 2
Longitud comprimido (Ls): 0.125 pulgadas
Fuerza en longitud comprimido (Fs): 21.0 libras
Tensión o esfuerzo en longitud comprimido (Ss): 120947.26017248076 psi
Margen de bobina (cc): 0.5625 pulgadas
Volumen (vol): 0.00985524404987131 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

CASO 2

Diseñe un resorte de compresión para que ejerza una fuerza de 22.0 lb cuando se le comprima hasta que alcance una longitud de 1.75". Cuando su longitud es 3.00" debe ejercer una fuerza de 5.0 lb. El resorte completará ciclos con rapidez y se requiere servicio severo. Utilice alambre de acero ASTM A401.

Sin algoritmos genéticos (Después de 30 iteraciones)

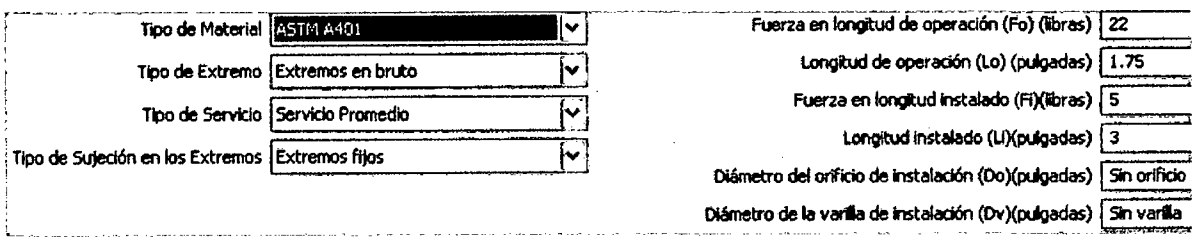


Material: ASTM A401
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi

Extremos del resorte: Extremos en bruto
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 22.0 libras
Longitud de operación (Lo): 1.75 pulgadas
Fuerza en longitud instalado (Fi): 5.0 libras
Longitud instalado (Li): 3.0 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 13.6
Longitud libre (Lf): 3.3676470588235294 pulgadas
Deflexión en longitud de operación (fo): 1.6176470588235294 pulgadas
Diámetro del alambre del resorte (Dw): 0.3625 pulgadas
Tensión de diseño (Sd): 131875.0 psi
Tensión máxima permisible (Smax): 154875.0 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 26.522983850334082
Diámetro medio (Dm): 2.0 pulgadas
Diámetro exterior (OD): 2.3625 pulgadas
Diámetro interior (ID): 1.6375 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.052572715302896
Esfuerzo o tensión en carga de operación (So): 2475.830257122135 psi
Número total de bobinas (N): 2
Longitud comprimido (Ls): 0.725 pulgadas
Fuerza en longitud comprimido (Fs): 35.94 libras
Tensión o esfuerzo en longitud comprimido (Ss): 4044.6063382258876 psi
Margen de bobina (cc): 0.5125 pulgadas
Volumen (vol): 1.2969277033306486 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



Material: ASTM A401
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi
Extremos del resorte: Extremos en bruto
Tipo de servicio: Servicio Promedio

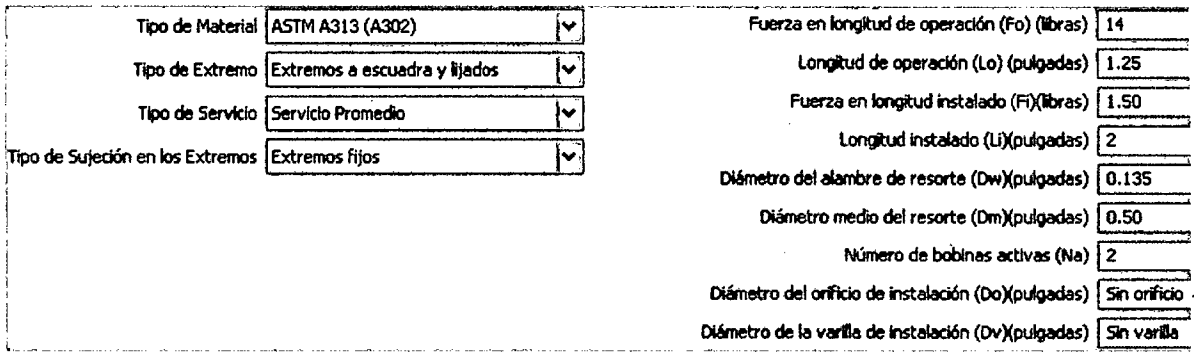
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 22.0 libras
Longitud de operación (Lo): 1.75 pulgadas
Fuerza en longitud instalado (Fi): 5.0 libras
Longitud instalado (Li): 3.0 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 13.6
Longitud libre (Lf): 3.3676470588235294 pulgadas
Deflexión en longitud de operación (fo): 1.6176470588235294 pulgadas
Diámetro del alambre del resorte (Dw): 0.054 pulgadas
Tensión de diseño (Sd): 160990.0 psi
Tensión máxima permisible (Smax): 190395.0 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 14.059966863984618
Diámetro medio (Dm):0.2 pulgadas
Diámetro exterior (OD): 0.254 pulgadas
Diámetro interior (ID): 0.14600000000000002 pulgadas
Razón Crítica (RC): 0.6356727662837107
Factor de Wahl (K): 1.1011686171291595
Esfuerzo o tensión en carga de operación (So): 78354.75139080414 psi
Número total de bobinas (N): 2
Longitud comprimido (Ls): 0.108 pulgadas
Fuerza en longitud comprimido (Fs): 44.331199999999995 libras
Tensión o esfuerzo en longitud comprimido (Ss): 157889.09794800074 psi
Margen de bobina (cc): 0.821 pulgadas
Volumen (vol): 0.0028779766433576566 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

CASO 3

Diseñe un resorte helicoidal de compresión para una válvula de alivio de presión. Cuando la válvula está cerrada, la longitud de resorte es de 2.00" y la fuerza del resorte debe ser 1.50 lb. A medida que se incrementa la presión de la válvula, una fuerza de 14.0 lb hace que se abra la válvula y se comprima el resorte hasta una longitud de 1.25". Utilice alambre de acero resistente a la corrosión ASTM A313 tipo 302.

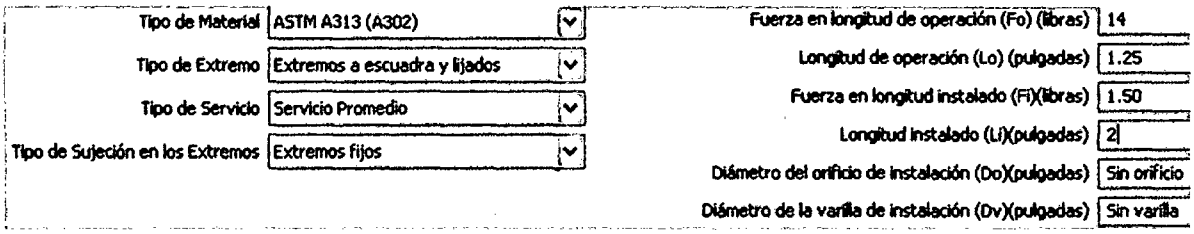
Sin algoritmos genéticos (Después de 30 iteraciones)



Material: ASTM A313 (A302)
Módulo de elasticidad de alambre para resorte en corte (G): 1.0E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 14.0 libras
Longitud de operación (Lo): 1.25 pulgadas
Fuerza en longitud instalado (Fi): 1.5 libras
Longitud instalado (Li): 2.0 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 16.666666666666668
Longitud libre (Lf): 2.09 pulgadas
Deflexión en longitud de operación (fo): 0.8399999999999999 pulgadas
Diámetro del alambre del resorte (Dw): 0.135 pulgadas
Tensión de diseño (Sd): 98046.87499999999 psi
Tensión máxima permisible (Smax): 108703.12499999999 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 17.17071363829998
Diámetro medio (Dm): 0.5 pulgadas
Diámetro exterior (OD): 0.635 pulgadas
Diámetro interior (ID): 0.365 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0821969413766421
Esfuerzo o tensión en carga de operación (So): 7840.489020304032 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.54 pulgadas
Fuerza en longitud comprimido (Fs): 25.833333333333332 libras
Tensión o esfuerzo en longitud comprimido (Ss): 14467.569025561012 psi
Margen de bobina (cc): 0.355 pulgadas
Volumen (vol): 0.08993677010492679 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



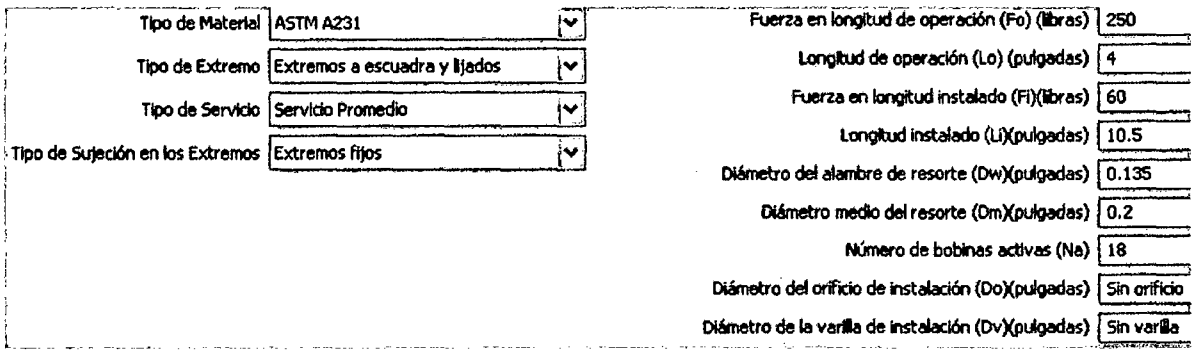
Material: ASTM A313 (A302)
Módulo de elasticidad de alambre para resorte en corte (G): 1.0E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 14.0 libras
Longitud de operación (Lo): 1.25 pulgadas
Fuerza en longitud instalado (Fi): 1.5 libras
Longitud instalado (Li): 2.0 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 16.666666666666668
Longitud libre (Lf): 2.09 pulgadas
Deflexión en longitud de operación (fo): 0.8399999999999999 pulgadas
Diámetro del alambre del resorte (Dw): 0.0625 pulgadas
Tensión de diseño (Sd): 118070.31249999999 psi
Tensión máxima permisible (Smax): 130917.96875 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 13.283232114782638
Diámetro medio (Dm):0.35867903796850203 pulgadas
Diámetro exterior (OD): 0.42117903796850203 pulgadas
Diámetro interior (ID): 0.29617903796850203 pulgadas
Razón Crítica (RC): 0.4177639971069449
Factor de Wahl (K): 1.1073578204055936
Esfuerzo o tensión en carga de operación (So): 57999.20908374978 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.25 pulgadas
Fuerza en longitud comprimido (Fs): 30.666666666666668 libras
Tensión o esfuerzo en longitud comprimido (Ss): 127045.88656440428 psi
Margen de bobina (cc): 0.5 pulgadas
Volumen (vol): 0.013828203952001659 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

CASO 4

Diseñe un resorte helicoidal de compresión que se utilizará para regresar un cilindro neumático a su posición original después que ha sido accionado. A una longitud de 10.50", el resorte debe ejercer una fuerza de 60 lb. A una longitud de 4.00" debe ejercer una fuerza de 250 lb. Se espera servicio severo. Utilice alambre de acero ASTM A231.

Sin algoritmos genéticos (Después de 30 iteraciones)

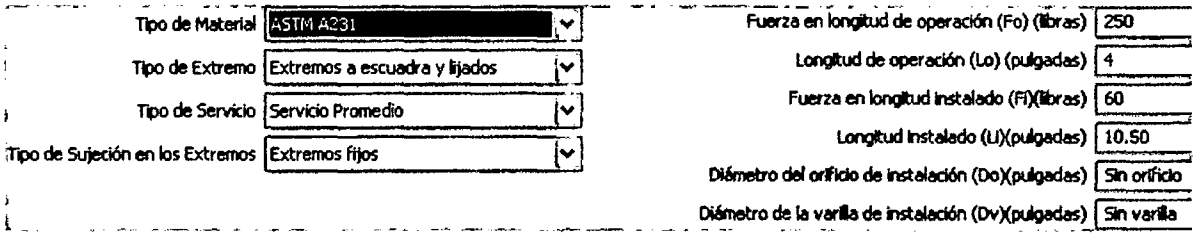


Material ASTM A231
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 250.0 libras
Longitud de operación (Lo): 4.0 pulgadas
Fuerza en longitud instalado (Fi): 60.0 libras
Longitud instalado (Li): 10.5 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 29.23076923076923
Longitud libre (Lf): 12.552631578947368 pulgadas
Deflexión en longitud de operación (fo): 8.552631578947368 pulgadas
Diámetro del alambre del resorte (Dw): 0.135 pulgadas
Tensión de diseño (Sd): 128437.49999999997 psi
Tensión máxima permisible (Smax): 118515.62499999997 psi
Número de bobinas activas (Na): 18
Índice de resorte (C): 7.108582666662789
Diámetro medio (Dm): 0.2 pulgadas
Diámetro exterior (OD): 0.335 pulgadas
Diámetro interior (ID): 0.065 pulgadas
Razón Crítica (RC): 34.75417194607829
Factor de Wahl (K): 1.2092932081346741
Esfuerzo o tensión en carga de operación (So): 62580.701469357984 psi
Número total de bobinas (N): 20
Longitud comprimido (Ls): 2.7 pulgadas
Fuerza en longitud comprimido (Fs): 287.99999999999994 libras
Tensión o esfuerzo en longitud comprimido (Ss): 72092.96809270038 psi
Margen de bobina (cc): 0.07222222222222222 pulgadas
Volumen (vol): 0.1798735402098536 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto

Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



Material: ASTM A231
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 250.0 libras
Longitud de operación (Lo): 4.0 pulgadas
Fuerza en longitud instalado (Fi): 60.0 libras
Longitud instalado (Li): 10.5 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 29.23076923076923
Longitud libre (Lf): 12.552631578947368 pulgadas
Deflexión en longitud de operación (fo): 8.552631578947368 pulgadas
Diámetro del alambre del resorte (Dw): 0.1205 pulgadas
Tensión de diseño (Sd): 131863.125 psi
Tensión máxima permisible (Smax): 122344.53125 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 14.236884034509274
Diámetro medio (Dm):0.2 pulgadas
Diámetro exterior (OD): 0.3205 pulgadas
Diámetro interior (ID): 0.07950000000000002 pulgadas
Razón Crítica (RC): 34.75417194607829
Factor de Wahl (K): 1.099857514947701
Esfuerzo o tensión en carga de operación (So): 80036.00532440329 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.482 pulgadas
Fuerza en longitud comprimido (Fs): 352.83384615384614 libras
Tensión o esfuerzo en longitud comprimido (Ss): 112957.64635759569 psi
Margen de bobina (cc): 1.759 pulgadas
Volumen (vol): 0.028661824660983548 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto

Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

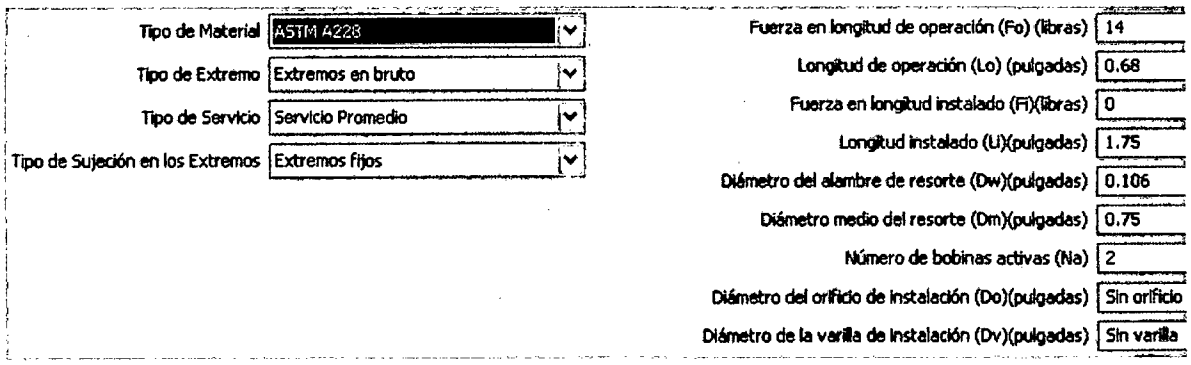
CASO 5

Diseñe un resorte helicoidal de compresión, utilizando alambre para instrumentos musicales, que ejercerá una fuerza de 14.00 lb cuando su longitud sea 0.68". La longitud libre debe ser 1.75". Utilice servicio promedio.

Consideraciones

En este caso la longitud de instalación es igual a la longitud libre con una Fuerza en longitud instalado de 0 lb.

Sin algoritmos genéticos (Después de 30 iteraciones)

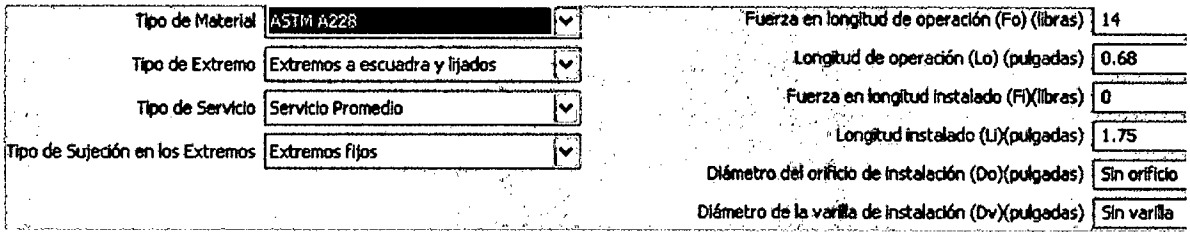


Material: ASTM A228
Módulo de elasticidad de alambre para resorte en corte (G): 1.185E7 psi
Extremos del resorte: Extremos en bruto
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 14.0 libras
Longitud de operación (Lo): 0.68 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 1.75 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 13.084112149532713
Longitud libre (Lf): 1.75 pulgadas
Deflexión en longitud de operación (fo): 1.0699999999999998 pulgadas
Diámetro del alambre del resorte (Dw): 0.106 pulgadas
Tensión de diseño (Sd): 120100.00000000001 psi
Tensión máxima permisible (Smax): 133079.99999999997 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 18.171327609689122
Diámetro medio (Dm): 0.75 pulgadas
Diámetro exterior (OD): 0.856 pulgadas
Diámetro interior (ID): 0.644 pulgadas
Razón Crítica (RC): 1.0E100

Factor de Wahl (K): 1.0775219891316763
Esfuerzo o tensión en carga de operación (So): 24190.116459029097 psi
Número total de bobinas (N): 2
Longitud comprimido (Ls): 0.212 pulgadas
Fuerza en longitud comprimido (Fs): 20.123364485981313 libras
Tensión o esfuerzo en longitud comprimido (Ss): 34770.46646166987 psi
Margen de bobina (cc): 0.23400000000000004 pulgadas
Volumen (vol): 0.04158557814399001 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



Material: ASTM A228
Módulo de elasticidad de alambre para resorte en corte (G): 1.185E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 14.0 libras
Longitud de operación (Lo): 0.68 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 1.75 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 13.084112149532713
Longitud libre (Lf): 1.75 pulgadas
Deflexión en longitud de operación (fo): 1.0699999999999998 pulgadas
Diámetro del alambre del resorte (Dw): 0.051 pulgadas
Tensión de diseño (Sd): 134555.0 psi
Tensión máxima permisible (Smax): 150455.0 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 14.238844983270527
Diámetro medio (Dm):0.3342668782810402 pulgadas
Diámetro exterior (OD): 0.38526687828104017 pulgadas
Diámetro interior (ID): 0.2832668782810402 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0998431733366894
Esfuerzo o tensión en carga de operación (So): 98805.63888772612 psi

Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.204 pulgadas
Fuerza en longitud comprimido (Fs): 20.228037383177575 libras
Tensión o esfuerzo en longitud comprimido (Ss): 142760.29693497627 psi
Margen de bobina (cc): 0.23800000000000004 pulgadas
Volumen (vol): 0.008580911899707501 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

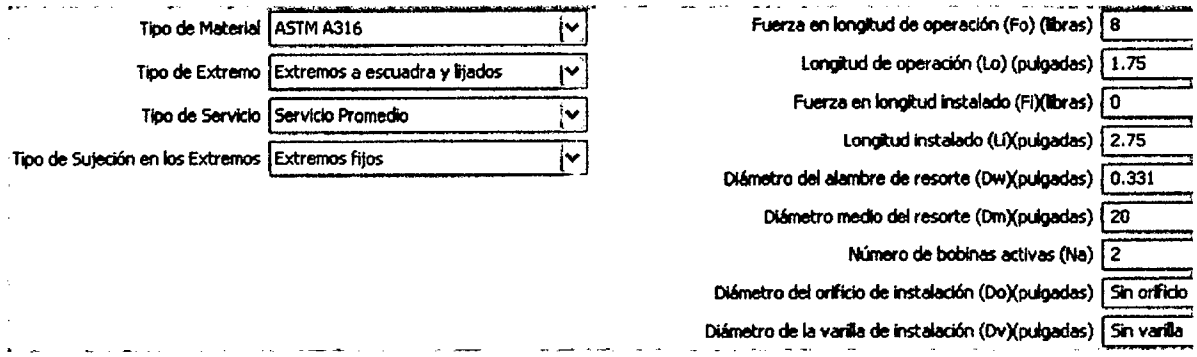
CASO 6

Diseñe un resorte helicoidal de compresión utilizando alambre de acero inoxidable, ASTM A313, tipo 316, para servicio promedio, el cual ejercerá una fuerza de 8.00 lb después de deflexionarse 1.75", a partir de una longitud libre de 2.75".

Consideraciones

Se considera que la longitud de instalación es igual a la longitud libre con una fuerza de instalación de cero libras.

Sin algoritmos genéticos (Después de 30 iteraciones)

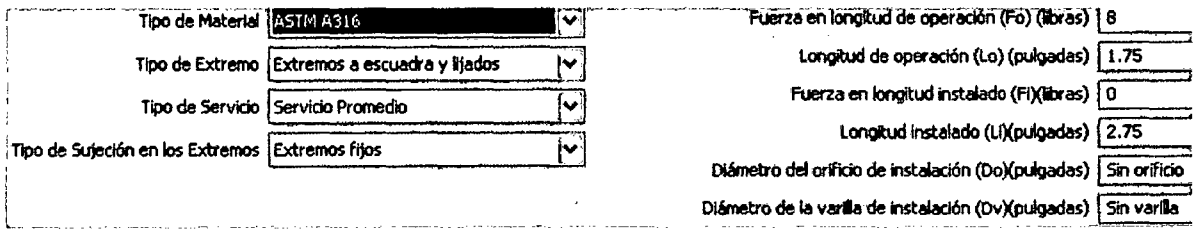


Material: ASTM A316
Módulo de elasticidad de alambre para resorte en corte (G): 1.0E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 8.0 libras
Longitud de operación (Lo): 1.75 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 2.75 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 8.0

Longitud libre (Lf): 2.75 pulgadas
Deflexión en longitud de operación (fo): 1.0 pulgadas
Diámetro del alambre del resorte (Dw): 0.331 pulgadas
Tensión de diseño (Sd): 73197.21875 psi
Tensión máxima permisible (Smax): 66936.4375 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 29.57145371507338
Diámetro medio (Dm): 20.0 pulgadas
Diámetro exterior (OD): 20.331 pulgadas
Diámetro interior (ID): 19.669 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0470470606654736
Esfuerzo o tensión en carga de operación (So): 11763.656039649128 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 1.324 pulgadas
Fuerza en longitud comprimido (Fs): 11.408 libras
Tensión o esfuerzo en longitud comprimido (Ss): 16774.973512539655 psi
Margen de bobina (cc): 0.21299999999999997 pulgadas
Volumen (vol): 21.626474555755024 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



Material: ASTM A316
Módulo de elasticidad de alambre para resorte en corte (G): 1.0E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 8.0 libras
Longitud de operación (Lo): 1.75 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 2.75 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 8.0
Longitud libre (Lf): 2.75 pulgadas
Deflexión en longitud de operación (fo): 1.0 pulgadas

Diámetro del alambre del resorte (Dw): 0.0625 pulgadas
Tensión de diseño (Sd): 100359.76562499999 psi
Tensión máxima permisible (Smax): 111280.27343749999 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 16.965110103718164
Diámetro medio (Dm):0.45632767671834945 pulgadas
Diámetro exterior (OD): 0.5188276767183495 pulgadas
Diámetro interior (ID): 0.39382767671834945 pulgadas
Razón Crítica (RC): 0.3692588342194922
Factor de Wahl (K): 1.0832283098219238
Esfuerzo o tensión en carga de operación (So): 41246.47708802955 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.25 pulgadas
Fuerza en longitud comprimido (Fs): 20.0 libras
Tensión o esfuerzo en longitud comprimido (Ss): 103116.19272007387 psi
Margen de bobina (cc): 0.75 pulgadas
Volumen (vol): 0.017592865806555873 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

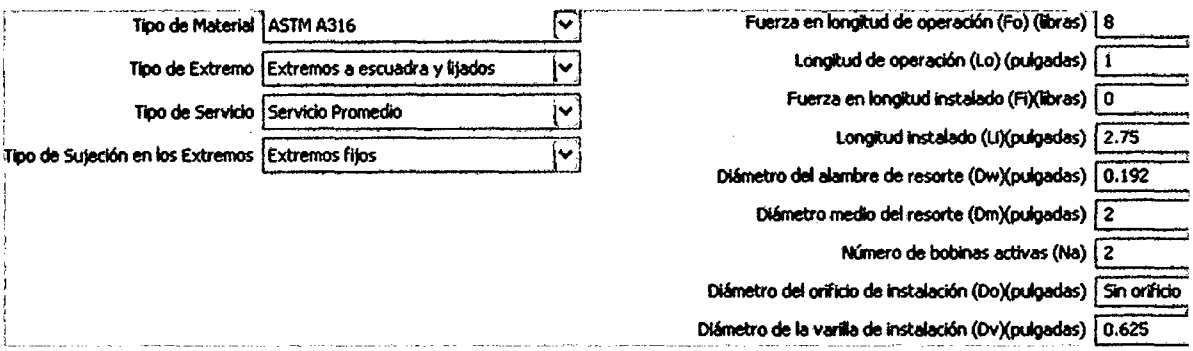
CASO 7

Diseñe un resorte helicoidal de compresión utilizando alambre de acero inoxidable, ASTM A313, tipo 316, para servicio promedio, el cual ejercerá una fuerza de 8.00 lb después de deflexionarse 1.75", a partir de una longitud libre de 2.75". El resorte debe operar alrededor de una varilla cuyo diámetro es 0.625"

Consideraciones

Al deflexionarse 1.75" llegará a una altura de 2.75" – 1.75" = 1" en longitud de operación.

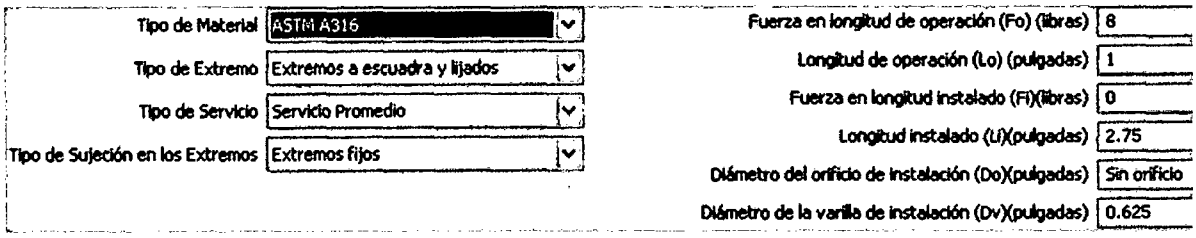
Sin algoritmos genéticos (Después de 30 iteraciones)



Material: ASTM A316
Módulo de elasticidad de alambre para resorte en corte (G): 1.0E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 8.0 libras
Longitud de operación (Lo): 1.0 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 2.75 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): 0.625 pulgadas
Razón de resorte (k): 4.571428571428571
Longitud libre (Lf): 2.75 pulgadas
Deflexión en longitud de operación (fo): 1.75 pulgadas
Diámetro del alambre del resorte (Dw): 0.192 pulgadas
Tensión de diseño (Sd): 73219.0 psi
Tensión máxima permisible (Smax): 81957.00000000001 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 29.719609763815647
Diámetro medio (Dm): 2.0 pulgadas
Diámetro exterior (OD): 2.192 pulgadas
Diámetro interior (ID): 1.808 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.046807968611228
Esfuerzo o tensión en carga de operación (So): 6025.920958928766 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.768 pulgadas
Fuerza en longitud comprimido (Fs): 9.060571428571428 libras
Tensión o esfuerzo en longitud comprimido (Ss): 6824.785908912465 psi
Margen de bobina (cc): 0.11599999999999999 pulgadas
Volumen (vol): 0.7276661932835162 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos:



Material: ASTM A316
Módulo de elasticidad de alambre para resorte en corte (G): 1.0E7 psi

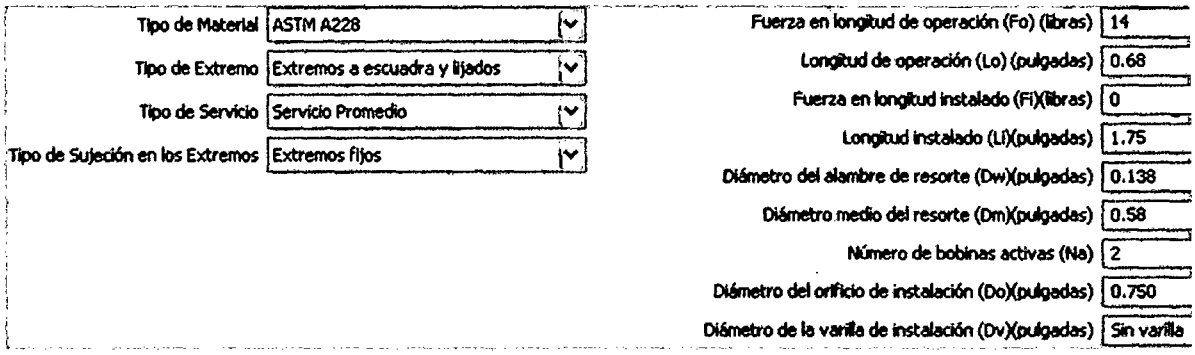
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 8.0 libras
Longitud de operación (Lo): 1.0 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 2.75 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): 0.625 pulgadas
Razón de resorte (k): 4.571428571428571
Longitud libre (Lf): 2.75 pulgadas
Deflexión en longitud de operación (fo): 1.75 pulgadas
Diámetro del alambre del resorte (Dw): 0.0625 pulgadas
Tensión de diseño (Sd): 100359.76562499999 psi
Tensión máxima permisible (Smax): 111280.27343749999 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 20.444164438678687
Diámetro medio (Dm):0.6943462336711025 pulgadas
Diámetro exterior (OD): 0.7568462336711025 pulgadas
Diámetro interior (ID): 0.6318462336711025 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0686539177587515
Esfuerzo o tensión en carga de operación (So): 61916.04619853216 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.25 pulgadas
Fuerza en longitud comprimido (Fs): 11.428571428571427 libras
Tensión o esfuerzo en longitud comprimido (Ss): 88451.49456933164 psi
Margen de bobina (cc): 0.375 pulgadas
Volumen (vol): 0.02676922907703177 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

CASO 8

Diseñe un resorte helicoidal de compresión, utilizando alambre para instrumentos musicales, que ejercerá una fuerza de 14.00 lb cuando su longitud sea 0.68". La longitud libre debe ser 1.75". Utilice servicio promedio. El resorte se instalará dentro de un orificio que tiene 0.750" de diámetro.

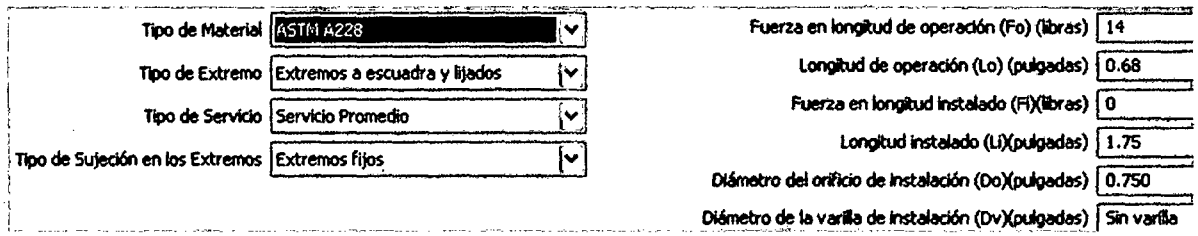
Sin algoritmos genéticos (Después de 30 iteraciones)



Material: ASTM A228
Módulo de elasticidad de alambre para resorte en corte (G): 1.185E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 14.0 libras
Longitud de operación (Lo): 0.68 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 1.75 pulgadas
Diámetro del orificio de instalación (Do): 0.75 pulgadas
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 13.084112149532713
Longitud libre (Lf): 1.75 pulgadas
Deflexión en longitud de operación (fo): 1.0699999999999998 pulgadas
Diámetro del alambre del resorte (Dw): 0.138 pulgadas
Tensión de diseño (Sd): 115420.00000000001 psi
Tensión máxima permisible (Smax): 128240.00000000001 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 19.841647600321007
Diámetro medio (Dm): 0.58 pulgadas
Diámetro exterior (OD): 0.718 pulgadas
Diámetro interior (ID): 0.44199999999999995 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0708008459992968
Esfuerzo o tensión en carga de operación (So): 8424.955002067547 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.552 pulgadas
Fuerza en longitud comprimido (Fs): 15.67476635514019 libras
Tensión o esfuerzo en longitud comprimido (Ss): 9432.80008642703 psi
Margen de bobina (cc): 0.064 pulgadas
Volumen (vol): 0.10901491280432052 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



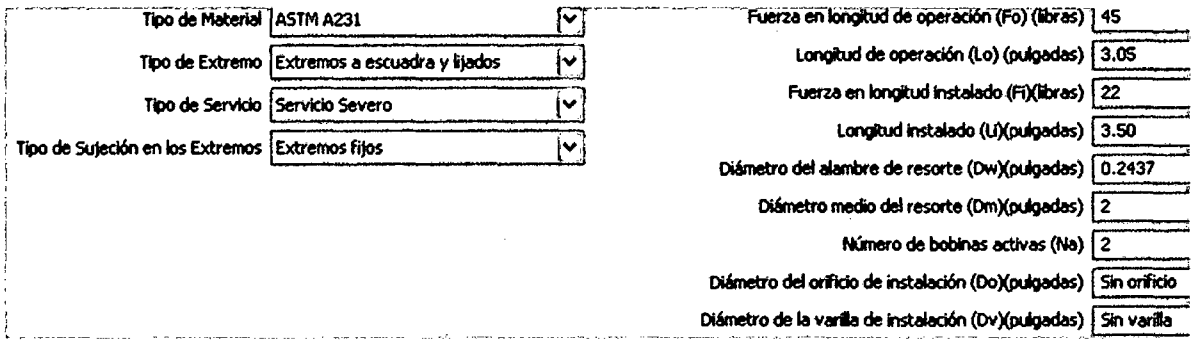
Material: ASTM A228
Módulo de elasticidad de alambre para resorte en corte (G): 1.185E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Promedio
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 14.0 libras
Longitud de operación (Lo): 0.68 pulgadas
Fuerza en longitud instalado (Fi): 0.0 libras
Longitud instalado (Li): 1.75 pulgadas
Diámetro del orificio de instalación (Do): 0.75 pulgadas
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 13.084112149532713
Longitud libre (Lf): 1.75 pulgadas
Deflexión en longitud de operación (fo): 1.0699999999999998 pulgadas
Diámetro del alambre del resorte (Dw): 0.051 pulgadas
Tensión de diseño (Sd): 134555.0 psi
Tensión máxima permisible (Smax): 150455.0 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 14.238844983270527
Diámetro medio (Dm):0.3342668782810402 pulgadas
Diámetro exterior (OD): 0.3852668782810402 pulgadas
Diámetro interior (ID): 0.2832668782810402 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0998431733366894
Esfuerzo o tensión en carga de operación (So): 98805.63888772612 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.204 pulgadas
Fuerza en longitud comprimido (Fs): 20.228037383177575 libras
Tensión o esfuerzo en longitud comprimido (Ss): 142760.29693497627 psi
Margen de bobina (cc): 0.23800000000000004 pulgadas
Volumen (vol): 0.008580911899707501 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

CASO 9

Diseñe un resorte helicoidal de compresión utilizando alambre de acero ASTM A231 para servicio severo, que ejercerá una fuerza de 45.00 lb a una longitud de 3.05" y una fuerza de 22.00 lb a una longitud de 3.50".

Sin algoritmos genéticos (Después de 30 iteraciones)

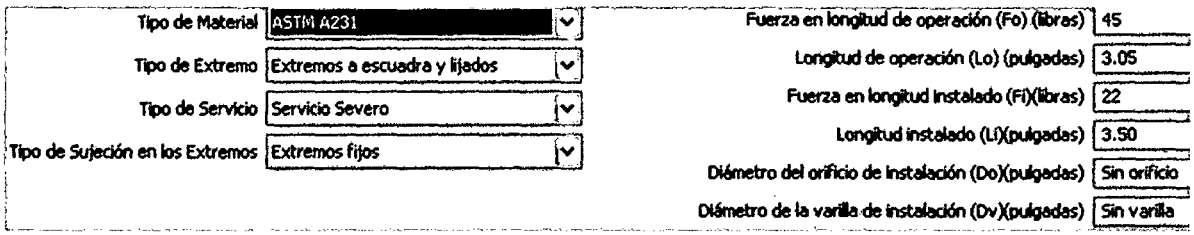


Material: ASTM A231
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Severo
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 45.0 libras
Longitud de operación (Lo): 3.05 pulgadas
Fuerza en longitud instalado (Fi): 22.0 libras
Longitud instalado (Li): 3.5 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 51.11111111111109
Longitud libre (Lf): 3.930434782608696 pulgadas
Deflexión en longitud de operación (fo): 0.8804347826086962 pulgadas
Diámetro del alambre del resorte (Dw): 0.2437 pulgadas
Tensión de diseño (Sd): 93722.5 psi
Tensión máxima permisible (Smax): 105890.38749999998 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 14.944432082458022
Diámetro medio (Dm): 2.0 pulgadas
Diámetro exterior (OD): 2.2437 pulgadas
Diámetro interior (ID): 1.7563 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0949373586370266
Esfuerzo o tensión en carga de operación (So): 17338.251969661524 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.9748 pulgadas
Fuerza en longitud comprimido (Fs): 151.06577777777773 libras
Tensión o esfuerzo en longitud comprimido (Ss): 58204.81153564455 psi
Margen de bobina (cc): 1.0375999999999999 pulgadas
Volumen (vol): 1.1723054916066653 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto

Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



Material: ASTM A231
Módulo de elasticidad de alambre para resorte en corte (G): 1.12E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Severo
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 45.0 libras
Longitud de operación (Lo): 3.05 pulgadas
Fuerza en longitud instalado (Fi): 22.0 libras
Longitud instalado (Li): 3.5 pulgadas
Diámetro del orificio de instalación (Do): Sin orificio
Diámetro de la varilla de instalación (Dv): Sin varilla
Razón de resorte (k): 51.11111111111109
Longitud libre (Lf): 3.930434782608696 pulgadas
Deflexión en longitud de operación (fo): 0.8804347826086962 pulgadas
Diámetro del alambre del resorte (Dw): 0.1055 pulgadas
Tensión de diseño (Sd): 110137.1875 psi
Tensión máxima permisible (Smax): 126125.00000000003 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 11.305203712745671
Diámetro medio (Dm):0.2 pulgadas
Diámetro exterior (OD): 0.3055 pulgadas
Diámetro interior (ID): 0.09450000000000001 pulgadas
Razón Crítica (RC): 1.188120392719819
Factor de Wahl (K): 1.1271784924537191
Esfuerzo o tensión en carga de operación (So): 21999.759759961387 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.422 pulgadas
Fuerza en longitud comprimido (Fs): 179.31999999999994 libras
Tensión o esfuerzo en longitud comprimido (Ss): 87666.598225695 psi
Margen de bobina (cc): 1.3139999999999998 pulgadas
Volumen (vol): 0.021970232877044967 pulgadas cúbicas

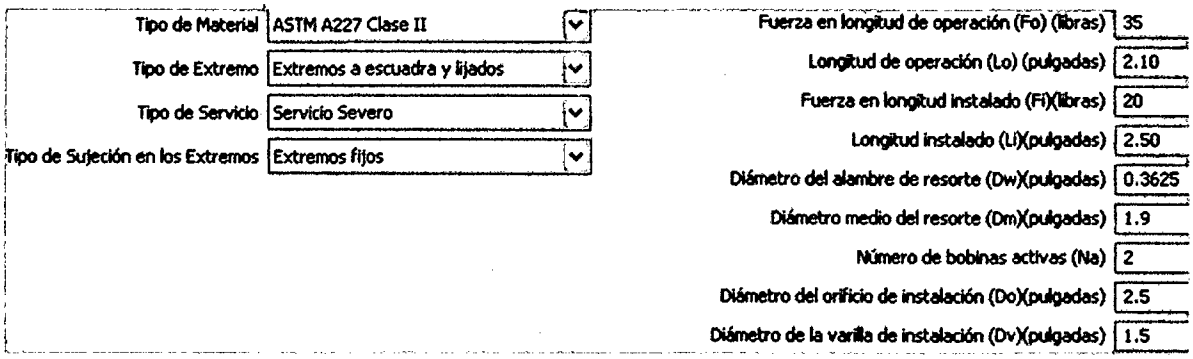
Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo

Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

CASO 10

Diseñe un resorte helicoidal de compresión utilizando alambre redondo de acero, ASTM 227. El resorte activará un embrague y debe soportar múltiples ciclos de operación. Cuando los discos de embrague estén en contacto, el resorte tendrá una longitud de 2.50" y debe ejercer una fuerza de 20 lb. Cuando el embrague no esté activado, la longitud del resorte será de 2.10" y debe ejercer una fuerza de 35 lb. El resorte se instalará alrededor de una flecha redonda cuyo diámetro es 1.50". El diámetro exterior no debe ser mayor a 2. 50".

Sin algoritmos genéticos (después de 53 iteraciones)

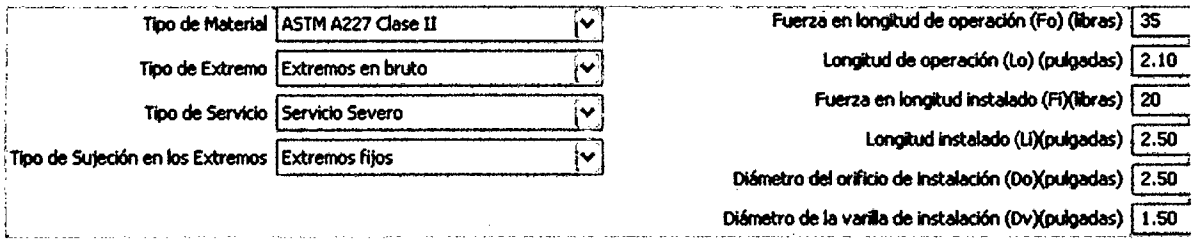


Material: ASTM A227 Clase II
Módulo de elasticidad de alambre para resorte en corte (G): 1.15E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Severo
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 35.0 libras
Longitud de operación (Lo): 2.1 pulgadas
Fuerza en longitud instalado (Fi): 20.0 libras
Longitud instalado (Li): 2.5 pulgadas
Diámetro del orificio de instalación (Do): 2.5 pulgadas
Diámetro de la varilla de instalación (Dv): 1.5 pulgadas
Razón de resorte (k): 37.50000000000001
Longitud libre (Lf): 3.033333333333333 pulgadas
Deflexión en longitud de operación (fo): 0.9333333333333331 pulgadas
Diámetro del alambre del resorte (Dw): 0.3625 pulgadas
Tensión de diseño (Sd): 73347.65625 psi
Tensión máxima permisible (Smax): 95874.99999999999 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 19.08174994119393
Diámetro medio (Dm): 1.9 pulgadas
Diámetro exterior (OD): 2.2624999999999997 pulgadas
Diámetro interior (ID): 1.5374999999999999 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.0737080346711698
Esfuerzo o tensión en carga de operación (So): 3817.0155558385095 psi
Número total de bobinas (N): 4

Longitud comprimido (Ls): 1.45 pulgadas
Fuerza en longitud comprimido (Fs): 59.37500000000001 libras
Tensión o esfuerzo en longitud comprimido (Ss): 6475.2942465117585 psi
Margen de bobina (cc): 0.32500000000000007 pulgadas
Volumen (vol): 2.4641626363282323 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

Con algoritmos genéticos



Material: ASTM A227 Clase II
Módulo de elasticidad de alambre para resorte en corte (G): 1.15E7 psi
Extremos del resorte: Extremos a escuadra y lijados
Tipo de servicio: Servicio Severo
Tipo de sujeción en los extremos del resorte: Extremos fijos
Fuerza en longitud de operación (Fo): 35.0 libras
Longitud de operación (Lo): 2.1 pulgadas
Fuerza en longitud instalado (Fi): 20.0 libras
Longitud instalado (Li): 2.5 pulgadas
Diámetro del orificio de instalación (Do): 2.5 pulgadas
Diámetro de la varilla de instalación (Dv): 1.5 pulgadas
Razón de resorte (k): 37.50000000000001
Longitud libre (Lf): 3.033333333333333 pulgadas
Deflexión en longitud de operación (fo): 0.9333333333333331 pulgadas
Diámetro del alambre del resorte (Dw): 0.162 pulgadas
Tensión de diseño (Sd): 84732.50000000001 psi
Tensión máxima permisible (Smax): 114200.00000000001 psi
Número de bobinas activas (Na): 2
Índice de resorte (C): 14.58883239328284
Diámetro medio (Dm):1.6830387010133074 pulgadas
Diámetro exterior (OD): 1.8450387010133074 pulgadas
Diámetro interior (ID): 1.5210387010133075 pulgadas
Razón Crítica (RC): 1.0E100
Factor de Wahl (K): 1.097347912542002
Esfuerzo o tensión en carga de operación (So): 38716.98760324354 psi
Número total de bobinas (N): 4
Longitud comprimido (Ls): 0.648 pulgadas
Fuerza en longitud comprimido (Fs): 89.45 libras
Tensión o esfuerzo en longitud comprimido (Ss): 98949.55831743243 psi

Margen de bobina (cc): 0.726 pulgadas
Volumen (vol): 0.4359371464244978 pulgadas cúbicas

Penalización acumulada (pen): 0.0
Factor de Wahl: Correcto
Índice de resorte: Correcto
Diámetro exterior: Correcto
Diámetro interior: Correcto
Pandeo: Sin pandeo
Esfuerzo o tensión en carga de operación: Correcto
Longitud comprimido: Correcto
Tensión o esfuerzo en longitud comprimido: Correcto
Margen de bobina: Correcto
Número de espiras activas: Correcto

J. NOMENCLATURA UTILIZADA

Capítulo I

δ	deflexión
Q	número de espiras inactivas en ambos extremos
P	carga
τ	esfuerzo
B	constante
c	índice del resorte
k_s	factor de esfuerzo para cortante transversal
d	diámetro de alambre de prueba
C	índice razonable de resorte
D	diámetro de la espira
N_a	número de espiras activas
K_S	factor de cortante directo
K_w	factor Wahl a la torsión
N_s	factor de seguridad para una carga estática
S_{ys}	límite elástico al cortante
τ	esfuerzo cortante
y	deflexión
G	módulo de corte del material
D_i	Diámetro interior
D_o	Diámetro exterior
F_{min}	carga fluctuante mínima
F_{max}	carga fluctuante máxima
F_a	Fuerza alternante
F_m	Fuerza media
R_F	razón de fuerzas

x	$\{x_1,\ldots,x_N\}$ vector de números reales

Capítulo II

EAs	Algoritmos Evolutivos
EC	Computación Evolutiva
GAs	Algoritmos Genéticos
t	tiempo en un momento dado
P(t)	población de individuos en un tiempo t
S	estructura de dato (esquema de representación)
x_i^t	Solución i en el tiempo t
m_i	Transformación unaria de mutación
c_j	Transformación de cruce de alto orden
m	longitud de cada esquema
$\xi(S,t)$	Número de cadenas en una población en el tiempo t
$\delta(S)$	Longitud del esquema S – la distancia entre la primera y última posición de la cadena fija
$o(S)$	Orden del esquema S – el número de posiciones 0 y 1 presentes en el esquema
$eval(S,t)$	Capacidad promedio de todas las cadenas en la población que coinciden con el esquema S
$F(t)$	Capacidad total de toda la población en el tiempo t
p_c	Probabilidad de cruce
p_m	Probabilidad de mutación
$v=(x,\sigma)$	Cromosoma, el primer vector x representa un punto en el espacio de búsqueda; el segundo vector σ es de desviaciones estándar
$N(0,\sigma)$	Vector de números aleatorios gaussianos independientes con una media igual a cero y desviación estándar σ
f	Función objetivo a maximizar
ES	Estrategia evolutiva
μ	Número de individuos de una población
λ.	Tamaño de la descendencia
EP	Programación evolutiva

a_i	Evento i
S_i	Estado i
tam_pob	Tamaño de la población
GP	Programación Genética o Programa Genético
e_1	estructura i
ADF	Funciones Definidas Automáticamente
sGA	Algoritmo Genético Estructurado
ADN	Acido Desoxirribonucleico
A	Adenina
T	Timina
C	Citosina
G	Guanina
$f(x)$	función de capacidad (aptitud)
b_i	alelo i, puede ser 0 o 1
c	es la cardinalidad del alfabeto
l	es la longitud de la cadena
mGA	Algoritmo genético desordenado
PMX	Cruce de mapeamiento parcial
OX	cruce de orden
CX	cruce de ciclo
$\oplus$	Operador XOR (O exclusivo)
$f(x)$	función f de x
$g(x)$	función g de x
c	factor de penalización definido por el usuario
$R_{i,j}$	coeficientes de penalización
l	nivel de violación definido por el usuario
C, α, β	constantes definidas por el usuario
τ	hora de enfriamiento
f_{peor}	valor de la función objetivo de la peor solución factible de la población
OD	Diámetro exterior
ID	Diámetro interior
D_w	Diámetro del alambre.

D_m	Diámetro medio
L_f	Longitud libre
L_s	Longitud Comprimido
L_o	Longitud de operación
L_i	Longitud instalado
F_s	Fuerza en longitud comprimido
F_o	Fuerza en longitud de operación
F_i	Fuerza en longitud instalado
F_f	Fuerza en longitud libre
k	razón de resorte
C	índice de resorte
N	número total de bobinas en un resorte
N_a	número de bobinas activas del resorte
p	espaciamiento del resorte
λ	ángulo de espaciamiento
OD_s	diámetro exterior real del resorte en longitud de comprimido
ID	diámetro interior del resorte
cc	margen de bobina
τ	tensión por esfuerzo de corte por torsión
T	par de torsión
c	distancia desde el eje neutro hasta la fibra externa
K	El factor de Wahl
θ	ángulo de combadura en radianes
T	torque aplicado
L	longitud del alambre
J	momento de inercia polar del alambre
f	deflexión lineal
G	módulo de elasticidad de alambre para resorte en corte
E	módulo de elasticidad de alambre para resorte en Tensión
f_o	deflexión de operación
τ_o	esfuerzo o tensión que se espera con la carga de operación
τ_s	tensión o esfuerzo en longitud comprimido
τ_d	tensión de diseño

τ_{max}	tensión máxima permisible con el esfuerzo en longitud comprimido
$D_{orificio}$	diámetro del orificio
$D_{varilla}$	diámetro de la varilla
IEEE	Instituto de Ingenieros Eléctricos y Electrónicos
S	Bit de signo
E	Bit del exponente
M	Bit de la mantisa
NaN	No es un número
Lf	Longitud libre
OD	Diámetro exterior
ID	Diámetro interior
Dw	Díámetro del alambre
Dm	Díámetro medio
Fs	Fuerza en longitud comprimido
Li	Longitud instalado;
Fi	Fuerza en longitud instalado
Ff	Fuerza en longitud libre
K	Razón de resorte
C	Índice de resorte
N	Número total de bobinas
Na	Número de bobinas activas
Namax	Número máximo de bobinas activas
p	Espaciamiento
ae	Angulo de espaciamiento
ODs	Diámetro exterior en longitud comprimido
cc	Margen de bobina
K	Factor de Wahl
Fo	Fuerza en longitud de operación
So	Esfuerzo o tensión en carga de operación
fo	Deflexión en longitud de operación
Lo	Longitud de operación
Ls	Longitud comprimido

Ss	Tensión o esfuerzo en longitud comprimido
Sd	Tensión de diseño
Smax	Tensión máxima permisible
Do	Diámetro del orificio de instalación
Dv	Diámetro de la varilla de instalación
RC	Razón Crítica
vol	Volúmen
pen	Penalización