UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA INDUSTRIAL Y DE SISTEMAS

OPTIMIZACION EN PROBLEMAS DE ASIGNACIÓN DE RECURSOS Y TAREAS USANDO COMPUTACIÓN EVOLUTIVA Y MULTIAGENTES INTELIGENTES

TESIS

Para optar el título profesional de

INGENIERO DE SISTEMAS

LEON SUEMATSU, Yutaka I.

MONTOYA VALENCIA, Wilberto W.

Lima - Perú

2002

OPTIMIZACIÓN EN PROBLEMAS DE ASIGNACIÓN DE RECURSOS Y TAREAS USANDO COMPUTACION EVOLUTIVA Y MULTIAGENTES INTELIGENTES

Dedicado a nuestras madres,

ISABEL y DONATILA

AGRADECIMIENTOS

A nuestros asesores:

Ing. LUIS ZULOAGA ROTTA

Lic. CELESTINA PEÑA QUIÑÓNEZ

Ing. BENITO ZARATE OTAROLA

Por la asesoría brindada en el desarrollo de la presente tesis.

Al

Ing. ROBERTO EYZAGUIRRE TEJADA

Por el apoyo brindado en el planeamiento del problema.

SUMARIO

Uno de los objetivos de la presente tesis es resolver el problema de Optimización en la Asignación de Recursos y Tareas, el cual es un problema común que enfrentan todas las empresas e industrias en las diferentes áreas de negocios. Para abordar este problema planteamos tres nuevos enfoques basados en metodologías de la Computación Evolutiva y Sistemas Multi-Agentes Inteligentes.

Estos enfoques muestran una gran eficiencia y eficacia al encontrar las mejores soluciones posibles para los casos de estudio planteados. Demostrándose de esta manera, que es factible y recomendable el uso de la Inteligencia Artificial, debido a que ofrece grandes ventajas competitivas, tales como mayor rapidez y mejor calidad de las soluciones, así como una mayor adaptabilidad a los cambios, permitiendo de ésta manera una reducción de costos y mejora del servicio.

DESCRIPTORES TEMÁTICOS

- Inteligencia Artificial
- Computación Evolutiva
- Algoritmos Genéticos
- Estrategias Evolutivas
- Multiagentes Inteligentes
- Asignación de Recursos y Tareas

CONTENIDO

SUMARIO **1**

DESCRIPTORES TEMÁTICOS **2**

CONTENIDO **3**

INTRODUCCION **6**

OBJETIVOS 10

ANTECEDENTES 10

ALCANCES 12

ESTRUCTURA 12

1. FUNDAMENTO TEÓRICO **14**

1.1 ALGORITMOS GENETICOS 17

1.1.1 La Población *18*

1.1.2 Función Fitness *20*

1.1.3 Operadores Genéticos *23*

1.1.4 Proceso Evolutivo *27*

1.2 ESTRATEGIAS EVOLUTIVAS 28

1.2.1 Población *29*

1.2.2 Función objetivo *30*

1.2.3 Estrategia Evolutiva (1+1) *30*

1.2.4 Estrategia Evolutiva (μ, λ) *32*

1.2.5 Algoritmo Básico *32*

1.2.6 Generación de Descendientes *33*

1.2.7 Proceso de Evolución *35*

1.3 AGENTES INTELIGENTES 36

1.3.1 Como los agentes deben actuar *36*

1.3.2 Mapeo ideal de las secuencias de percepción a acciones *38*

1.3.3 Autonomía *39*

1.3.4 Estructura de los agentes inteligentes *40*

1.4 SISTEMAS CLASIFICADORES DE APRENDIZAJE 51

1.4.1 Descripción de un Sistema Clasificador *55*

1.4.2 Funcionamiento básico de un SC *64*

2. DESCRIPCIÓN DEL PROBLEMA 67

2.1 CARACTERÍSTICAS 69

2.2 CASOS DE ESTUDIO 72

2.3 CUANTIFICACION DE LOS CASOS DE ESTUDIO 72

2.3.1 Caso 1 ..... *72*

2.3.2 Caso 2 ..... *74*

3. DESCRIPCIÓN DEL MODELO 78

3.1 MÉTODO DE BÚSQUEDA EXHAUSTIVA 80

3.1.1 Complejidad de la búsqueda *80*

3.2 MÉTODO DE ALGORITMOS GENETICOS 83

3.3 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES 84

3.4 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES Y ESTRATEGIAS EVOLUTIVAS 88

4. EVALUACION DE RESULTADOS 91

4.1 MÉTODO DE BÚSQUEDA EXHAUSTIVA 91

4.1.1 Resolución del Caso 1 94

4.1.2 Resolución de Caso 2 95

4.2 MÉTODO DE ALGORITMOS GENETICOS 96

4.2.1 Resolución Caso 1 96

4.2.2 Resolución Caso 2 103

4.3 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES 113

4.3.1 Resolución Caso 1 113

4.3.2 Resolución Caso 2 122

4.4 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES Y ESTRATEGIAS EVOLUTIVAS 129

4.5 DETERMINACION DE LA EFICIENCIA DE LOS METODOS 136

4.5.1 Método de Algoritmos Genéticos, Resolución Caso 3 138

4.5.2 Método de Sistemas Multiagentes con Sistemas Clasificadores Resolución Caso 3 152

5. CONCLUSIONES Y RECOMENDACIONES 171

APENDICES 178

TEOREMA DE SCHEMADATA 178

BIBLIOGRAFÍA 184

INTRODUCCION

En un mundo en constante cambio, la capacidad de adaptación es muy importante para poder sobrevivir. Desde el inicio de la vida en la tierra, gracias al proceso evolutivo, todas las criaturas han tenido que adaptarse, y muchas veces hacer cambios fisiológicos que les dotaron de capacidades necesarias para su subsistencia, tal como sostiene la teoría de Selección Natural de Charles Darwin. El proceso evolutivo, permite la aparición de criaturas cada vez más complejas e "inteligentes", dotadas de capacidades de aprendizaje, organización, cooperación y adaptación a su medio ambiente.

Estos sistemas naturales han sido a lo largo de la historia una gran fuente de inspiración de pensadores y científicos, por ejemplo, entender el proceso de aprendizaje, simular comportamientos naturales con medios artificiales entre otros.

La Inteligencia Artificial es un tema que interesó a muchos desde hace siglos, tratando de entender y simular las capacidades cognoscitivas humanas, pero fue desde la aparición del primer computador que tomó la importancia necesaria. En los 50's tomó el nombre de "Inteligencia Artificial" y surgen dos corrientes, las cuales están inspiradas en:

1. La obtención de resultados similares a los que se pueden obtener los procesos naturales. En esta corriente podemos agrupar a los Algoritmos de Búsqueda (Sin Información y las Heurísticas), Sistemas Expertos y otros.

2. La imitación de procesos naturales para la solución de problemas que son difíciles de resolver con técnicas tradicionales. En esta corriente podemos agrupar a las Redes Neuronales, Computación Evolutiva, Agentes Inteligentes y otros.

En los orígenes de la Inteligencia Artificial, la primera corriente tuvo mayor acogida por la comunidad científica, pero en estas dos ultimas décadas, la segunda corriente se está desarrollando con mayor interés, debido fundamentalmente a la complejidad y limitado dominio de problemas que pueden ser resueltos por los métodos de la primera corriente, sumados a la participación de científicos y expertos en las diferentes ramas de las ciencias naturales y sociales. A la fecha, el desarrollo de la Inteligencia Artificial ha logrado grandes avances, y las aplicaciones son muy diversas, desde sistemas netamente matemáticos hasta sistemas psicológicos y sociales. En los últimos años, las técnicas que tienen mayor acogida son:

- *Redes Neuronales Artificiales*: Que son técnicas inspiradas en la forma cómo las neuronas cerebrales trabajan en procesos de aprendizaje, reconocimiento de patrones, control de acciones y otros.

- *Computación Evolutiva*: Algoritmos basados en la forma cómo entes naturales logran adaptarse a los cambios por medio del proceso de evolución. Estas técnicas son muy usadas en procesos de optimización, creación de programas autónomos

con capacidades de auto-modificación y adaptación, estudios de biología evolutiva entre otras aplicaciones.

- *Agentes Inteligentes:* Permite el estudio de sistemas complejos por medio de simulación de sistemas inteligentes distribuidos, esta área es usada para entender sistemas sociales, sistemas organizacionales entre otros

- *Machine learning:* Conjunto de algoritmos de aprendizaje aplicados en diferentes tipos de problemas, como en el control de acciones de autómatas en ambientes conocidos y desconocidos.

- *Autómata Celular:* modelos matemáticos de máquinas de estado finito.

Entre las áreas de estudio donde se usan estas técnicas podemos mencionar:

- *Lenguaje Natural*: Aplicación de algoritmos para simular la forma como los humanos pueden comunicarse entre sí por medio de un lenguaje, esto permite una comunicación mas fluida entre el hombre y la máquina. Estas técnicas son usadas para la creación de sistemas de interacción humana, así como en la creación de robots sociales.

- *Vida Artificial*: Estudia la vida "natural" con el fin de recrear fenómenos biológicos en medios artificiales. La vida artificial trata de entender teóricamente el fenómeno a estudiar, así como los principios biológicos para la creación de aplicaciones prácticas en la industria y la tecnología.

- *Simulación de Sistemas Sociales*: Aplicación de algoritmos para simular el comportamiento de sistemas sociales como resultado de las interacciones entre los agentes que participan en el modelo. Entre los ejemplos más comunes están las simulaciones de sistemas económicos, sistemas organizacionales entre otros.

Estas técnicas son de gran utilidad para crear sistemas más inteligentes, con capacidad de aprendizaje y toma de decisiones en algunos puntos críticos del sistema, lo que hace que las empresas logren mayor eficiencia, dejando a los empleados las tareas importantes. Dotándoles de capacidades de evolución y adaptación, el sistema puede, en forma automática y autónoma, modificarse para soportar nuevos cambios en el ambiente. Por estas razones, la Inteligencia Artificial es una de las herramientas tecnológicas más importantes, que debería ser considerada por empresas, industrias y científicos.

En el Perú la Inteligencia Artificial aún no es utilizada, y solo es considerado su estudio muy superficialmente por algunas universidades. La mayoría de los interesados que quieren profundizar sus conocimientos es esta área, optan por realizar estudios en el extranjero o aprender en forma autodidacta..

La presente tesis pretende mostrar el gran potencial que tienen los sistemas basados en Inteligencia Artificial, para atacar problemas que enfrentan empresas e industrias.

OBJETIVOS

- Demostrar el potencial del uso de las técnicas de Inteligencia Artificial para la solución de problemas de asignación de recursos y tareas.
- Describir y comparar diversos enfoques basados en Inteligencia Artificial para el diseño de sistemas de optimización en asignación de recursos y tareas.
- Resaltar la robustez de los sistemas basados en Inteligencia Artificial para afrontar modificaciones en el problema planteado.

ANTECEDENTES

Muchos problemas que enfrentan diversas empresas están directamente relacionados con la dificultad que tienen en adaptarse, para enfrentar los nuevos retos. Nuestro mundo cambiante obliga a tener estructuras y organizaciones flexibles, capaces de cambiar según las necesidades. El adecuado uso del computador en todas las áreas del negocio, es un factor fundamental para tener empresas más efectivas.

Una persona ó empresa es más eficiente, si sabe manejar en forma efectiva la información. Para esto, muchas empresas tienen sistemas computacionales que les ayudan desde el control de asistencia hasta el control automatizado de la producción. El diseño de sistemas es un factor

muy importante para la efectividad de la aplicación y la capacidad de soportar cambios en el entorno de acción.

Los grandes problemas al usar diseños de sistemas tradicionales son:

- Rigidez del sistema, no son fáciles de hacer modificaciones.
- Hace difícil la tarea de modificación por personal nuevo.
- El adicionar alguna nueva variable, hace muchas veces que se tenga que cambiar todo el sistema.
- Cualquier cambio tiene que ser hecho por personal calificado.
- En caso de anomalías, sólo personal calificado puede resolver el problema.

Algunos de éstos problemas se solucionarían fácilmente con métodos de Inteligencia Artificial, lo que da capacidades de adaptación, aprendizaje y coordinación entre sistemas.

Para mostrar estas ventajas, ésta tesis utilizará el potencial de métodos tales como la computación evolutiva y multiagentes inteligentes, los cuales pueden resuelve problemas en las diferentes áreas de un negocio. La aplicación está hecha para optimizar la asignación de recursos y tareas, problema que tiene gran utilidad en industrias y empresas; el diseño no solo brindará la solución óptima, sino facilitará actuar en casos de anomalías brindando nuevas soluciones óptimas por medio de la reutilización del conocimiento obtenido previamente.

ALCANCES

- Definir modelos de optimización en la asignación de recursos y tareas
- Hacer uso de técnicas evolutivas en ambientes multiagentes para la implementación del diseño de nuestro sistema de optimización: Algoritmos Genéticos, Estrategias Evolutivas, Sistemas Multi-Agentes, Sistemas Clasificadores.

ESTRUCTURA

La presente tesis esta organizada de la siguiente manera: En el capítulo 1 se explica los fundamentos teóricos de todos los métodos estudiados para la implementación de los modelos planteados; debido a la complejidad de estos métodos, se optó, en la medida de lo posible, por describirlos detallada y didácticamente, a fin de que el lector de esta tesis cuente con el conocimiento mínimo necesario para captar fácilmente los modelos planteados, que de por sí poseen cierto grado de complejidad. En el capítulo 2 se describe el problema de estudio de la presente tesis, indicando las características de éste. En el capítulo 3 se describen los cuatro modelos estudiados, así como la determinación del grado de complejidad de los problemas a ser resueltos. La evaluación de los resultados de los diferentes experimentos realizados es discutida en el capitulo 4. Finalmente, las conclusiones y recomendaciones serán dadas en el capítulo .

CAPITULO I

FUNDAMENTO TEORICO

1. FUNDAMENTO TEÓRICO

En este capitulo se hará una breve reseña de los dos métodos que van a ser usados para resolver el problema de optimización planteado en esta tesis. Estos métodos son:

- Computación Evolutiva
- Agentes Inteligentes

La primera, es un conjunto de metodologías de búsqueda basadas en el proceso de *Evolución Natural*, que permiten hallar una solución a un problema por medio de la adaptación de los individuos. Daremos una breve explicación de las metodologías más importantes de la Computación Evolutiva como son los Algoritmos Genéticos (Genetic Algoritms - GA), Sistemas Clasificadores (Clasifier Systems - CS) y Estrategias Evolutivas (Evolution Strategies - ES), asimismo, presentaremos en el apéndice la base matemática del Schema-Data, que explica la convergencia de los Algoritmos Genéticos, el que es considerado como el fundamento formal del resto de metodologías evolutivas.

Los agentes inteligentes, son metodologías de inteligencia artificial para el modelamiento de sistemas complejos, donde el sistema es representado por uno o

varios elementos individuales llamados agentes, que interactúan entre sí para cumplir con un objetivo o meta global. Daremos una breve explicación de las metodologías más importantes como son Agente Autónomos (Autonomous Agent AA) y Sistemas Multi-Agentes (Multi-Agent System – MAS).

Antes de detallar los métodos usados, explicaremos un marco general para la ubicación de éstos métodos:

La computación evolutiva se ubica en las técnicas de búsqueda, las que a su vez son clasificadas en tres categorías básicas.

1. Enumerativo. Todas las posibles soluciones son generadas y probadas para encontrar la solución óptima. Esto requiere un cómputo excesivo en problemas que involucran un gran número de variables.

2. Aleatorios. Las soluciones son halladas en forma casual, cuando se generan posibles soluciones en forma aleatoria.

3. Clásico o cálculo-basado. Estas usan un acercamiento determinístico para encontrar una solución buena, aplicándose a los problemas bien estudiados, debido a que requieren de la información necesaria para asociar un valor a cada nodo de la búsqueda, éste valor da una idea de cuan cerca se encuentra de la solución. Es decir, este método requiere el conocimiento de la pendiente o derivadas de orden superior comúnmente llamada función heurística, la que define la dirección de la búsqueda. En esta categoría se encuentran todos los métodos de búsqueda heurística.

4. Seudo-aleatorios. Son métodos de búsqueda intermedios entre los enumerativos y los clásicos. La búsqueda se realiza sobre todo el espacio de soluciones, sin realizar una búsqueda exhaustiva, sino que usa información adicional para realizar pequeños cambios a las mejores soluciones, que le permiten encontrar una mejor solución. Los algoritmos evolutivos son ejemplos típicos de esta clase de métodos de búsqueda.

La ventaja de los sistemas de búsqueda seudo-aleatorios es que pueden abordar problemas de gran complejidad con mayor éxito que los métodos clásicos, que fallan debido a la imposibilidad de analizar las pendientes por el gran número de variables involucradas, y es mejor que los enumerativos porque no necesitan evaluar todas las posibilidades.

Agentes inteligentes se ubica en el modelamiento de sistemas complejos, los que pueden ser clasificados en dos categorías básicas.

1. Sistemas Centralizados. Donde todo el sistema está modelado en un solo módulo llamado control central, el que evalúa el estado actual del sistema y determina las acciones que éste debe realizar, para cumplir con el objetivo o meta del sistema. Muchas veces estos sistemas son muy complejos para implementar la toma de decisiones del agente, en un solo módulo. Los algoritmos heurísticos tradicionales, así como la mayoría de las metodologías usadas en la inteligencia artificial pertenecen a este grupo.

2. Sistemas Distribuidos. Donde el sistema esta modelado en varios módulos independientes, los cuales interactúan entre sí, tratando de cumplir sus metas individuales y a la vez dando origen a una sinergia para cumplir con los objetivos globales o metas del sistema.

1.1 ALGORITMOS GENETICOS

El fracaso de las técnicas de optimización tradicionales para búsquedas con restricciones complejas, a generado el planteamiento de métodos alternativos; por esta razón, los métodos evolutivos han ganado gran popularidad como métodos de optimización de uso general y como técnicas de búsqueda.

Los métodos evolutivos son inspirados en el proceso de evolución natural, donde solo los mejores sobreviven. Estos métodos empezaron a ser estudiados desde los años 60, y entre los primeros métodos estudiados están los algoritmos genéticos, que fue propuesto por Holland.

Los métodos evolutivos usan mecanismos de búsqueda probabilística, para localizar el óptimo global en un entorno multimodal comúnmente llamado espacio de soluciones. Siendo métodos auto-adaptativos de las estrategias de búsqueda, basado en la exploración seudo aleatoria del espacio de soluciones, acoplada con un componente de memoria (población) que le permite aprender el camino de búsqueda óptima.

Los AG son los métodos más representativos y ampliamente usados dentro de los algoritmos evolutivos, y se usan específicamente para tratar problemas que involucran espacios de búsqueda grandes, que contienen mínimos o máximos locales (soluciones no necesariamente optimas) múltiples.

1.1.1 La Población

El punto de partida de todo algoritmo evolutivo es la población. Una población es un conjunto de individuos homogéneos, es decir individuos con estructuras genéticas idénticas, los que a su vez representan una posible solución al problema. Estos individuos están compuestos por genes que asumen varios valores llamados alelos. Al valor de un gen se le llama valor alélico, y normalmente éste se restringe a los valores {0,1}.

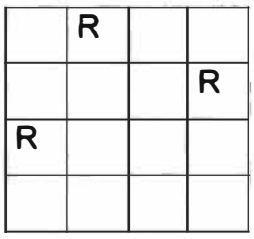
Estos individuos son representados como cadenas binarias de longitud fija, por ejemplo:

"10001011101"

Si la cadena binaria tiene longitud N, entonces pueden generarse 2^n posibles cadenas binarias. Cada una de éstas, representa al individuo, y estos a su vez representan un punto de la búsqueda en el espacio de soluciones potenciales en un problema de optimización dado.

Por ejemplo, para resolver el problema típico de las n reinas, donde en un tablero de nXn se deben ubicar n reinas, de manera que ninguna de ellas se haga "jaque" a otra, se usará el caso simple de 4 reinas.

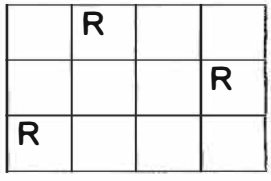
La primera tarea es identificar al individuo, el que representa un punto en el espacio de soluciones del problema y está compuesto por un número fijo de genes. En un primer modelo, los genes podrían representar una celda del tablero, donde tomarían el valor 0 si no hay reina en la celda, y 1 en caso contrario. A continuación se muestra un posible individuo.



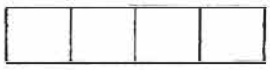
= 0, 1, 0, 0
= 0, 0, 0, 1
= 1, 0, 0, 0
= 0, 0, 0, 0

Individuo = "0100" + "0001" + "1000" + "0000" = "0100000110000000"

De igual forma, se pueden plantear representaciones alternativas. Por ejemplo, una representación alternativa consideraría que cada gen representa a una fila del tablero, así el valor alélico del gen sería {0,1,2,3,4}. Donde 1,2,3 y 4 indican la presencia de una reina en la celda 1,2,3 o 4 respectivamente y 0 indica que ninguna reina se encuentra presente en dicha fila.



= 2
= 4
= 1

 = 0

Individuo = "2" + "4" + "1" + "0" = "2410"

Para efectos didácticos, se tomará el modelo de 16 genes binarios.

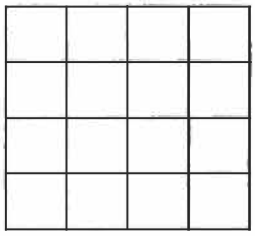
1.1.2 Función Fitness

En todo proceso evolutivo, los individuos de una población están sujetos a una medida de calidad llamada *Fitness*, el cual esta dado por la *Función Fitness*. La función fitness es el segundo elemento más importante de todo algoritmo genético, asegurando la correcta convergencia de la búsqueda a la solución. El valor fitness permite identificar qué individuo son superiores, y así transferir su información genética a la siguiente generación.

Para el problema de las n reinas, se desea maximizar el número de reinas en el tablero y minimizar el número de jaques entre ellas. Se podría considerar que una solución es buena, si tiene más reinas en el tablero, incremento en un $+\Delta f$ su valor fitness; pero las reinas ubicadas en el tablero no deben hacerse "jaque" entre ellas, por lo que se puede penalizar con una $-\Delta r$ por cada jaque en el tablero. De esta forma, la función fitness puede ser representada por la siguiente formula:

Función Fitness = Δf * #reinas - Δr * #reinas en jaque

Analizando casos extremos:



= "0000000000000000"

Valor Fitness = $\Delta f * 0 - \Delta r * 0$

R	R	R	R
R	R	R	R
R	R	R	R
R	R	R	R

= "1111111111111111"

Valor Fitness = $\Delta f * 16 - \Delta r * 16$

En ambos casos, si Δf y Δr son iguales, se obtendrá un valor fitness de cero. Muchas veces se desea incrementar el efecto de las restricciones, aumentando el valor Δr, de manera que sí $\Delta f < \Delta r$ se obtendrá un valor negativo. Esto tiene ciertas ventajas, por ejemplo:

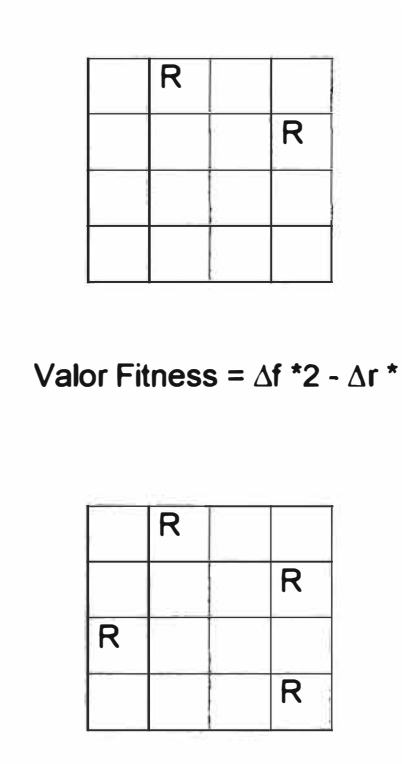
	R		
			R
R			
	R		

= "0100000110000100"

Valor Fitness = $\Delta f * 4 - \Delta r * 4$

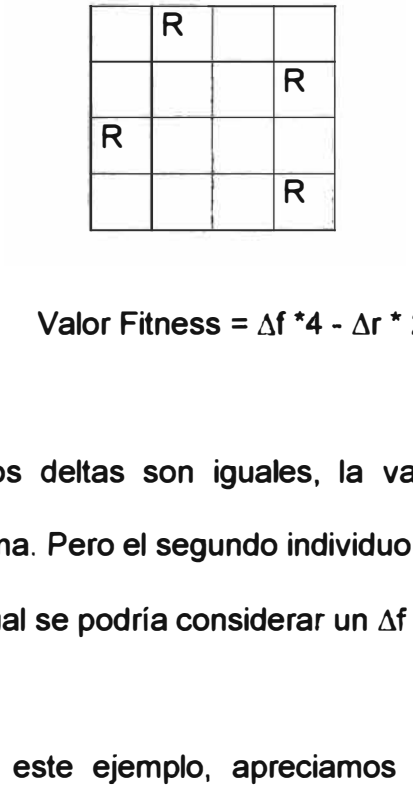
Si los deltas son iguales este individuo tendrá un valor fitness de 0, sin embargo esta solución esta más cerca de la solución óptima, a comparación del código "1111111111111111".

Para los casos:



= "0100000100000000"

Valor Fitness = $\Delta f * 2 - \Delta r * 0$



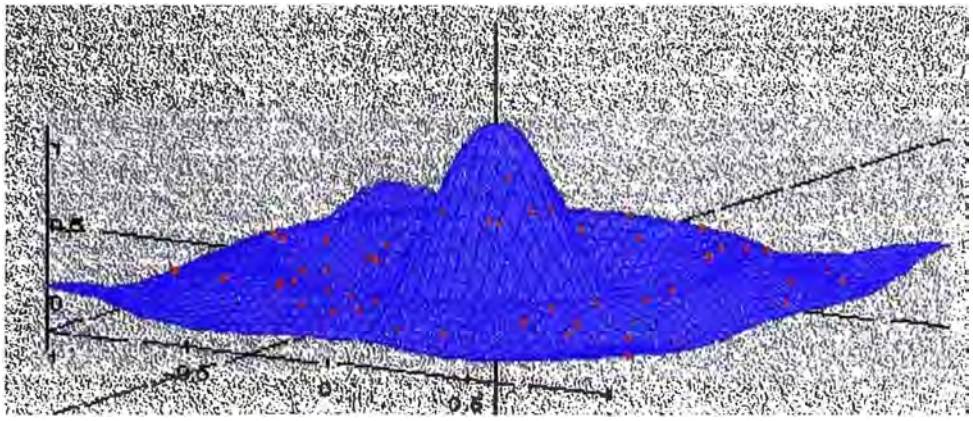
= "0100000110000001"

Valor Fitness = $\Delta f * 4 - \Delta r * 2$

Si los deltas son iguales, la valoración de ambos individuos sería la misma. Pero el segundo individuo esta mas cerca de la solución, razón por la cual se podría considerar un $\Delta f > \Delta r$.

Con este ejemplo, apreciamos el grado de importancia que juega la función fitness, en el proceso de búsqueda.

La función fitness puede verse como una superficie multidimensional. En el caso de un problema de minimización, los pozos representan las soluciones de calidad superior, mientras las crestas representan las soluciones de poca calidad. En el caso de un problema del maximización, el punto superior en la topografía es la mejor solución. Una buena función fitness nos asegura la convergencia, mientras que lo contrario nos deja en una búsqueda prácticamente aleatoria.



1.1.3 Operadores Genéticos

En todo proceso evolutivo, el proceso de búsqueda es realizado por los *operadores genéticos*, los que son aplicados sobre los individuos. Aunque existen muchas variaciones y adaptaciones los más comunes son la reproducción, la recombinación (crossover) y la mutación. Estos operadores reciben como entrada algunos individuos seleccionados seudo-aleatóriamente y devuelven como resultados nuevos individuos que pueden ser similares o incluso iguales a los individuos originales.

1.1.3.1 Reproducción

Este operador produce una nueva población, Ar(t), extrayendo algunos individuos de la población anterior, A(t), sin que éstos experimenten variación alguna en su código genético. Así, la nueva población viene a ser un subconjunto de la población original. La extracción puede llevarse a cabo de varias maneras:

- ***Selección de rueda de ruleta***, este método es el más usado, y consiste en calcular el valor fitness para todos los individuos de la población vieja A(t), ordenarlos de mayor a menor a los individuos de acuerdo a su fitness, calcular una probabilidad para todos los individuos, el cual es proporcional al valor fitness pr(Ai(t)). Por ejemplo:

$$pr(Ai(t)) = \frac{fitness(Ai(t))}{\sum_{j=1}^{j=n} fitness(Aj(t))}$$

para luego disparar un número aleatorio con valor entre 0 y 1, Rnd(1), y buscar el valor j tal que cumpla:

$$Rnd(1) \leq \sum_{i=1}^{i=j} pr(Ai(t))$$

Así el individuo $A_j(t)$ será seleccionado para formar parte de la nueva población Ar(t). De esta manera se logra que los individuos con mayor

fitness tengan más probabilidad de ser elegidos. Aunque el método de la ruleta es el más difundido, requiere de gran capacidad de procesamiento para calcular la probabilidad de todos los individuos de la población, ordenarlos y realizar la sumatoria de probabilidades.

- **Selección por duelo**, este método se basa en el principio de competencia. En la naturaleza es muy poco probable que todos los miembros de una población compitan entre sí para reproducirse. Lo habitual es que solo se realice la competencia en un grupo reducido o incluso solo dos individuos. De forma similar, se selecciona el grupo de individuos de forma completamente aleatoria, y se compara los valores fitness de los individuos de este grupo. El individuo de mayor fitness es el seleccionado para reproducirse.

Cabe recalcar que es recomendable evitar incluir el mismo individuo mas de una vez, con la finalidad de mantener la diversidad genética.

El operador de reproducción desempeña una labor de "memoria" de la población, asegurando que los mejores individuos tengan una mayor probabilidad de sobrevivir inalterados en la siguiente población A(t+1). Dicho de otra forma la reproducción aumenta nuestra probabilidad de no perder las mejores soluciones.

1.1.3.2 Crossover.

Este operador también genera una nueva población Ac(t) a partir de una población original A(t). Existe una diversidad de variaciones de este operador, los que dependen del tipo de problema a modelar, pero se explicará el crossover normal. En el crossover normal se escogen pares de individuos al azar, ya sea por ruleta o por duelo, de tal manera que los de mayor fitness tienen más probabilidad de ser elegidos. Cada par es entonces recombinado, escogiendo uno o más puntos, de acuerdo a una probabilidad uniformemente distribuida sobre la longitud de las cadenas genéticas de los padres, y cortándolos en dos o más partes. Los nuevos individuos se forman por la yuxtaposición de los segmentos impares del primer padre y los segmentos pares del segundo padre. El siguiente ejemplo muestra gráficamente este operador.

1011011000100101	Padre
0010110110110111	Padre

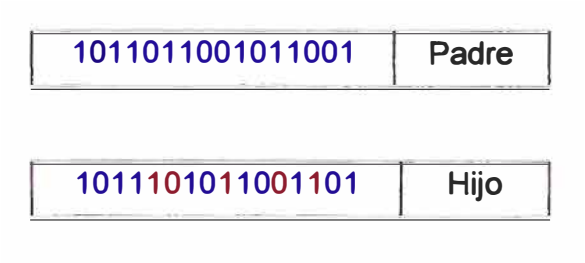
1011010110110101	Hijo
0010111000100111	Hijo

Sin duda alguna, el crossover es el operador genético de mayor importancia y garantiza la convergencia del algoritmo a una mejor solución, permitiendo que el mejor código genético se transmita de generación en generación. Esta afirmación es demostrada

matemáticamente en el teorema del Schemadata que se encuentra el apéndice.

1.1.3.3 Mutación

Este operador modifica cada alelo (un bit en la cadena de bits) de algunos individuos de la población, escogidos por probabilidad. El nuevo valor del alelo es aleatorio, con una distribución uniforme de probabilidad. Por ejemplo:



1.1.4 Proceso Evolutivo

Luego de revisar los conceptos de los algoritmos genéticos, se pasará a la descripción del algoritmo evolutivo en sí, que es el que encuentra la solución al problema modelado. El proceso evolutivo de un algoritmo genético tradicional puede describirse de la siguiente forma:

Paso 1	Generar al azar una población inicial conformada por un número determinado de individuos. Generalmente esta población es creada por medios estocásticos, a fin de asegurar una buena diversidad en los individuos. $A(0) := (A_1(0).. A_n(0))$
Paso 2	Calcular el valor de la función del fitness F(Ai(t)) para cada individuo de la población A(t).
Paso 3	Evaluar si se encontró la solución, en caso afirmativo la búsqueda termina.
Paso 3	Generar una población intermedia Ar(t) aplicando el operador de reproducción.
Paso 4	Generar A(t+1) aplicando algunos otros operadores a la población A(t)
Paso 5	Hacer t := t + 1 e ir a Paso 2.

1.2 ESTRATEGIAS EVOLUTIVAS

Una aproximación para simular la evolución natural aplicado para resolver problemas de optimización, dio como resultado el desarrollo de las Estrategias Evolutivas (EE) en 1964 por Rechenberg, en la Universidad Técnica de Berlín (Alemania).

En el sentido Neo-Darwiniano, en la evolución natural intervienen cuatro procesos estadísticos principales, que actúan sobre las poblaciones y especies para llegar a la solución óptima. Estos procesos son la reproducción, la mutación, la competición y la selección. La evolución es

entonces el resultado de estos procesos estocásticos fundamentales cuando ellos actúan en poblaciones, generación tras generación. La versión más simple de Estrategias Evolutivas es (1+1) EE.

1.2.1 Población

La población en Estrategias Evolutivas esta definida por un conjunto de individuos compuestos a su vez de dos cadenas de números reales de longitud fija. A estas dos cadenas se les denominan cadena de parámetros y cadena del objeto.

La cadena de objetos, contiene la configuración de los valores que se desea optimizar

$$x = (x_1, x_2, \ldots, x_n)$$

La cadena de parámetros de la estrategia, controla la mutación de los parámetros del objeto

$$\sigma_{padre} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$$

1.2.2 Función objetivo

Al igual que todos los algoritmos de la computación evolutiva, en Estrategias Evolutivas se hace uso de una función objetivo, que determina si una solución es mejor que otra.

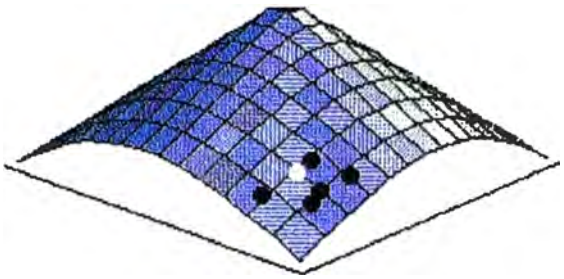
La principal diferencia con los algoritmos genéticos, es que sólo los mejores se reproducen (selección elitista).

1.2.3 Estrategia Evolutiva (1+1)

En este modelo, existen dos individuos y se utilizan los procesos estocásticos de mutación y selección.

La configuración inicial del padre $x_{padre,ini}$ se selecciona al azar de la región factible del espacio de parámetros multi-dimensional. Cada componente de esta cadena corresponde a uno de los parámetros de optimización, llamado objeto de parámetros. Una función del valor objetivo f(x) es asociada con esta configuración. Mutar esta configuración inicial significa, que el vector v, cuyos elementos tienen una distribución Gauseana con valor medio de cero y una desviación estándar σ llamado stepsize, se agrega a la configuración del padre.

$$x_{hijo} = x_{padre} + v\ (0, \sigma_{padre})$$



Es bien conocido, que aproximadamente 68% de todos los valores seleccionado en forma aleatoria se encontrarán dentro del intervalo [-σ, +σ], y un 95.5% dentro del intervalo [-2σ, +2σ]. La distribución de Gauss del componente del vector v corresponde a la observación de la naturaleza, *cambios pequeños suceden mas frecuentemente que cambios grandes*. Se compara la calidad del x_{hijo} de la configuración descendiente con el x_{padre} de la configuración paternal, la configuración que tenga el mejor valor de la función objetivo f(x) es designado a ser padre para la próxima generación. Este proceso se repite hasta que algún criterio de finalización sea activado. Sin embargo, después de un cierto número de generaciones la desviación estándar σ del vector mutación v se adapta al progreso de optimización.

Sea el ratio de las mutaciones positivas

p(pos) = Número de mutaciones positivas / Número de mutaciones totales

Si éste ratio es menor que la probabilidad predefinida (normalmente 1/5), la desviación estándar σ es disminuida multiplicándolo con el factor 0.85; caso contrario la desviación estándar σ es aumentada dividiéndolo entre el

mismo factor 0.85. Esta adaptación dinámica de la desviación estándar σ es una de las características cruciales de muchas estrategias estocásticas.

1.2.4 Estrategia Evolutiva (μ, λ)

Este es una generalización del modelo anterior, logrando una mejor adecuación del proceso de evolución natural, donde en cada generación, μ padres generan λ hijos haciendo uso de las operaciones genéticas de recombinación y mutación.

1.2.5 Algoritmo Básico

Paso 1	Generar una población inicial, donde los elementos de la cadena objetivo son inicializados en forma aleatoria, mientras que los elementos de la cadena de parámetros son inicializados con un valor constante.
Paso 2	Seleccionar los μ mejores individuos basados en algún algoritmo de la selección (función objetivo).
Paso 3	Usar los μ individuos para generar λ hijos.
Paso 4	Ir al Paso 2 hasta que se logre el resultado deseado.

1.2.6 Generación de Descendientes

El número de padres en cada generación esta definida por μ, si este factor es muy grande, muchos rasgos malos pueden persistir, pero si es pequeño, la solución puede estancarse.

El número de hijos producidos esta representado por λ, el cual esta limitado por los recursos disponibles de computación, mientras más grande el número más rápida será la evolución.

Los operadores genéticos usados son muy similares a los usados en algoritmos genéticos, siendo la mutación el operador más importante.

1.2.6.1 Mutación

Es el encargado de la diversidad genética y permite insertar nuevos genes a la población que pueden ser necesarios en la solución óptima. Este proceso es estocástico y cambia los parámetros del vector objetivo y desviación estándar como fue explicado en la el modelo (1-1)

$$x_{hijo} = x_{padre} + v\,(0, \sigma_{padre})$$

Por ejemplo:

Padre	X	8	12	31	...	5
	σ	0.1	0.3	0.2	...	0.5

Hijo	X	8.2	11.9	31.3	...	5.7

1.2.6.2 Recombinación

Proceso similar al usado en algoritmos genéticos, donde la probabilidad de combinación es la misma para los dos padres.

Por ejemplo:

Padre 1	8	12	31	...	5
Padre 2	2	5	23	...	14

Hijo	2	12	31	...	14

En muchos casos, se usa la recombinación media, debido a que muchos padres tienen algunos parámetros malos, los que pueden heredados a los hijos. Por esta razón, los elementos de los hijos serán los promedios de los padres.

Padre 1	8	12	31	...	5
Padre 2	2	5	23	...	14

Hijo	5	8.5	27	...	9.5

1.2.7 Proceso de Evolución

Existen cuatro tipos de evolución en las estrategias evolutivas, las que definen el tipo de operador genético a usar:

- μ, λ
- $\mu/\rho, \lambda$
- $\mu+\lambda$
- $\mu/\rho+\lambda$

1.2.7.1 Estrategias Evolutivas (μ, λ)

Los μ padres producen λ hijos, usando únicamente la mutación (ninguna recombinación). Los mejores μ hijos serán los padres de la próxima generación. Los padres no formarán parte de la siguiente generación, para lo cual se debe tomar en cuenta que el número de hijos es mayor o igual al número de padres($\lambda \geq \mu$). El proceso ($\mu/\rho, \lambda$) es igual que el anterior, con la diferencia que hace uso de la recombinación.

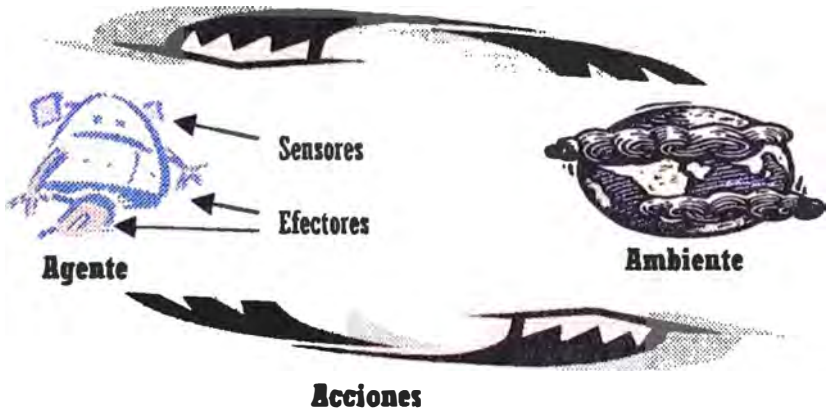
1.2.7.2 Estrategias Evolutivas ($\mu+\lambda$)

Los μ padres producen λ hijos usando únicamente la mutación. Los mejores μ individuos (padres e hijos) serán los padres de la próxima generación, es decir, los padres se mantienen en la siguiente generación.

La población total en cada generación es igual a $\lambda+\mu$. El proceso $(\mu/\rho+\lambda)$ es similar al proceso $(\mu+\lambda)$, considerando el operador de recombinación.

1.3 AGENTES INTELIGENTES

Un agente es un ente que cumple una función, el cual percibe su ambiente a través de los sensores y actúa en éste a través de los efectores. Por ejemplo, un agente humano tiene ojos, orejas y otros órganos como sensores, y las manos, piernas, boca y otras partes del cuerpo como efectores. Un agente robótico consta de cámaras y buscadores de rangos de infrarrojo como sensores y los varios motores por efectores. Un agente del software tiene codificado una cadena de bits como sus perceptores y ejecutores.



1.3.1 Como los agentes deben actuar

Un agente racional es uno que hace las cosas correctas, donde la acción correcta es uno que causará al agente ser muy exitoso. El problema es decidir cómo y cuándo evaluar el éxito del agente.

Se usa el término *medida de actuación,* también llamado función de valoración, para evaluar cuan exitoso es un agente, esta medida debe ser objetiva e impuesta por una autoridad. En otras palabras, los diseñadores como observadores externos establecerán un estándar de que significa ser exitoso en un ambiente y usarlo para medir la actuación de los agentes.

Como un ejemplo, se puede considerar el caso de un agente que se supone debe barrer un piso sucio. Una medida de actuación plausible podría ser la cantidad de suciedad limpiada en ocho horas. Una medida de la actuación más sofisticada podría considerar la cantidad de electricidad consumida y la cantidad de ruido generado, entre muchas otras alternativas.

El "cuando evaluar" la actuación también es importante. Si se midiera cuánta suciedad el agente ha limpiado en la primera hora del día, se podría premiar a los agentes que empiezan rápido (aun cuando ellos hacen pequeño o ningún trabajo después), y castigando a esos que trabajan en forma consistente. Para una mejor medida podría ser en un turno de ocho horas o en todo el tiempo de vida.

Se debe tener cuidado para distinguir entre la racionalidad y omnisciencia. Un agente omnisciente sabe el resultado real de sus acciones, y puede actuar de acuerdo a ello; pero la omnisciencia es imposible en realidad.

Es imposible culpar a un agente de no tener en cuenta que algo extraño e impredecible puede pasar, o por tomar una acción que nunca debería haber tomado. El punto es que si se especifica que un agente inteligente siempre debe hacer lo que realmente es lo correcto, será imposible diseñar un agente que cumpla con todas las especificaciones. En resumen, lo que es racional en cualquier momento dado depende de cuatro cosas:

- La medida de la actuación que define el grado de éxito.
- Todo lo que el agente ha percibido hasta ahora. Llamaremos a eso una historia completa de la sucesión de percepciones.
- Lo que el agente sabe sobre el ambiente.
- Las acciones que el agente puede realizar.

Esto lleva a una definición de un **agente racional ideal**: Para cada posible sucesión de percepciones, un agente racional ideal debe realizar la acción esperada que maximice su medida de actuación, sobre la base de la evidencia proporcionada por la sucesión de percepciones y cualquier conocimiento preprogramado que tiene el agente. Realizar acciones para obtener información útil, es una parte importante de la racionalidad.

1.3.2 Mapeo ideal de las secuencias de percepción a acciones

Una vez comprendido que la conducta de un agente depende de su sucesión de percepciones, entonces se puede describir un agente

particular cualquiera, creando una tabla de acciones que tomaría en respuesta a cada posible secuencia de percepción; para la mayoría de los agentes, esta será una lista grande de hecho, a menos que se ponga un límite en la longitud de sucesiones de percepción que se desea considerar; Este tipo de lista es llamado **mapeo** de percepciones a acciones. Si los mapeos describen a los agentes, entonces los **mapeos ideales** describen a los agentes ideales. *Especificar que acciones ha de realizar un agente, en respuesta a cualquier sucesión de percepciones, a fin de obtener el diseño para un agente ideal.*

Esto no significa por supuesto, que se tiene que crear una tabla explícita con una entrada para todas las posibles sucesiones de percepción. Es posible definir una especificación del mapeo sin enumerarlo exhaustivamente.

1.3.3 Autonomía

Hay un elemento importante en la definición de un agente ideal: "creación de conocimiento". Si las acciones del agente son basados completamente sobre el conocimiento construido, este no necesita poner atención a sus percepciones, entonces se dice que al agente le falta autonomía. La conducta de un agente puede ser basada en su propia experiencia y en el conocimiento creado, usado en la construcción del agente para un ambiente particular en el cual opera.

Un sistema es autónomo, cuando su conducta es determinada por su propia experiencia. Sin embargo, es demasiado severo requerir autonomía completa en todo el sentido de la palabra: cuando el agente ha tenido pequeña o ninguna experiencia, tendrá que actuar aleatóriamente a menos que el diseñador le brinde alguna ayuda.

Así como la evolución proporciona a los animales diferentes tipos de reflejos, para que puedan sobrevivir y aprendan de ellos mismos, sería razonable proporcionar a un agente de inteligente artificial, un poco de conocimiento inicial así como la habilidad de aprender.

Un agente que opera basándose en asunciones construidas, sólo operará exitosamente cuando esas asunciones son dadas y así pierde flexibilidad. Un agente inteligente verdaderamente autónomo, debe operar con éxito en una gran variedad de ambientes, dándole el tiempo suficiente para adaptarse.

1.3.4 Estructura de los agentes inteligentes

El trabajo de IA es el de diseñar el programa del agente: una función que lleva a cabo el mapeo de las percepciones a acciones. Se asume que este programa correrá en alguna clase de dispositivo computacional al cual se le llama **arquitectura**. La arquitectura podría ser una computadora común o podría incluir hardware de propósito específico para ciertas tareas, como el procesamiento de imágenes o el filtro de la entrada de audio. En

general, la arquitectura proporciona las percepciones de los perceptores al programa, ejecuta el programa, y realiza las diferentes acciones del programa, cuando ellos se generan. La relación entre agente, arquitectura y programa pueden resumirse como sigue:

agente = arquitectura + programa

Antes de diseñar un programa agente, se debe tener una idea bastante buena de las posibles percepciones y acciones, qué metas o medida de la actuación se supone que el agente deba lograr, y en qué clase de ambiente operará. Por ejemplo, a continuación se muestra algunos elementos básicos para la selección de los tipos de agentes.

Tipo de Agente	Percepción	Acciones	Metas	Ambiente
Sistema de diagnóstico médico	Síntomas, hallazgos, respuesta del paciente	Preguntas, pruebas, tratamientos	Paciente saludable, minimizar costos	Paciente, hospital
Sistema de análisis de imágenes satelitales	Píxeles de diferente intensidad y color	Imprimir una categorización de la escena	Categorización correcta	Imágenes de satélites en orbita
Robot para tomar piezas	Píxeles de diferente intensidad	Recoge partes y los ordena en las cajas	Poner las partes en las cajas correctas	Faja transportadora de partes
Controlador de refinería	Lectura de presión y temperatura	Abrir, cerrar válvulas, ajustar temperatura	Maximizar la pureza, rendimiento y seguridad	Refinería

Tutor interactivo de ingles	Palabras digitadas	Imprimir ejercicios, sugerencias y correcciones	Maximizar el puntaje de los estudiantes	Grupo de estudiantes

1.3.4.1 Programa del agente

Para construir agentes inteligentes se usara el mismo esqueleto, aceptando percepciones de un ambiente y generando acciones. Las primeras versiones del programa del agente tendrían una forma muy simple.

Función CUERPO-AGENTE(percepción)

Estático *memoria //La memoria del agente*

Memoria = ACTUALIZAR-MEMORIA(*memoria, percepción*)

Acción = *E*SCOGER-MEJOR-ACCIÓN(*memoria*)

Memoria = ACTUALIZAR-MEMORIA(*memoria, acción*)

Retornar *acción*

Cada uno usará algunos datos de la estructura interna que serán actualizados como nuevas percepciones. Estas estructuras de datos son operadas por el proceso de toma de decisiones del agente, para la selección de la acción, el cual será pasado a la arquitectura para su ejecución.

Hay dos cosas por notar sobre este bosquejo del programa.

- Aunque se definió el mapeo como una función de las sucesiones de percepción a acciones, el programa del agente solo recibe una única percepción como entrada. Luego, depende del agente construir la sucesión de percepciones en memoria si es necesario. En algunos ambientes, es posible tener bastante éxito sin guardar la sucesión de percepciones, y en dominios complejos, es inaplicable guardar la sucesión completa.

- La meta o la medida de la actuación no es parte del bosquejo del programa, porque la medida de la actuación se aplica para juzgar la conducta del agente externamente, a menudo es posible lograr buenas actuaciones sin un conocimiento explícito de la medida de la actuación.

1.3.4.2 Tipo de Agentes

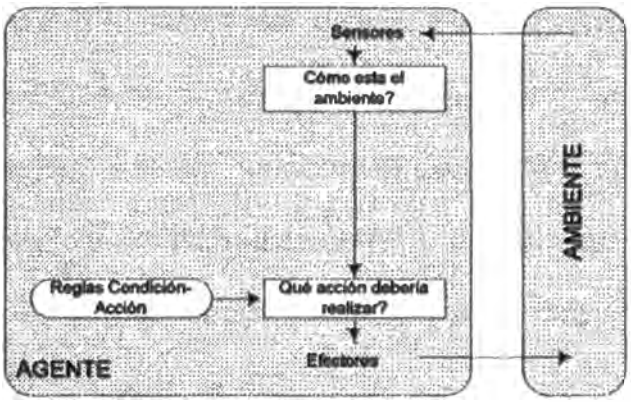
Existe una diversidad de agentes:

- Agentes de reflejo simple
- Agentes que guardan huella del mundo
- Agentes basados en metas
- Agentes basados en utilidad

1.3.4.2.1 Agentes de reflejo simple

Es inaplicable el crear completamente la tabla de comportamiento, sin embargo, se puede resumir porciones de la tabla notando ciertas ocurrencias comunes de las asociaciones de entrada / salida. Estas simplificaciones pueden darse gracias a un conjunto de entradas o condiciones, los cuales activan algunas conexiones en el programa del agente para luego tomar la acción más adecuada. Esta conexión es llamada regla condición-acción representada por:

si condición **entonces** acción

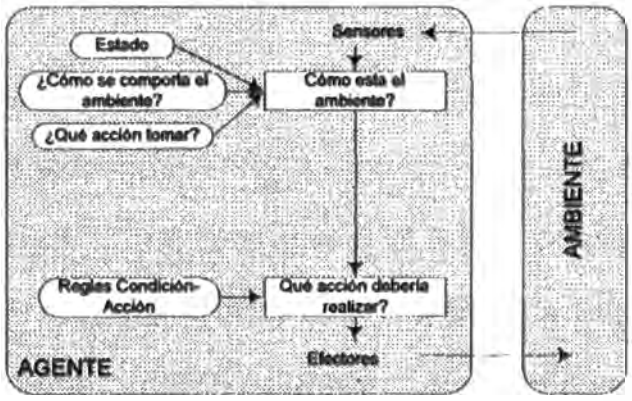


La figura anterior muestra en forma esquemática, una estructura del agente de reflejo simple, mostrando cómo las reglas de condición-acción le permiten al agente hacer la conexión de la percepción a la acción.

1.3.4.2.2 Agentes que guardan huella del mundo

El agente de reflejo simple descrito antes, sólo trabajará si la decisión correcta puede tomarse basándose en la percepción actual, muchas veces los sensores no proporcionan acceso completo del estado del mundo y los sistemas de reflejo simple pueden fallar. Para resolver este problema, el agente puede necesitar mantener alguna información del estado interno para distinguir entre estados del mundo que generan la misma entrada sensorial, pero no obstante es significativamente diferente. "Significativamente diferente" es referida a que diferentes acciones son apropiadas en estados similares.

Actualizando esta información del estado interno, conforme el tiempo avanza, requiere dos tipos de conocimiento a ser codificados en el programa del agente. Primero, se necesita cierta información sobre cómo el mundo evoluciona independientemente del agente. Segundo, se necesita un poco de información sobre cómo las acciones del agente afecten al mundo.



La figura anterior muestra una estructura del agente, mostrando cómo la percepción actual combinada con el estado interno anterior, generan una descripción actualizada del estado actual, permitiendo una mejor toma de decisiones por parte de los agentes.

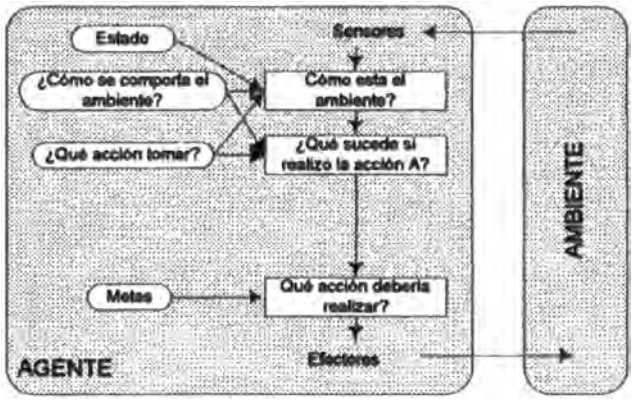
1.3.4.2.3 Agentes basados en metas

Conocer el estado actual del ambiente no siempre es suficiente para decidir que hacer. En otras palabras, así como una descripción del estado actual, el agente necesita un tipo de información sobre la meta, el cual describe situaciones que son deseables. El agente puede combinar esta con información sobre posibles resultados de las acciones, para escoger acciones que logren cumplir con la meta. A veces esto será simple, cuando la satisfacción del objetivo es el resultado inmediato de una simple acción; a veces será más difícil, cuando el agente tiene que considerar largas secuencias de acciones para lograr la meta. Búsquedas y planificación, son herramientas de la inteligencia artificial consagradas a encontrar sucesiones de acciones que permiten lograr los objetivos del agente.

La toma de decisiones de este agente, es fundamentalmente diferente a las reglas condición-acción descritas anteriormente, éste involucra considerar el futuro. "¿Qué sucedería si se hace tal-o-cual?", "¿Me hará sentir feliz?". En los planes del agente reflejo, esta información no es explícitamente usada, porque el diseñador tiene preprogramado la acción correcta para varios casos.

Para los agentes reflejos, si se desea cambiar el comportamiento de estos, es necesario rediseñar todas las reglas condición-acción que describirían el nuevo comportamiento. Por el contrario, el agente basado en metas es más flexible con respecto a cambios. Simplemente especificando la nueva meta, se puede conseguir que el agente basado en metas tenga una nueva conducta.

A continuación se muestra la estructura del agente basado en metas.

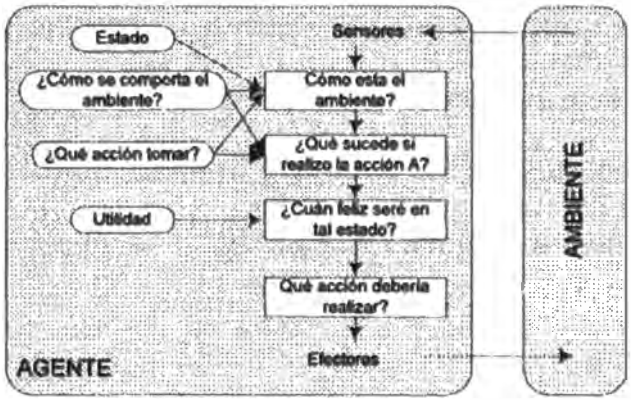


1.3.4.2.4 Agentes basados en utilidad

Las metas por si solas no son suficientes para generar una conducta de alta calidad, éstas apenas proporcionan una distinción cruda entre los estados "feliz" e "infeliz", considerando que una medida de actuación más general debe permitir una comparación de diferentes estados del ambiente (o sucesiones de estados) de acuerdo a cuan feliz harían al agente si pueden ser satisfechos. Porque "feliz" no parece muy científico, es mejor decir que si un estado del ambiente es preferido a otro, entonces tiene una mas alta utilidad para el agente.

La utilidad es por consiguiente, una función que mapea un estado hacia un número real, el que describe el grado asociado de felicidad. Una especificación completa de la función de utilidad permite decisiones racionales en los casos donde las metas son conflictivas.

La estructura del agente basado en utilidad es mostrada en la siguiente figura:



1.3.4.3 Ambientes

Los agentes proporcionar su estado a los agentes y pueden cambiar con en respuesta a las acciones realizas sobre ellos, por parte de los agentes. Se describirán los diferentes tipos de ambientes y cómo ellos afectan el diseño de los agentes.

1.3.4.3.1 Propiedades de los ambientes

Los ambientes vienen en diferentes tipos. Las distinciones principales a ser hechas son las siguientes:

- **Accesibilidad vs. Inaccesibilidad**

  Si el aparato sensorial de un agente tiene acceso completo al estado del ambiente, entonces se dirá que el ambiente es accesible para el agente. Un ambiente es efectivamente accesible, si los sensores descubren todos los aspectos que son relevantes para la selección de una acción. Un ambiente accesible es conveniente porque el agente no necesita mantener ningún estado interno que mantiene huella del mundo.

- **Deterministico vs. Nodeterminístico**

  Si el siguiente estado del ambiente esta completamente determinado por el estado actual y la acción seleccionada por el agente, entonces se podrá decir que el ambiente es determinístico. En principio, el agente no necesita preocuparse por la incertidumbre en un accesible ambiente determinístico. Si el ambiente es inaccesible, entonces puede parecer ser nodeterminístico; esto es particularmente verdadero si el ambiente es complejo, haciendo difícil guardar rastro de todos los aspectos inaccesibles.

- **Episódico vs. Noepisódico**

  En un ambiente episódico, la experiencia del agente es dividida en **episodios**. Cada episodio consiste en la percepción y acción del agente. La calidad de su acción simplemente depende del episodio en sí, porque los subsecuentes episodios no dependen de qué acciones ocurrieron en episodios anteriores.

- **Estática vs. Dinámica**

  Si el ambiente puede cambiar mientras un agente está actuando, entonces decimos que el ambiente es dinámico para ese agente; en caso contrario es estático. Los ambientes estáticos son fáciles de manejar, porque el agente no necesita seguir viendo al mundo mientras está decidiendo una acción, ni necesita preocuparse por el tiempo. Si el ambiente no cambia en el tiempo pero sí la evaluación de la actuación, entonces decimos que el ambiente es semi dinámico.

- **Discreto vs. Continuo**

  Si hay un número limitado de distintas y claramente definidas percepciones y acciones entonces se dice que el ambiente es discreto. El ajedrez es discreto, porque tiene un número determinado de posibles movimientos en cada turno.

Hay que aclarar que, diferentes tipos de ambientes requieren diferentes programa de agentes para manejarlos eficazmente. Como se puede esperar, el caso más difícil es inaccesible, noepisódico, dinámico y continuo.

1.3.5 Objetivo de los Sistemas Multi-Agentes

Intentan estudiar e implementar la coordinación necesaria para realizar trabajos modulares, los cuales tienen objetivos individuales y deben negociar sus prioridades para llegar a un objetivo global. Los sistemas multiagentes trabajan concentrándose en la explotación del poder del paralelismo y las ventajas de la modularidad, aplicando una arquitectura distribuida entre sistemas para resolver problemas de IA.

1.4 SISTEMAS CLASIFICADORES DE APRENDIZAJE.

Una de las criticas que se oyen más a menudo respecto a la Inteligencia Artificial (IA), es que las máquinas no pueden considerarse como "inteligentes" hasta que no sean capaces de aprender a hacer cosas nuevas y adaptarse a las nuevas situaciones, en lugar de limitarse a hacer aquellas actividades para las que fueron programadas. Por ello, los investigadores de IA están destinando muchos esfuerzos en la investigación de *sistemas de aprendizaje* capaces de dotar a las máquinas de "inteligencia".

Un sistema de aprendizaje es un programa que es capaz por un lado, de tomar decisiones basándose en la experiencia acumulada en la resolución de problemas, y por otro, de perfeccionar sus habilidades. Un importante dilema a resolver en los sistemas de aprendizaje es como determinar que

cambios son precisos y como hacerlos para perfeccionar sus habilidades. Un sistema de aprendizaje esta conformado por dos componentes:

I. *Un subsistema de tarea* o *componente de rendimiento,* cuya conducta se modifica a lo largo del tiempo vía un proceso de aprendizaje y,
II. *Un subsistema de aprendizaje*, responsable de observar al subsistema de tareas a lo largo del tiempo y efectuar los cambios oportunos en su conducta.

El dilema es el cómo determinar el espacio de cambios estructurales posibles y las operaciones legales que permiten hacerlos, de forma que la conducta del sistema mejore. Resolver este dilema no es nada fácil, y sobre todo cuando afecta a tareas que requieren gran capacidad de adaptación robusta, como por ejemplo las desarrolladas en entornos que varían en el tiempo, pobremente definidos y con continuos cambios. Algunas aproximaciones al dilema que están obteniendo gran éxito, se han basado en modelos de organismos naturales; como por ejemplo, sucede en el caso de las *redes neuronales, la computación evolutiva, etc.*

Los algoritmos genéticos han mostrado ser una herramienta de búsqueda muy poderosa, y esto permite pensar a los AGs como una técnica apropiada para el diseño de sistemas de aprendizaje, en los cuales las acciones de control requeridas para controlar el subsistema de tarea varían rápidamente de forma que no pueden ser preestablecidas.

De acuerdo con De Jong, existe una gran variedad de aproximaciones a los sistemas de aprendizaje basadas en AGs. Las tres más importantes son las siguientes:

- Restringir los cambios estructurales a la *modificación de parámetros* que controlan la conducta del subsistema de tarea, y usar los AGs para desarrollar una estrategia que rápidamente localice las combinaciones útiles de los valores de los parámetros.
- Usar AGs para cambiar las estructuras de datos complejas, como las agendas, que controlan la conducta del subsistema de tarea.
- Usar AGs para cambiar el subsistema de tarea, de modo que este evolucione por sí mismo.

Normalmente un subsistema de tarea se considera como un *sistema de producción*, formado por un conjunto no ordenado de reglas, que representa la población de individuos que será explotado y explorado por el subsistema de aprendizaje basado en AGs. Históricamente hay dos modos de usar los sistemas de producción para representar al conjunto de la población:

- *La Aproximación Pittsburg*, donde cada miembro de la población representa un conjunto de reglas de producción, y por tanto, una población es un conjunto de conjuntos de reglas. Cada conjunto de reglas tiene asignado un fitness, calculado sobre la base de una medida de rendimiento, el cual es usado por el AG para navegar en el espacio de posibles conjuntos de reglas.

- *La Aproximación Michigan*, donde cada miembro de la población representa una regla de producción individual, y por tanto, una población es un conjunto de reglas. Cada regla tiene asociado un fitness, calculado sobre la base de un algoritmo de asignación de crédito (por ejemplo el algoritmo del *bucket-brigade*), el cual se usa cuando se aplica el AG y cuando surgen conflictos entre las reglas.

En la aproximación Pitt el AG evalúa implícitamente reglas individuales, mientras en la aproximación Michigan lo hace explícitamente de acuerdo a un algoritmo. La aproximación Pitt conlleva un mayor gasto computacional, tanto en tiempo de proceso como en memoria, que la aproximación Michigan. Ahora, ¿cual de las dos aproximaciones es mejor en el sentido de ser más efectiva al evolucionar el subsistema de tareas?. No esta clara la respuesta, depende del tipo de aplicación. Se acepta que la aproximación Pitt es mas útil en entornos de trabajo off-line en los cuales cambios radicales de conducta son permisibles, mientras la Michigan es mas útil en entornos de trabajo on-line en tiempo real en los cuales cambios radicales de conducta no pueden ser tolerados.

Dado que estamos interesados en tratar problemas en tiempo real en los cuales las condiciones de trabajo varían muy rápidamente, nuestro estudio se centrara en los sistemas de aprendizaje basados en AGs inspirados en la aproximación Michigan, y más concretamente en una clase de sistemas de aprendizaje de propósito general conocidos con el nombre de *Sistemas Clasificadores (SCs), cuyos* fundamentos fueron presentados por Holland.

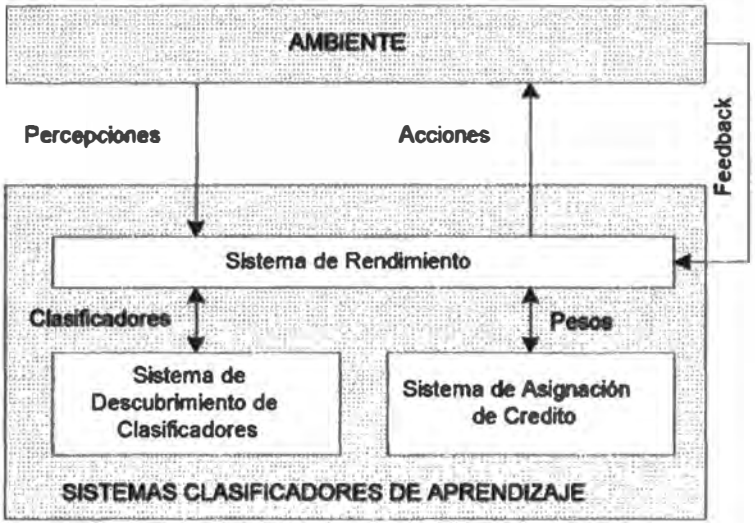
Ahora bien, existen versiones de SCs que usan la aproximación Pitt, pero los mas estudiados son los basados en la aproximación Michigan.

1.4.1 Descripción de un Sistema Clasificador

Las ideas básicas de los SCs fue presentada por Holland y el primer SC se presento por Holland y Reitman, desde entonces diferentes tipos de SCs se han propuesto. Los SCs se definen como sistemas paralelos basados en reglas, de paso de mensajes, que son capaces de interactuar con el entorno y de aprender reforzadamente mediante la asignación de recompensas y el descubrimiento de reglas. Típicamente opera en entornos tales como la robótica, sistemas económicos, juegos como el ajedrez, etc., en los que se exhiben algunas de las siguientes propiedades: la aparición constante de eventos nuevos, acompañados por grandes cantidades de ruido o datos irrelevantes continuos, a menudo en tiempo real, requerimientos para actuar:

La estructura más común de un SC esta compuesta de tres partes:

I. *El Sistema de Rendimiento:* desarrolla actividades como la interacción con el medio ambiente y el procesamiento de mensajes.
II. *El Sistema de Asignación de Crédito (AC):* desarrolla la actividad de aprendizaje mediante la modificación de pesos.
III. *El Sistema de Descubrimiento de Clasificadores:* desarrolla la actividad de aprendizaje mediante el descubrimiento de nuevas reglas.



1.4.1.1 Sistema de Rendimiento

Es el encargado de interactuar con el exterior. Esta compuesto de seis elementos básicos:

1. *Interfase de Entrada.* Esta formada por detectores (al menos uno) que se encargan de captar el estado actual del entorno y codificarlo en mensajes estándar *(mensajes de entrada).* Normalmente, todos los mensajes suelen tener la misma longitud y se representan usando el alfabeto binario {0,1}, es decir, los mensajes se representan mediante cadenas binarias de longitud fija.

2. *Interfase de Salida.* Esta formada por efectores (al menos uno) que se encargan de convertir los *mensajes de acción o salida* en acciones que actúan modificando el estado del entorno.

3. *Lista de Clasificadores.* Es el conjunto de reglas o clasificadores *(classifiers)* que representa el conocimiento que posee el sistema de aprendizaje. Cada clasificador se representa como una cadena de símbolos definidos sobre el alfabeto {0,1, #} con el formato *Condición/Acción,* de modo que la parte condición especifica los mensajes que satisfacen o activen el clasificador *(mensajes externos) y la* parte acción especifica los mensajes *(mensajes internos)* que son enviados cuando el clasificador es satisfecho. Normalmente en cada ciclo de acción del SC un número indeterminado de clasificadores se dispara en paralelo.

4. *Sistema Emparejador de Patrones.* Es el encargado de emparejar los mensajes con las condiciones de los clasificadores a identificar que clasificadores se satisfacen o cubren.

5. *Lista de Mensajes.* Contiene todos los mensajes que existen en cada momento circulando en el SC, tanto los generados en la interfase de entrada como, los generados por los clasificadores que disparan.

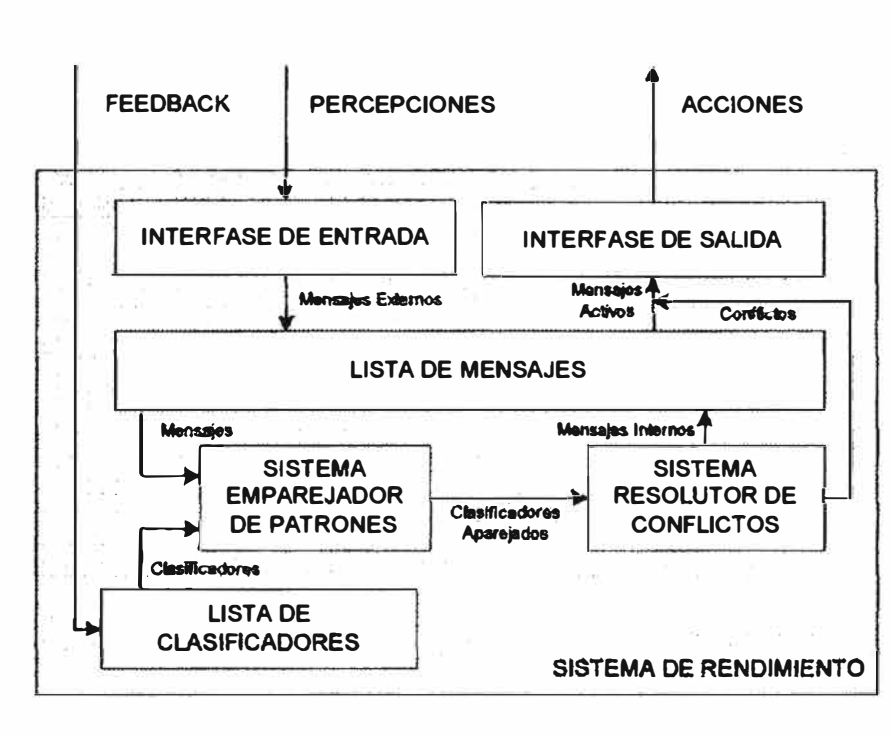
6. *Sistema Resolutor de Conflictos (RC).* Es el encargado de resolver los conflictos que puedan surgir entre los clasificadores en un ciclo de funcionamiento del SC. Se pueden presentar dos situaciones conflictivas:

   - Cuando el número de clasificadores satisfechos es mayor que el número que puede albergar la lista de mensajes.

- Cuando las acciones propuestas a los efectores por los mensajes de acción colocados por los clasificadores son inconsistentes, por ejemplo "alinear a la derecha" y a la vez exigir "alinear a la izquierda".

En ambos casos el sistema RC decide que clasificadores considera tomando como parámetro alguna medida de utilidad asociada a los mismos, como puede ser, por ejemplo, la adecuación a la situación y el peso asociado a la utilidad pasada.

El sistema RC, estableciendo una analogía económica, funciona como un mercado de oferta y demanda entre los clasificadores. Los clasificadores emparejados ofertan una porción de sus medidas de utilidad y, los conflictos son resueltos usando una distribución de probabilidad sobre esas ofertas. Los clasificadores que ofertan una mayor porción tienen mas posibilidades de disparar, y por tanto, de poner sus mensajes en la lista de mensajes. Cuando un clasificador dispara ve decrementado su peso asociado en un valor equivalente al ofertado. Finalmente, cuando se obtiene una recompensa del entorno los clasificadores que han disparado se la reparten de acuerdo a su participación en la acción desencadenada usando algún algoritmo de AC. Es por ello quo se dice que los sistemas de RC y AC constituyen el *motor de inferencia* de los SCs y por tanto, sus actividades están interrelacionadas.



1.4.1.2 Sistema de Asignación de Crédito

El aprendizaje de un SC esta compuesto de dos procesos:

- *El proceso de aprendizaje desarrollado en el sistema de AC.*
- *El proceso de aprendizaje desarrollado en el sistema de descubrimiento de clasificadores.*

En el primero, se aprende el uso del conjunto de clasificadores por medio de la recompensa *(feedback)* procedente del entorno y, en el segundo, el aprendizaje consiste en crear nuevos, y posiblemente mas útiles clasificadores, usando la experiencia pasada, es decir, los antiguos clasificadores. Por tanto, la principal tarea del sistema de AC incluye la actividad del aprendizaje por modificación y ajuste de los pesos de los clasificadores.

Tradicionalmente se vienen usando dos diferentes esquemas para controlar la acción del sistema de AC:

1.4.1.2.1 El Algoritmo Bucket Brigade (ABB).

Es un esquema de aprendizaje local que precisa pocos requerimientos computacionales, tanto de memoria como de tiempo de CPU.

En un SC con este esquema, el ABB se enlaza con el sistema de RC, formando el mecanismo que conduce la competencia entre los clasificadores del SC, conocido con el nombre del sistema AC/RC. El ABB en su forma estándar funciona como explicamos a continuación. En cada proceso de competición, cada clasificador C_j hace su oferta Bid_j,

$$Bid_j = b * Str_j * Sped_j \text{ "sin soporte"}$$

ó

$$Bid_j = b * Str_j * Sped_j * Sup_j \text{ "con soporte"}$$

donde b es una constante pequeña llamada factor de riesgo, Str_j es el peso del clasificador C_j (inicializado con el mismo valor para todos los clasificadores), $Spec_j$ es la especificidad de C_j (definida como el número de 0s y 1s de la parte condición dividido por la longitud total de la parte condición), y

$$Supj = \sum_{m \in M_i (m \in M_j)} Int(m)$$

es el soporte de Cj, donde Tj es el conjunto de todos los mensajes de la lista de mensajes que satisfacen a Cj y, Int(m) es la intensidad del mensaje m, esto es, un valor igual a la oferta del clasificador que envió el mensaje m. La probabilidad de que un clasificador gane la competición viene dada por la expresión siguiente:

$$\frac{Bidj}{\sum_{Ck \in S} Bidk}$$

Donde S es el conjunto de todos los clasificadores satisfechos. Un clasificador ganador C_j, reduce su peso en el valor de su oferta, es decir,

$$Str_j = Str_j - Bid_j;$$

y esa cantidad Bid_j se reparte entre sus predecesores, es decir, entre los clasificadores cuyas actividades anteriores posibilitan que C_j sea activo, de acuerdo a la siguiente expresión:

$$Stri = Stri + \frac{Bidj}{|Pj|} \quad \forall Ci \in Pj$$

donde P_j es el conjunto de predecesores de C_j , esto es,

$$Pj = \{Ck : \exists T \in Tj, \exists m \in T(\text{Ck envio m})\}$$

Lógicamente si un clasificador no oferta lo suficiente para ganar, el no paga nada. Cuando se recibe una recompensa externa, esta es repartida

igualmente entre los clasificadores que enviaron los mensajes acción que activaron los efectores.

1.4.1.2.2 *El Plan de Profit-Sharing (PPS).*

Es un esquema de aprendizaje global que normalmente consigue mejor rendimiento que el ABB. Los procesos de oferta y de selección de los clasificadores ganadores son hechos igual que con el ABB. Las recompensas del entorno son recibidas tras la finalización de un *episodio,* donde un episodio es definido como el intervalo de tiempo entre la recepción de dos recompensas externas consecutivas. Cuando se recibe una recompense externa *Ext,* esta es repartida de modo que el peso Str_j de cada C_j que haya estado activo al menos una vez durante el episodio es modificado de acuerdo a la siguiente expresión,

$$Str_i = Str_j - Bid_j + b * Ext.$$

Una diferencia fundamental entre el ABB y el PPS es que el primero actualiza los pesos de los clasificadores al final de un ciclo de ejecución del SC, mientras que el PPS actualiza los pesos de los clasificadores al final de un episodio.

1.4.1.3 El Sistema de Descubrimiento de Clasificadores

El proceso de descubrimiento de clasificadores en un SCs se realiza por AGs, es decir, el sistema desarrolla su proceso de aprendizaje generando nuevos clasificadores aplicando un AG sobre el conjunto de clasificadores. Básicamente el funcionamiento es como sigue: un AG selecciona los clasificadores de mayor adecuación o fitness como padres para que a partir de ellos, formen nuevos clasificadores usando los operadores genéticos y fragmentos de su composición. El fitness de un clasificador es determinado por su peso asociado, asignado por el sistema de AC, en vez de una función de adecuación. En los clásicos SCs la población del AG suele ser una porción del conjunto de clasificadores (aquellos con fitness mas alto) con el fin de preservar el sistema de rendimiento, el AG solo puede reemplazar un subconjunto del conjunto total de clasificadores, es decir, un subconjunto de los m peores clasificadores se reemplaza.

El sistema de descubrimiento de clasificadores entra en funcionamiento cuando el sistema AC alcanza una situación de "steady-state" (equilibrio o uniformidad), es decir, cuando el peso de cada clasificador refleja realmente su utilidad. Esto suele suceder normalmente entre 1000 y 10000 ciclos de aplicación del sistema AC, y por tanto con una frecuencia muy baja. Un ciclo de ejecución de un SC consta de los siguientes pasos:

Paso 1	Tomar al conjunto de clasificadores como la población inicial P.
Paso 2	Ordenar los individuos de la población P en orden decreciente de acuerdo a su fitness asociado.
Paso 3	Elegir *2k* individuos a ser reemplazados de entre aquellos de más baja adecuación.
Paso 4	Elegir *k* parejas de individuos para ser reproducidos de entre aquellos de mayor adecuación.
Paso 5	Aplicar los operadores genéticos a los *k* pares de individuos seleccionados en el paso 4, creando nuevos clasificadores descendientes.
Paso 6	Reemplazar los *2k* individuos seleccionados en el paso 3 con los descendientes creados en el paso 5.

1.4.2 Funcionamiento básico de un SC

Todos los componentes de un SC entran en funcionamiento en un ciclo de ejecución formado por los siguientes pasos:

Paso 1	Inicialmente, un conjunto de clasificadores es creado aleatóriamente o por algún algoritmo que tiene en cuenta alguna información del dominio del problema y, a todos se ha asignado el mismo peso.
Paso 2	La interfase de entrada codifica las señales de entrada como mensajes externos.
Paso 3	Adicionar todos los mensajes externos a la lista de mensajes.
Paso 4	El sistema de emparejamiento de patrones determina el conjunto de clasificadores que son empatados por algún mensaje de la lista de mensajes.

Paso 5	El sistema CR resuelve los conflictos entre los clasificadores emparejados o cubiertos y determina el conjunto de clasificadores activos.
Paso 6	Borrar la lista de mensajes.
Paso 7	Colocar los mensajes disparados por los clasificadores activos en la lista de mensajes.
Paso 8	Permite que los efectores que se activen por algún mensaje de la lista de mensajes efectúen sus acciones sobre el entorno. En caso de conflicto o inconsistencia llamar al sistema CR.
Paso 9	Si una señal de recompensa se detecta, distribuirla con el sistema de AC.
Paso 10	Si el sistema de AC esta en un estado de steady-state, aplicar el sistema de descubrimiento de clasificadores sobre el sistema de rendimiento.
Paso 11	Volver al paso 2

CAPITULO II

DESCRIPCIÓN DEL PROBLEMA

2. DESCRIPCIÓN DEL PROBLEMA

En la actualidad, debido a que se vive un periodo de globalización, el mercado exige a las empresas producir productos de buena calidad con bajos costos y mejorar la atención a los clientes, de acuerdo a estándares internacionales de calidad. Debido a que competimos en un mercado global, las empresas están obligadas a contar con procesos eficientes y eficaces en todos los niveles de su organización. Esto conlleva inevitablemente a racionalizar y asignar eficientemente los recursos de la empresa, conceptos tales como calidad total, just in time, los estándares ISO en general y otros son sólo reflejo de esta necesidad.

Un problema común en todas las áreas de la empresa, es la asignación de recursos y tareas, por ejemplo:

Se puede observar como problemas de asignación recursos en el proceso productivo; tales como asignación de máquinas a las diferentes tareas a realizar, asignación de las materias primas a los trabajos, elaboración de turnos y horarios de los recursos humanos, determinación del orden de ejecución de las tareas para cumplir con las fechas de entrega de cada pedido. También se encuentra presente en problemas de venta y distribución, tales como la asignación de cartera de clientes y rutas a los funcionarios en el departamento de ventas; asignación y

planificación del transporte de los productos a los clientes. En aspectos logísticos tales como el almacenamiento de materias primas y productos terminados. En la atención al cliente, como la atención de reclamos. E incluso en temas puramente organizacionales como la elaboración de cronogramas de vacaciones y workflow en general. Y muchas otras formas de presentación de este problema.

Estos problemas crecen en complejidad conforme pasa el tiempo, debido a la tendencia de crecimiento de las empresas. Así, antaño las empresas tenían un tamaño reducido de clientes, volúmenes de producción no muy elevados y la misma empresa contaba con pocos empleados. En esas condiciones, se podía resolver en forma manual problemas de determinación de rutas de distribución, asignación de tareas y determinación del orden de ejecución en la línea de producción y otros, solo para citar algunos ejemplos mencionados anteriormente.

En la actualidad, las empresas enfrentan una mayor dificultad para resolver estos problemas, haciendo necesario el uso de los sistemas computacionales.

En la última década el gobierno peruano a demostrado su interés en la apertura de los mercados y la participación de empresas peruanas en los mercados internacionales. Para que las empresas puedan beneficiarse de esta oportunidad de negocio, es necesario que mejoren la calidad de sus productos y procesos para cumplir con los estándares internacionales. Para lograr esto creemos que el problema de asignación de recursos y tareas es fundamental para el logro de este objetivo. En el Perú la mayoría de empresas no cuentan con mucha experiencia en la asignación eficaz de los recursos y tareas en el proceso productivo, debido fundamentalmente a dos razones: la primera razón es que las empresas no cuentan

o asignan los recursos necesarios para la contratación de personal calificado que desempeñe esta labor, la cual es realizada artesanalmente por personal no calificado. La segunda razón es que no se cuenta con herramientas apropiadas que sirvan de apoyo en esta labor.

En la presente tesis, se desarrolló este problema orientado al proceso productivo de las industrias, específicamente a líneas de producción, debido a su alto grado de complejidad.

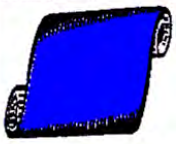
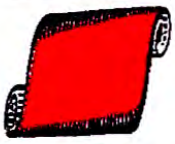
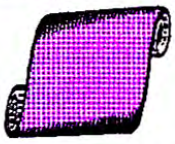
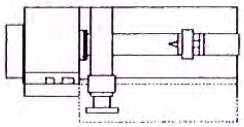
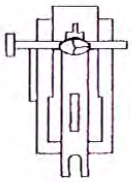
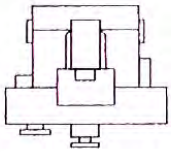
2.1 CARACTERÍSTICAS

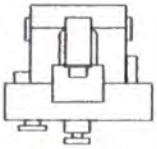
El problema elegido tiene las siguientes características:

1. La empresa produce T diferente tipos de productos.
2. La empresa cuenta con M máquinas.
3. Los productos pueden ser realizados en cualquier máquina.
4. La eficiencia de cada máquina para producir un determinado tipo de producto Ti es variable de máquina en máquina.
5. La empresa tiene P pedidos que pueden ser mucho mayores al número de máquinas.
6. Cada pedido de producción es de un solo tipo de producto y el número de unidades a producir es variable.
7. Cada pedido cuenta con un plazo de entrega diferente.

8. Existe un tiempo de carga variable para preparar la máquina Mh que realizo anteriormente un pedido del tipo de producto Ti, para que produzca otro pedido del tipo de producto Tj.

El objetivo de los modelos, es encontrar la mejor solución posible para el problema planteado, produciendo todos los productos evitando o minimizando las demoras en las entregas de los pedidos y a su vez minimizando el tiempo total de producción.

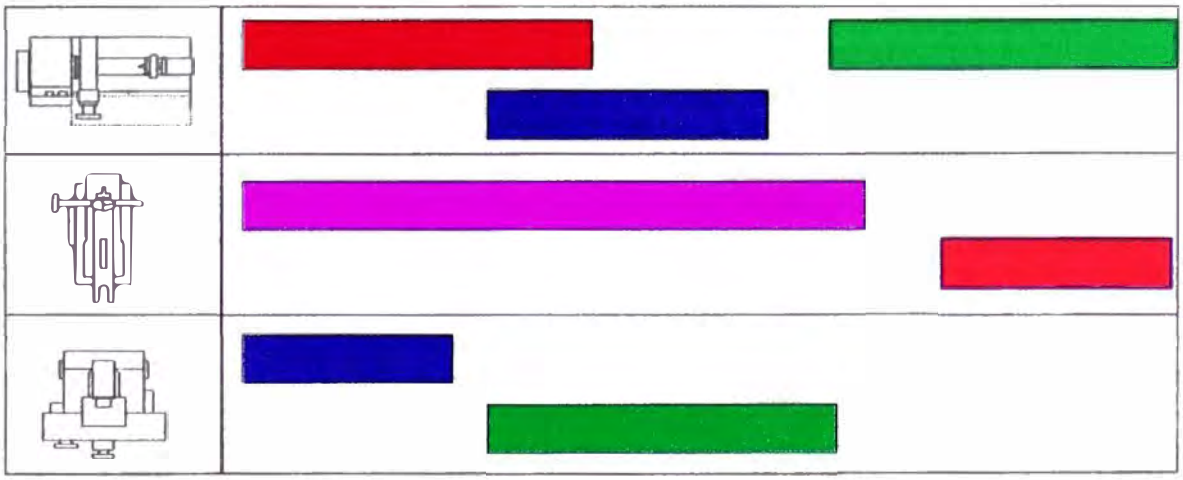
Tipos de Producto / Maquinas				
	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{1,1}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{1,2}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{1,3}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{1,4}$
	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{2,1}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{2,2}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{2,3}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{2,4}$
	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{3,1}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{3,2}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{3,3}$	• Puede Producirlo: Si • Eficiencia(tiempo produccion unidad): $TP_{3,4}$



Maquina 1

	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{1,1}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{1,2}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{1,3}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{1,4}$
	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{2,1}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{2,2}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{2,3}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{2,4}$
	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{3,1}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{3,2}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{3,3}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{3,4}$
	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{4,1}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{4,2}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{3,4}$	• Tiempo de preparacion de la maquina(tiempo de carga)$TC_{4,4}$

Pedidos							
	• Tipo Producto • Cantida C1 • Tiempo de entrega T1	• Tipo Producto • Cantida C2 • Tiempo de entrega T2	• Tipo Producto • Cantida C3 • Tiempo de entrega T3	• Tipo Producto • Cantida C4 • Tiempo de entrega T4	• Tipo Producto • Cantida C5 • Tiempo de entrega T5	• Tipo Producto • Cantida C6 • Tiempo de entrega T6	• Tipo Producto • Cantida C7 • Tiempo de entrega T7



2.2 CASOS DE ESTUDIO

Para efectos prácticos los métodos fueron evaluados con tres casos de estudio, los cuales son:

Caso de Estudio 1:

Se consideró 12 pedidos de 3 tipos de producto diferentes a ser ejecutados en 3 máquinas.

Caso de Estudio 2:

Se consideró 24 pedidos de 3 tipos de productos diferentes a ser ejecutados en 3 máquinas.

Caso de Estudio 3:

Cuatro experimentos considerando 10, 15, 20 y 40 pedidos a ser ejecutados en 10, 15, 20 y 40 máquinas respectivamente.

2.3 CUANTIFICACION DE LOS CASOS DE ESTUDIO

Los autores propusieron las siguientes configuraciones para los casos de estudio a ser analizados.

2.3.1 Caso 1

Se trabaja con 3 tipos de productos: producto 1, 2 y 3

Atendiendo 12 pedidos:

Pedido	Producto	Cantidad (expresado en unidades de producto)	Tiempo de entrega (expresado en unidades de tiempo)
1	1	1	7
2	2	2	12
3	3	3	17
4	3	4	39
5	2	5	39
6	1	6	39
7	1	1	46
8	2	2	51
9	3	3	56
10	3	4	78
11	2	5	78
12	1	6	78

Se dispone de 3 máquinas:

Máquina	Producto	Tiempo de producción de una unidad de producto(expresado en unidades de tiempo)
1	1	5
	2	10
	3	10
2	1	10
	2	5
	3	10
3	1	10
	2	10
	3	5

Máquina	Producto producido	Producto a producir	Tiempo de Carga (expresado en unidades de tiempo)
1	1	1	2
	2	2	2
	3	3	2
2	1	1	2
	2	2	2
	3	3	2
3	1	1	2
	2	2	2
	3	3	2

2.3.2 Caso 2

Se trabaja con 3 tipos de productos: producto 1, 2 y 3

Atendiendo 24 pedidos:

Pedido	Producto	Cantidad (expresado en unidades de producto)	Tiempo de entrega (expresado en unidades de tiempo)
1	1	1	7
2	2	2	12
3	3	3	17
4	3	4	39
5	2	5	39
6	1	6	39
7	1	1	46
8	2	2	51

9	3	3	56
10	3	4	78
11	2	5	78
12	1	6	78
13	1	1	85
14	2	2	90
15	3	3	95
16	3	4	117
17	2	5	117
18	1	6	117
19	1	1	124
20	2	2	129
21	3	3	134
22	3	4	156
23	2	5	156
24	1	6	156

Se dispone de 3 máquinas:

Máquina	Producto	Tiempo de producción de una unidad de producto(expresado en unidades de tiempo)
1	1	5
	2	10
	3	10
2	1	10
	2	5
	3	10
3	1	10
	2	10
	3	5

Máquina	Producto producido	Producto a producir	Tiempo de Carga (expresado en unidades de tiempo)
1	1	1	2
	2	2	2
	3	3	2
2	1	1	2
	2	2	2
	3	3	2
3	1	1	2
	2	2	2
	3	3	2

CAPITULO III

DESCRIPCIÓN DEL MODELO

3. DESCRIPCIÓN DEL MODELO

La complejidad de los problemas de asignación de recursos y tareas planteados, hace muy difícil la implementación de sistemas de información que tengan la capacidad de resolver estos problemas, haciendo necesario para ello el uso de técnicas más elaboradas.

El primer paso se logró integrando métodos de investigación de operaciones los cuales demostraron ser muy efectivos para problemas específicos, y de un grado de complejidad limitado. Sin embargo, esta solución no fue adoptada plenamente por las empresas, debido fundamentalmente a que estas enfrentaban problemas de una complejidad mucho mayor.

Otros métodos alternativos son las búsquedas heurísticas. Las búsquedas heurísticas representan una mejora de las búsquedas exhaustivas, haciendo uso de una función de evaluación llamada función heurística para dirigir el proceso de búsqueda, evitando recorrer o seguir buscando alternativas que la función considera no viables. Estas búsquedas no aseguran encontrar la solución óptima, por la sencilla razón de que sólo evalúan una pequeña fracción de las posibilidades. La eficiencia de la solución depende de la función heurística, y el grado de complejidad de su implementación depende proporcionalmente al grado de

complejidad del problema. Normalmente estas búsquedas son implementadas para brindar la primera solución posible que encuentre, evitando la carga de procesamiento que implica la búsqueda exhaustiva.

Actualmente muchos países desarrollados hacen uso extensivo de las diferentes metodologías de inteligencia artificial para resolver problemas complejos similares al grado de complejidad del problema propuesto. Entre las metodologías que podrían ser usadas para resolver este problema se tienen: Computación Evolutiva y Sistemas Multiagentes. Como explicamos anteriormente, la computación evolutiva tiene la capacidad de encontrar soluciones optimas, usando modelos algorítmicos que simulan el proceso de evolución natural. Estos métodos, a diferencia de los heurísticos, no descartan a priori las alternativas del espacio de solución, sino que intentan encontrar mejores soluciones permanentemente, mostrando un buen desempeño en el proceso de búsqueda. A su vez, los sistemas multiagentes permiten modelar sistemas complejos de una manera sencilla, distribuyendo la compleja búsqueda total en agentes que realizan búsquedas parciales mucho más sencillas y fáciles de implementar, dejando la dificultad de la búsqueda en los procesos de coordinación de éstos agentes.

Los modelos evaluados son los siguientes:

1. Búsqueda exhaustiva,
2. Algoritmos genéticos puros,
3. Sistemas multiagentes con sistemas clasificadores; y
4. Sistemas multiagentes con sistemas clasificadores y estrategias evolutivas.

3.1 MÉTODO DE BÚSQUEDA EXHAUSTIVA

El método de búsqueda exhaustiva es un método de búsqueda enumerativo. Como se explicó en el fundamento teórico, este método evalúa todas las posibles soluciones con el fin de encontrar la solución óptima al problema. Para el problema planteado en la presente tesis, se determinó el grado de complejidad de la búsqueda.

3.1.1 Complejidad de la búsqueda

Para estimar la complejidad de la búsqueda, inicialmente se redujo el grado de complejidad del sistema, para luego generalizar la fórmula considerando el total de las restricciones del problema.

Caso 1: P Pedidos, 1 Máquina

En este caso, se consideró inicialmente que se tiene solo una máquina y P pedidos. El número de posibles soluciones estaba dado solamente por la secuencia en la que se realizarían los pedidos. Así, si tenemos P pedidos con tipos de producto, cantidades y tiempos de entrega diferentes se tuvo:

Secuencia	$Pedido_1$	$Pedido_2$	...	$Pedido_{P-1}$	$Pedido_P$
Posibilidades	P	P-1		2	1

Para el primer pedido a ejecutar, se pudieron escoger cualquiera de los P pedidos, para el siguiente fueron P-1 posibilidades, así hasta para el último pedido en la secuencia, donde solamente quedó una única posibilidad. La fórmula obtenida en este caso fue:

$$P!$$

Caso 2: P Pedidos, 2 Máquinas

En este caso, el modelo se complicó debido a que cada pedido pudo ser realizado en cualquiera de las dos máquinas, lo cual pudo originar la formación dos grupos de pedidos, que pudieron ser ejecutados en las máquinas correspondientes.

	Comb. 1	Comb. 2		Comb. P	Comb. P+1
Maq. 1	0	1	...	P-1	P
Maq. 2	P	P-1		1	0

A su vez, para cada una de las combinaciones, se debieron consideran todas las posibles secuencias de los pedidos participantes de cada grupo. La fórmula obtenida fue:

$$\sum_{g=0}^{P} C_g^P \, g!(P-g)!$$

Caso 3: P Pedidos, 3 Máquinas

De igual forma, en el caso los P pedidos se debieron dividir en 3 grupos, y en cada grupo se debió considerar todas las posibles secuencias de los pedidos participantes en cada grupo.

	Comb. 1	Comb. 2		Combinación ((P+1)(P+2)/2) -1	Combinación ((P+1)(P+2)/2)
Maq. 1	0	0	...	P-1	P
Maq. 2	0	1		1	0
Maq. 3	P	P-1		0	0

La fórmula obtenida es:

$$\sum_{g1=0}^{P}\sum_{g2=0}^{P-g1} C_{g1}^{P} C_{g2}^{P-g1}\, g1!\,g2!\,(P-g1-g2)!$$

Generalizando por inducción matemática: para P Pedidos y M Máquinas se obtuvo la siguiente fórmula:

$$\sum_{g1=0}^{P}\sum_{g2=0}^{P-g1}\sum_{g3=0}^{P-g1-g2}\cdots\sum_{g(m-1)}^{P-\sum_{i=1}^{m-2} gi} C_{g1}^{P} C_{g2}^{P-g1} C_{g3}^{P-g1-g2}\cdots C_{g(m-1)}^{P-\sum_{i=1}^{m-2} gi}\left(\prod_{i=1}^{m-1} gi!\right)\left(P-\sum_{i=1}^{m-1} gi\right)!$$

Esta fórmula permitió calcular el número de posibles soluciones que deben ser evaluados para encontrar la solución óptima.

3.2 MÉTODO DE ALGORITMOS GENETICOS

Como se explicó en el fundamento teórico, las posibles soluciones del problema debieron representarse en cadenas binarias, los que fueron evaluados por la función fitness.

Genoma:

Cada individuo esta compuesto por P grupos de genes, donde cada grupo representa a un pedido en particular. A su vez, estos genes constan de dos partes, la primera parte indica la máquina donde se ejecutará el pedido y la segunda parte indica la secuencia de ejecución en esa máquina.

Pedido1		Pedido2		Pedido3		...	Pedido(P-1)		PedidoP	
M1	S1	M2	S2	M3	S3	...	M(P-1)	S(P-1)	MP	SP

Función Fitness

Para el problema planteado, se definió una función fitness que es directamente proporcional al Tiempo Promedio de Ejecución de las tareas en las M máquinas, a la sumatoria de las demoras de los P pedidos y al mayor tiempo de uso de la máquina mas usada.

$$Fitness = \alpha(\sum_{i=1}^{M} UsoMaquina_i / M) + \beta \sum_{i=1}^{P} DemoraPedido_i + \delta MAX_{i=1}^{M}(UsoMaquina_i)$$

Como el fitness es proporcional a las demoras y al tiempo de uso de la máquina se buscara minimizar el valor de esta función

Constantes del algoritmo

Población	200
Iteraciones	1000
Probabilidades	
Mutación	1%
Reproducción	10%
Crossover	89%

3.3 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES

En el capítulo 1, se observó que los sistemas multiagentes tienen gran potencial para el modelamiento de sistemas complejos, definiendo elementos autónomos básicos llamados agentes. La interacción entre estos agentes hace posible el surgimiento de comportamiento complejos. Para el problema planteado, haremos uso de esta metodología implementando un modelo, donde se definió al agente como la representación de un pedido en particular. A se describen los elementos participantes del modelo.

En el modelo se consideró que los agentes(pedidos) se movilizan libremente, determinando su tiempo de inicio y la máquina donde se ejecutarán.

Agente: Un agente representa a un pedido en particular. Los elementos de la arquitectura del agente son:

Objetivo: Satisfacer sus restricciones:

- Evitar la superposición con otros pedidos, es decir, no elegir tiempos de ejecución donde ya existan otros pedidos que se estén ejecutando en la misma máquina en dicho periodo de tiempo.
- Minimizar los retrasos en el período de entrega de este.

Acciones: Para lograr su objetivo, el agente puede hacer uso de las siguientes acciones:

- Adelantar el inicio de ejecución del pedido en cualquier máquina, donde exista un espacio libre en cual se pueda ejecutar. Este se puede apreciar con un movimiento a la izquierda.
- Retrazar el inicio de ejecución del pedido en cualquier máquina, donde exista un espacio libre en cual se pueda ejecutar.
- Cambiar de máquina y escoger un tiempo de inicio de ejecución en forma aleatoria.
- Superponer el tiempo de ejecución con el pedido anterior que se ejecuta en la misma máquina, tomando su mismo tiempo de inicio.

- Superponer el tiempo de ejecución con el pedido posterior que se ejecuta en la misma máquina, tomando su mismo tiempo de inicio.
- Cambiar a una máquina de mayor desempeño, escogiendo el tiempo de inicio límite que satisfaga sus plazos de entrega.
- Adelantar el inicio de ejecución del pedido en la máquina actual, donde exista un espacio libre en el cual se pueda ejecutar. Esto se puede apreciar con un movimiento a la izquierda.
- Retrazar el inicio de ejecución del pedido en la máquina actual, donde exista un espacio libre en cual se pueda ejecutar.
- No realizar ninguna acción.

Condiciones o Estados: Cada agente percibe parte del ambiente por medio de sus sensores para tomar una acción. Cada uno de estos sensores, están diseñados para identificar el estado de cada una de sus restricciones, esta información es traducida por el agente y comparada con las condiciones de las reglas de comportamiento de éste. Para este modelo se definió las siguientes condiciones:

- Condiciones de superposición y condición de retrazo insatisfechas.
- Condición de superposición insatisfecha y condición de retrazo satisfecha.
- Condición de superposición satisfecha y condición de retrazo insatisfecha.
- Condiciones de superposición y condición de retrazo satisfechas.

Conocimiento: Es el conjunto de reglas de comportamiento con sus respectivas probabilidades de ejecución. Cada regla consta de una acción asociada a una condición. Para la actualización y generación de nuevas reglas se hizo uso de los Sistemas Clasificadores como metodología de aprendizaje.

Memoria: El agente mantiene un registro de las acciones realizadas desde su ultima valoración hecha por el ambiente.

Ambiente: El ambiente representa el problema propiamente dicho, donde interactúan los agentes. El ambiente retribuye las acciones de los agentes, otorgando una valoración de la eficiencia de las acciones realizadas por éstos, esta retribución puede ser incentivadora o inhibidora. Los elementos de la arquitectura son:

Objetivos: Los objetivos del ambiente son:

- Eliminar las superposiciones entre pedidos;
- Minimizar o eliminar las demoras en las entregas de cada pedido y finalmente,
- Minimizar el tiempo total de ejecución de los pedidos.

Memoria: Mantiene un registro de la mejor solución encontrada durante el proceso de solución del problema.

Componentes: En ambiente consta de las M máquinas y los P pedidos (agentes) que interactúan en el ambiente.

Función de valoración: Para el problema planteado, definimos la función de valoración de las acciones como una función dependiente de la mejora del tiempo total de ejecución, de la disminución del tiempo muerto, número de superposiciones y de la sumatoria de las demoras de los P pedidos.

$$Valoración = \alpha \times MejoraTiempoTot + \beta \times ReduccionTiempoMuerto - \delta \times NumeroAgtSuperpuestos - \varepsilon \times DemoraTot$$

3.4 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES Y ESTRATEGIAS EVOLUTIVAS

Uno de los puntos críticos del método anterior, fue la determinación de la función de valoración. Las constantes de esta función determinan la eficacia del modelo. Normalmente, la determinación de estas constantes es realizada por el usuario según su criterio o basándose en experimentos tipo ensayo-error. Sin embargo, este procedimiento no asegura tener un juego de constantes que optimicen la eficacia del sistemas.

Para solucionar este problema se usó una técnica de la computación evolutiva llamada Estrategias Evolutivas, que permite calcular máximos y/o mínimos de funciones desconocidas. El conjunto de coeficientes de la

función de valoración está codificado en una cadena genética de números reales, que a su vez esta sujeto a un proceso de búsqueda independiente del problema planteado.

Genoma:

Cada individuo esta compuesto por 4 números reales que representan los cuatro coeficientes que se desearon obtener.

Coeficiente1	Coeficiente2	Coeficiente3	Coeficiente4
0.0087	0.0058	0.0070	0.0045

Función Fitness:

La función fitness usada por las estrategias evolutivas, esta definida por una función que depende del tiempo total, el tiempo muerto y la demora en el plazo de entrega obtenidos por la mejor solución, además de la iteración donde se obtuvo la solución.

$$Fitness = \alpha \times MejoraTiempoTot + \beta \times ReduccionTiempoMuerto + \delta \times DemoraTot + \varepsilon \times Iteracion$$

El objetivo de este modelo es buscar los coeficientes que minimicen esta función fitness, es decir, los coeficientes que permiten encontrar la solución óptima planteada en esta tesis en el menor tiempo.

CAPITULO IV

EVALUACIÓN DE RESULTADOS

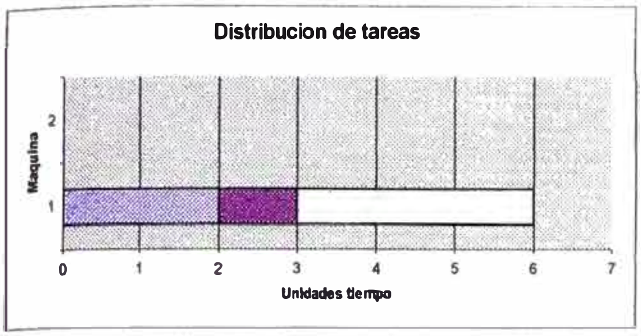
4. EVALUACION DE RESULTADOS

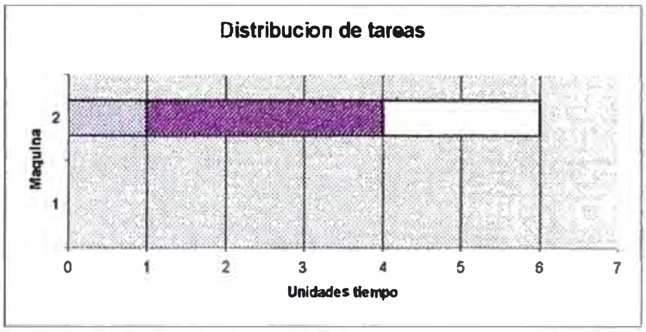
En este capítulo se muestran las evaluaciones realizadas para medir la efectividad de los métodos planteados en la resolución de problemas de asignación de recursos y tareas.

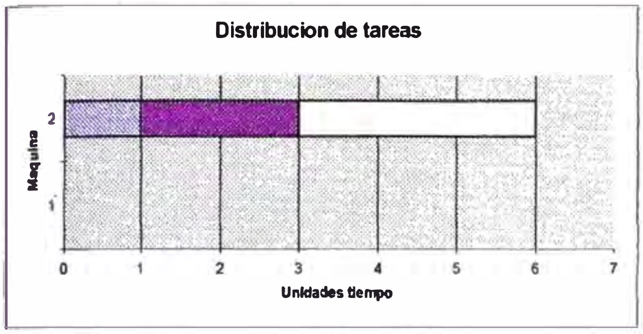
4.1 MÉTODO DE BÚSQUEDA EXHAUSTIVA

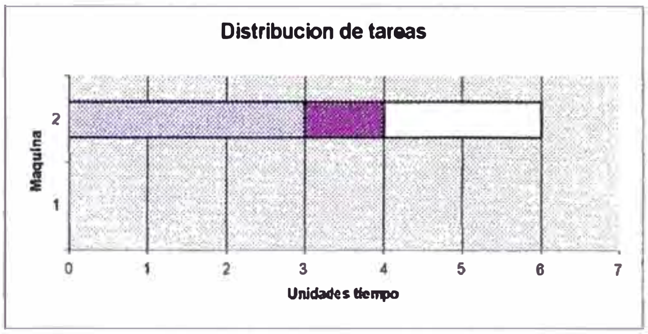
Antes de mostrar los resultados obtenidos en el caso estudio 1, los autores mostrarán con un ejemplo, el grado de complejidad del problema, evaluando el resultado de la formula, hallando todas las posibles soluciones en forma manual.

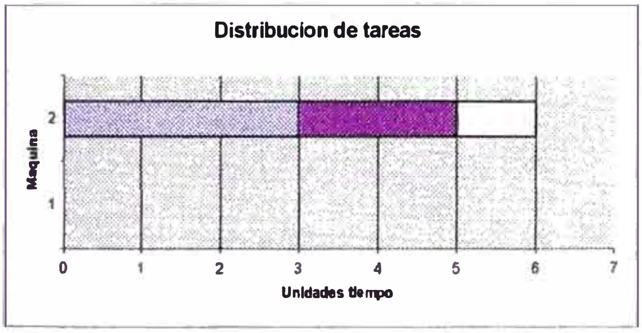
Debido a que las combinaciones de los casos 1 o 2, implicarían millones de millones de combinaciones, se evaluó el caso de 3 pedidos a ser programados en 2 máquinas. Donde cada producto tiene plazo de entrega distinto, y la eficiencia de las dos máquinas es distinta. La formula dio como resultado 24 soluciones a evaluar, que se muestran a continuación:

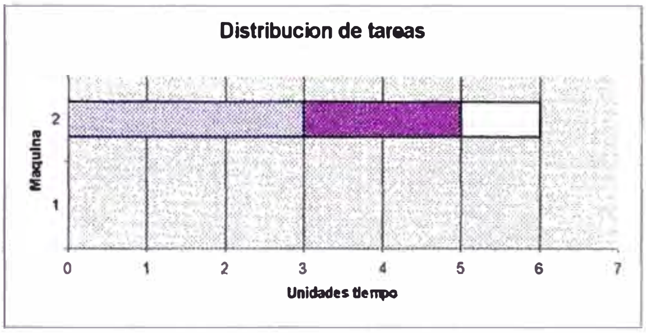

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

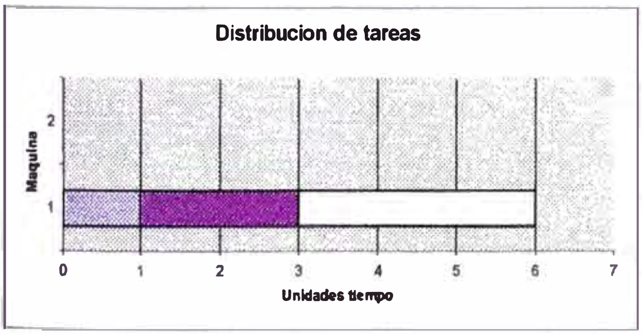

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

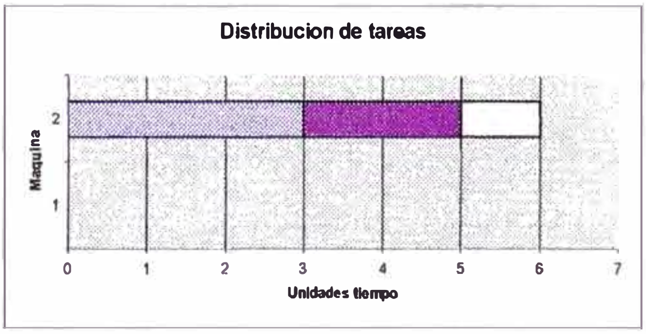

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

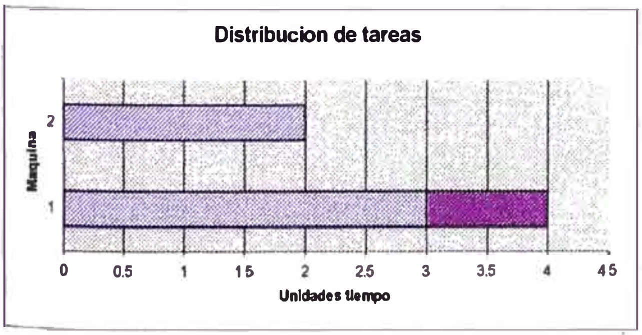

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

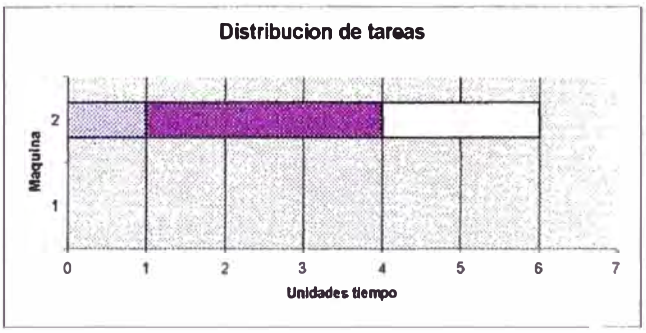

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

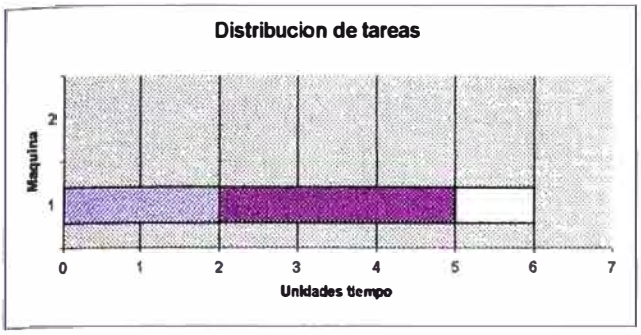

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

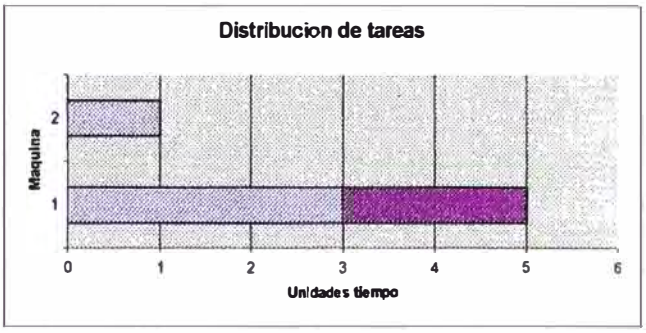

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

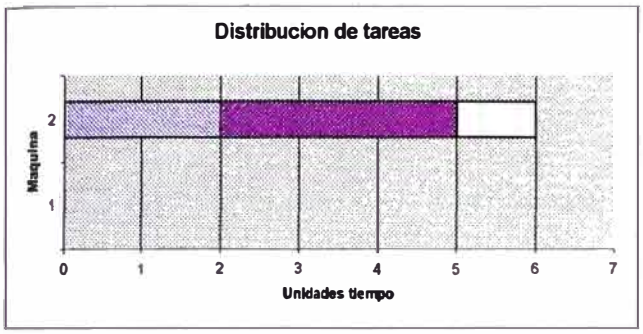

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

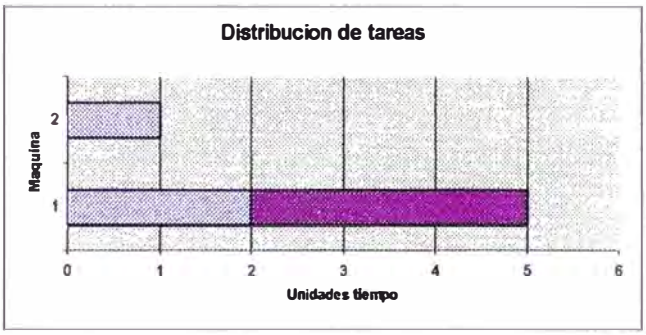

Distribucion de tareas
Maquina
2
1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
Unidades tiempo

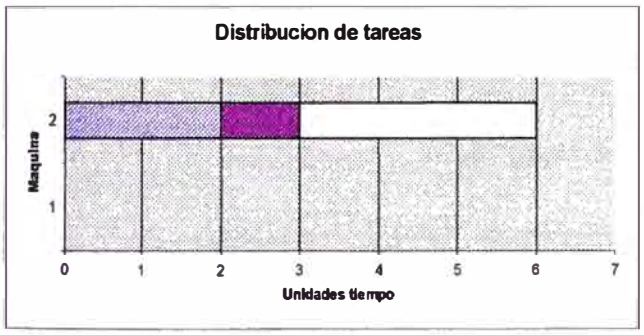

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

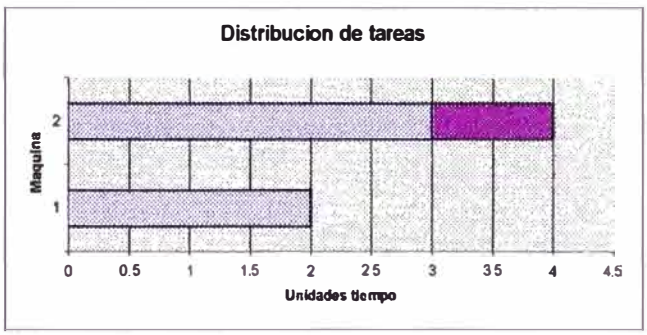

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

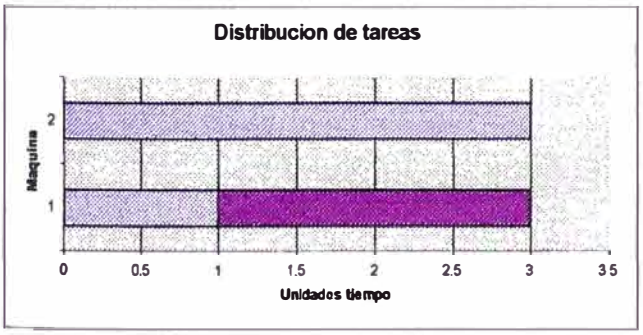

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6
Unidades tiempo

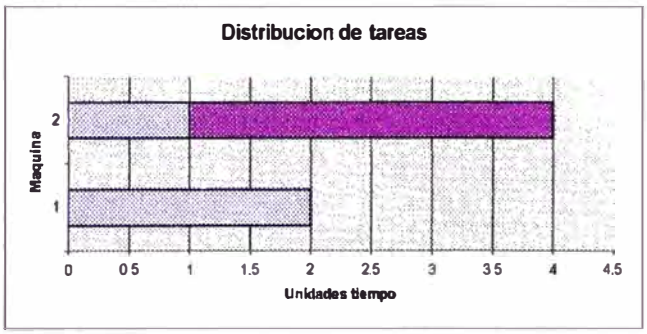

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

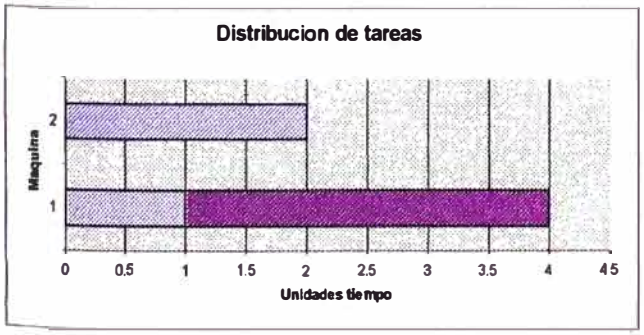

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6
Unidades tiempo

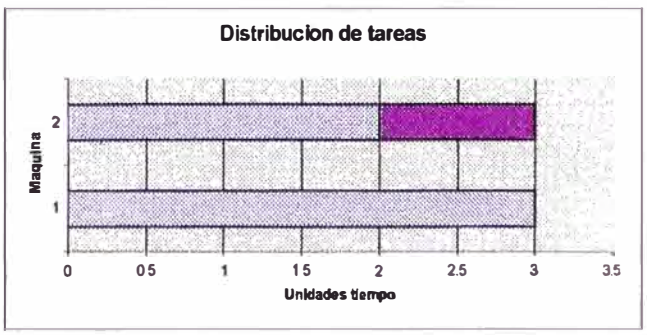

Distribucion de tareas
Maquina
2
1
0 1 2 3 4 5 6 7
Unidades tiempo

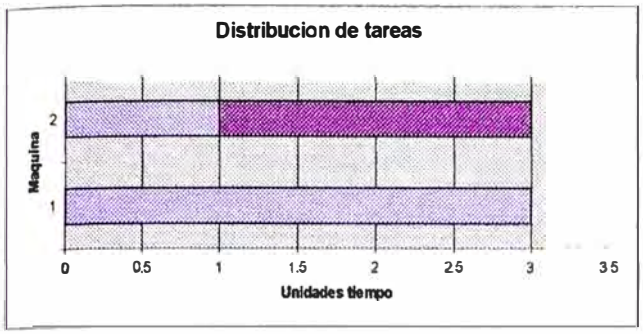

Distribucion de tareas
Maquina
2
1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
Unidades tiempo

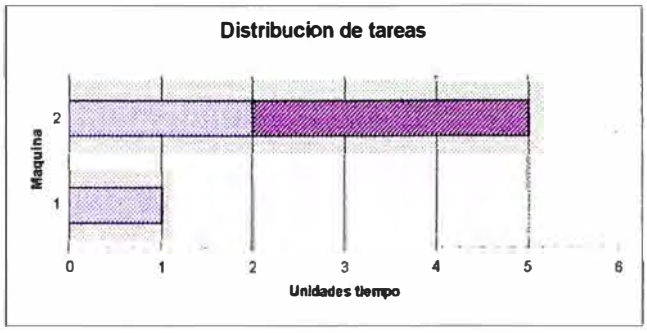

Distribucion de tareas
Maquina
2
1
0 0.5 1 1.5 2 2.5 3 3.5
Unidades tiempo

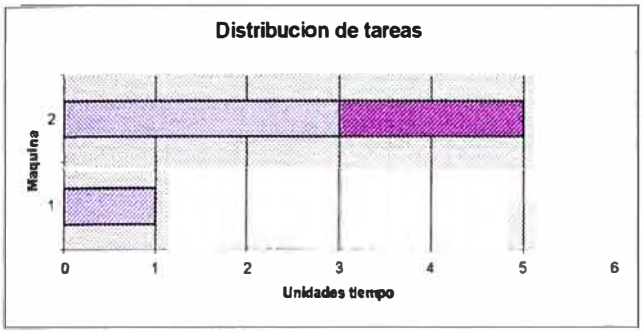

Distribucion de tareas
Maquina
2
1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
Unidades tiempo

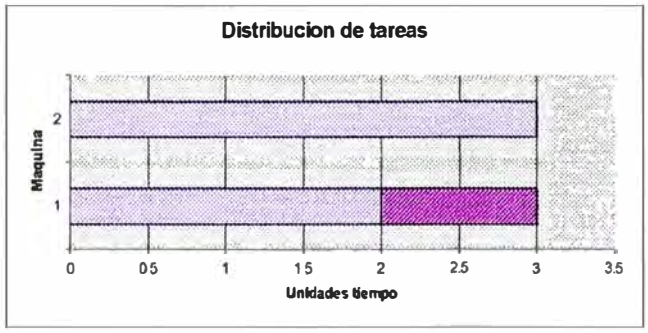

Distribucion de tareas
Maquina
2
1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
Unidades tiempo


Distribucion de tareas
Maquina
2
1
0 0.5 1 1.5 2 2.5 3 3.5
Unidades tiempo









El método de búsqueda exhaustiva tendría que evaluar cada una de estas soluciones, a fin de verificar si los plazos de entrega de cada pedido se cumplan, y además, que la solución elegida tenga el menor tiempo de producción. Considerando esto pasaremos a ejecutar la fórmula, esta vez con los casos de estudio planteados en esta tesis.

4.1.1 Resolución del Caso 1

Para el caso 1 utilizando la fórmula se obtuvo:

$$\sum_{g1=0}^{12}\sum_{g2=0}^{12-g1} C_{g1}^{12} C_{g2}^{12-g1}\, g1!\,g2!\,(12-g1-g2)! = 43{,}589'145{,}600$$

La fórmula dio como resultado 43,589'145,600 posibles soluciones que deberían ser evaluadas. Si usáramos el computador más rápido disponible, optimizando al máximo el algoritmo de evaluación de las posibles soluciones. Asumiendo que cada evaluación demoraría 0.1 milisegundos, se estimó que la búsqueda demoraría más de 1,210 horas, es decir, unos 50 días ininterrumpidos de consumo de procesador. Demostrando de esta manera la ineficiencia de este método para problemas de mediana complejidad.

4.1.2 Resolución de Caso 2

Para el caso 2 utilizando la fórmula tenemos:

$$\sum_{g1=0}^{24}\sum_{g2=0}^{24-g1} C_{g1}^{24} C_{g2}^{24-g1}\, g1!\,g2!\,(24-g1-g2)! =$$

201'645,730'563,303'000,000'000,000(aprox.)

El resultado nos da más de doscientos un cuatrillones de posibles soluciones a ser evaluadas. De igual forma que el caso anterior, estimando la evaluación de cada posible solución a 0.1 milisegundos, el proceso de búsqueda de la solución optima demoraría 639'414,417'057,659 años.

Con estos resultados se llegó a la conclusión, que el método de búsqueda exhaustiva no es aplicable para los problemas planteados en esta tesis.

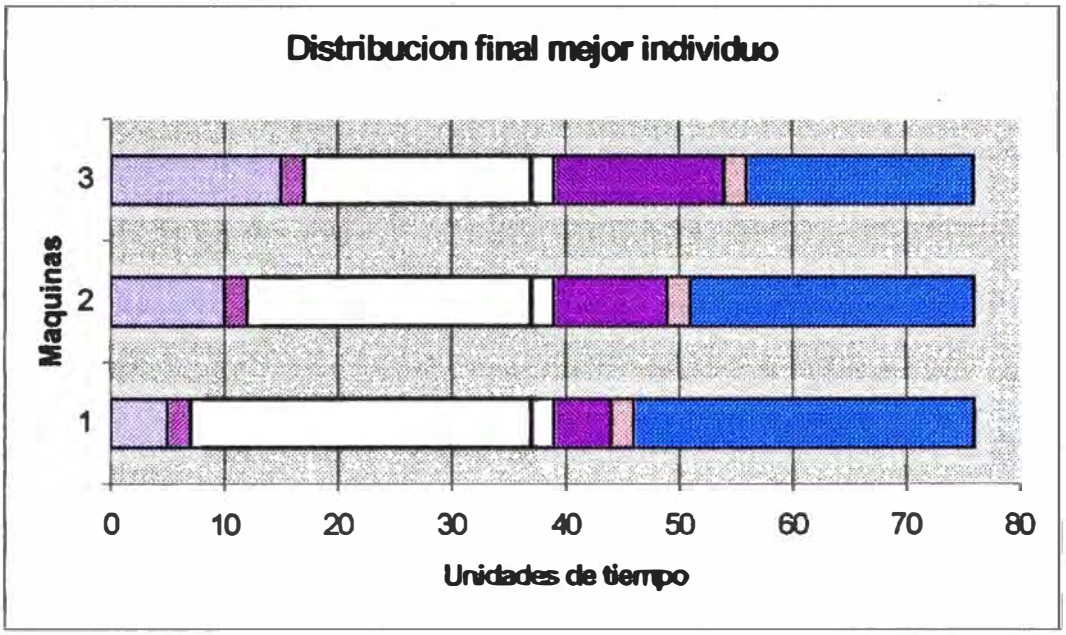
4.2 MÉTODO DE ALGORITMOS GENETICOS

4.2.1 Resolución Caso 1

Se evaluó la efectividad del método de algoritmos genéticos para hallar la solución al problema planteado en el caso 1. Para este caso se determinó los siguientes parámetros de simulación del modelo:

Población	200
Iteraciones	1000
Probabilidades	
Mutación	1%
Reproducción	10%
Crossover	89%

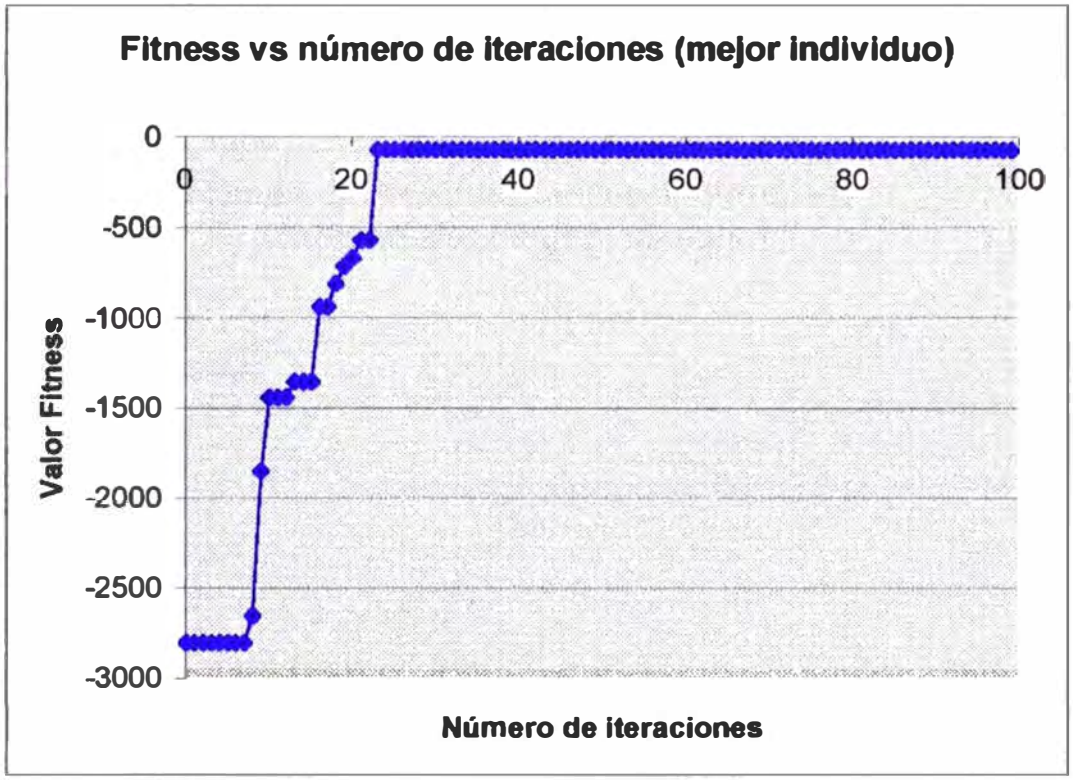
La solución optima se obtuvo en la iteración 24, y la distribución final se muestra en el siguiente gráfico. En ella se puede observar, que efectivamente el método encontró la solución óptima, la cual es la única solución entre las 43,589'145,600 posibilidades.



Considerando que el método demoró 24 iteraciones y que la población de la simulación es de 200 individuos, se puede afirmar que se realizaron 4,800 evaluaciones antes de encontrar la solución, lo cual reveló la efectividad del método. Pero ¿se puede afirmar que el resultado obtenido fue fruto del método y no de la casualidad? Para se observó el progreso de la búsqueda respecto a las iteraciones, a fin de verificar la convergencia del método de búsqueda, basándose en tres indicadores de la mejor solución obtenida en cada iteración: el valor de la función fitness, el tiempo total de proceso y las demoras en el plazo de entrega.

En el siguiente grafico se presenta la evolución de la función fitness del mejor individuo. Se pudo notar dos aspectos interesantes de esta gráfica. El primero, nunca fue decreciente, lo cual indicó que el algoritmo siempre mantuvo un individuo con un fitness igual o superior al mejor individuo de la generación anterior. El segundo aspecto, el mejor individuo no sufre mejoras continuas en el valor de la función fitness. Este hecho es similar al

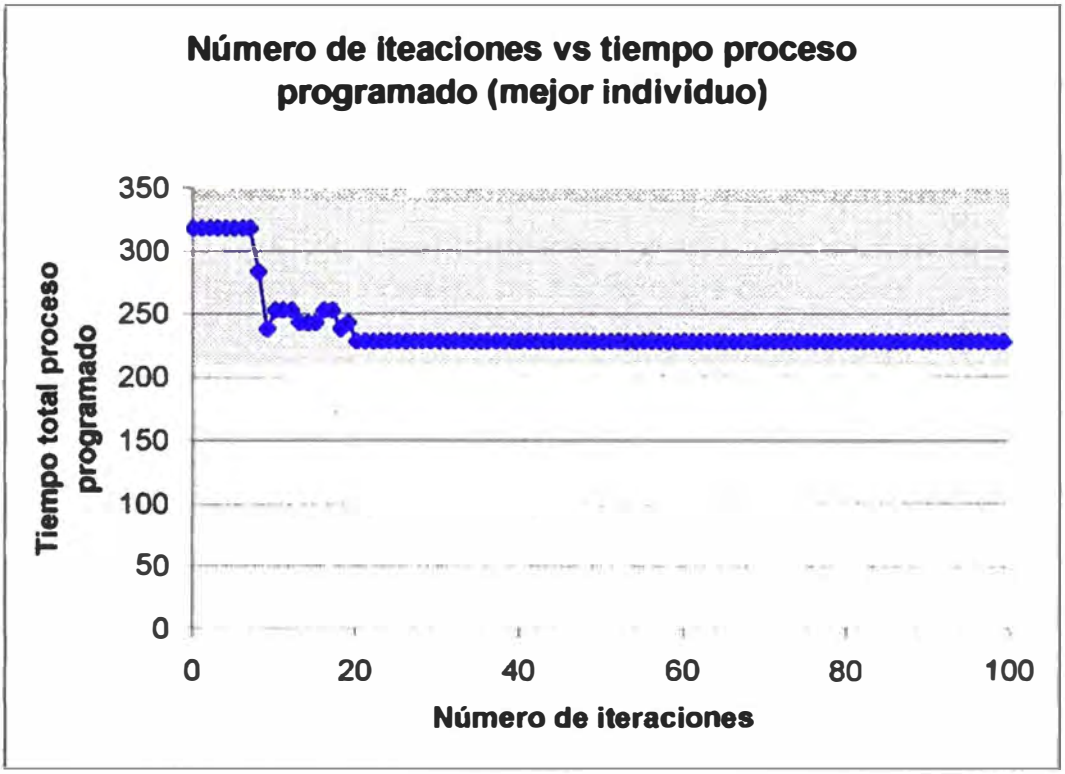
proceso evolutivo natural donde las mejoras o adaptaciones de los individuos no se dan paulatinamente sino que se sufre saltos evolutivos, seguidos de una etapa de estabilización. En la gráfica notamos el mismo comportamiento. En principio, el valor fitness se mantuvo las primeras 8 iteraciones, luego de la cual, sufrió una gran mejora cerca de la iteración 10, posteriormente notaron más mejoras pasando a la iteración 20, hasta que encuentra la solución óptima en la generación 20, manteniéndose esta solución por el resto de la simulación.

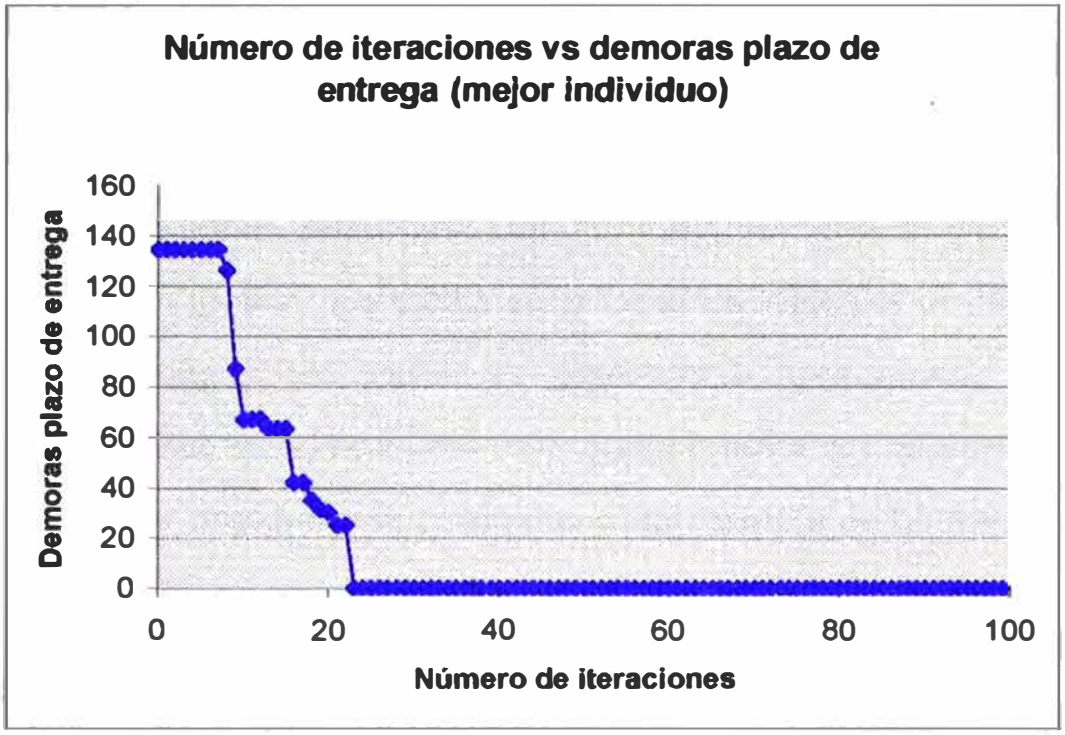


Con respecto al tiempo de proceso programado (tiempo total de trabajo de las máquinas produciendo los pedido), se observó un comportamiento similar a la evolución de la función fitness, con la diferencia que, aquí se notaron puntos en los cuales el valor de tiempo total decreció. Esto se debió a que, se le asigno un menor peso al tiempo de proceso, en

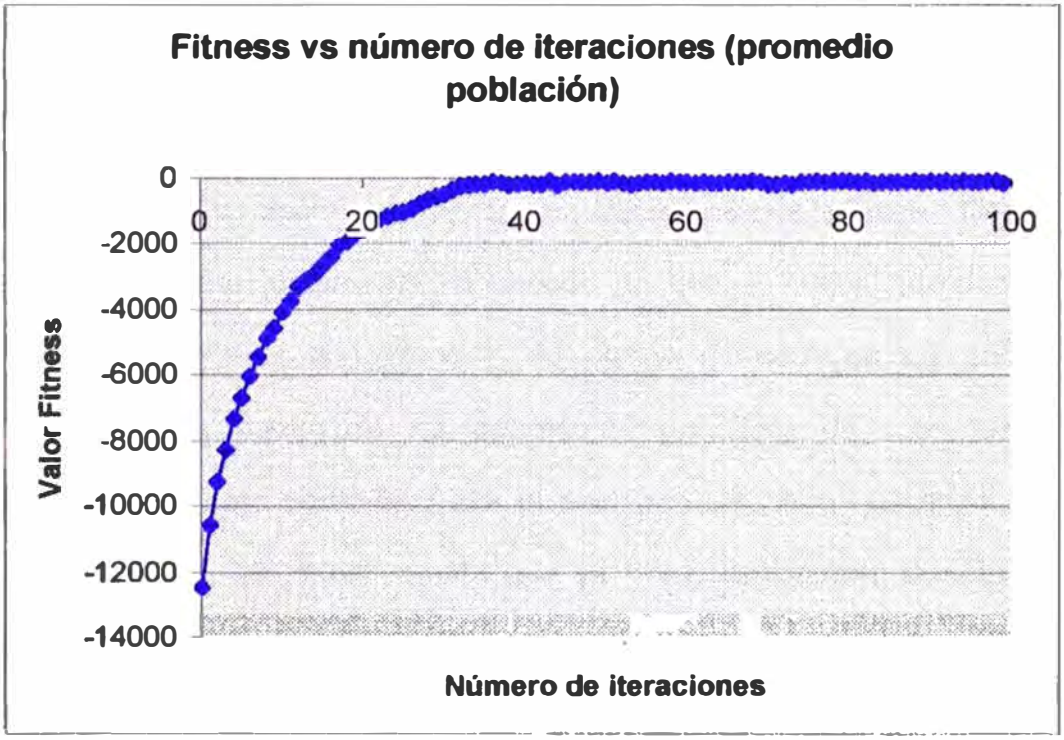
comparación, al cumplimiento de los plazos de entrega. Así como se observa en el grafico siguiente "demoras vs número de iteraciones", se constató que la iteración en la cual se incremento el valor de tiempo de proceso, correspondió a una disminución de la demora en el tiempo de entrega. En este caso la función fitness sacrificó el tiempo de producción de las máquinas, a condición de, disminuir las demoras en los pedidos.

Una vez mas se constató que después de encontrar la solución óptima en la iteración 24, no se notó una disminución o aumento en el valor de las demoras, que se mantuvieron con valor 0, o del tiempo del proceso programado; lo cual nos indicó que no se perdió la solución obtenida.





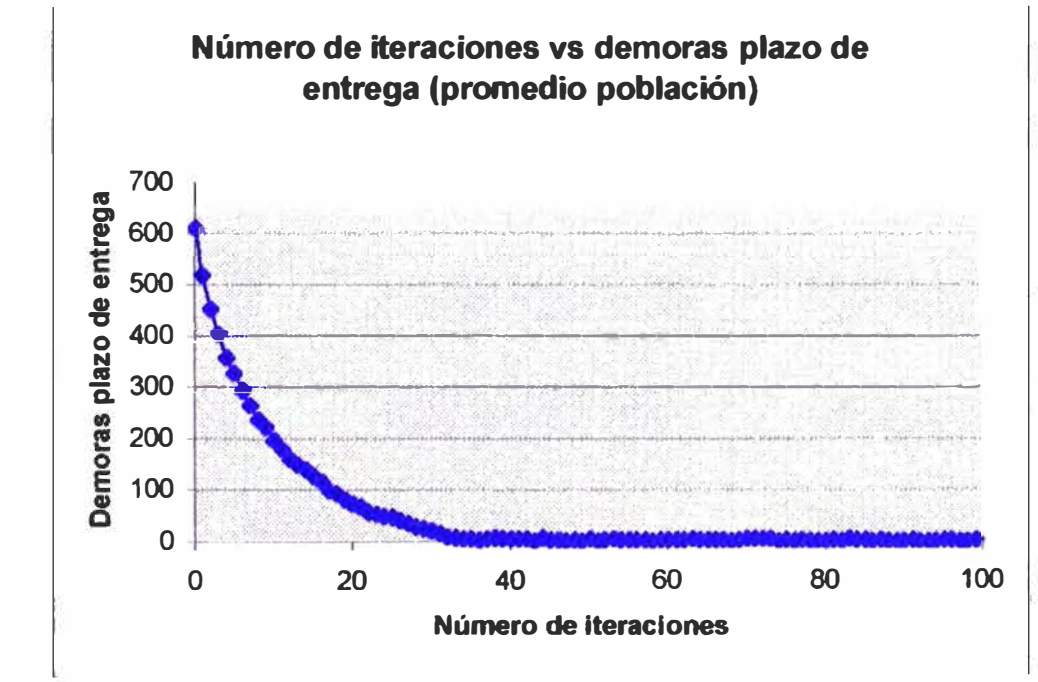
Ahora, se examinará los resultados obtenidos de la población entera; para esto se examinaron los valores promedios del fitness y de las demoras de la población, los cuales e aprecian en la siguiente grafica:



En este caso, se pudo notar que, a pesar de que el mejor individuo de la población mantuvo constante su valor fitness en las primeras iteraciones, el valor promedio del fitness de la población sufrió un rápido crecimiento en estas primeras iteraciones. Este fenómeno se debió al proceso evolutivo en general, que estaba descartando los individuos de menor fitness gracias a los operadores genéticos de reproducción, que permitieron que los individuos con mejor código genético de dupliquen y al operador de crossover, que combinó el código genético de los mejores individuos. Si en promedio, estos individuos creados por el operador genético del crossover, no presentaron un mejor código genético que los individuos desplazados esta curva, la curva de esta grafica no mostraría una pendiente creciente, sino se hubiera mantenido constante o incluso decreciente. Esto dio una idea de la efectividad del operador de crossover ya demostrado por el teorema del Schemadata en los apéndices de esta tesis.

Otro detalle interesante fue, que una vez alcanzada la solución óptima en la iteración 24, se siguió observando una disminución de la función fitness, este período fue justamente, el periodo en que el mejor individuo se mantuvo y empezó a reproducir, su código genético en los demás individuos, por medio de los operadores genéticos de crossover y reproducción. Esto nos indicó que el algoritmo Continuó buscando una mejor solución, sin darse cuenta que ya había encontrado la solución óptima.

En esta gráfica también se notó otro detalle interesante: se observó pequeños saltos en el valor promedio del fitness, después de pasar la iteración 50, donde se asume que la solución óptima ya se encontraba estabilizada, ¿a qué se debió esto?. Pues, se debió al operador genético de la mutación, si bien, los otros dos operadores tienen la tendencia de generar individuos con igual o mejor valor fitness que sus padres, el operador de la mutación tiene la tendencia a generar individuos de menor fitness, debido fundamentalmente, al carácter estocástico de este operador. El código genético mutado, fue responsable de que ocurrieran pequeñas disminuciones en el valor promedio del fitness de la población. Estas disminuciones duraron poco, ya que para la siguiente generación, fueron descartados los individuos mutados que presentaron un valor fitness bajo.

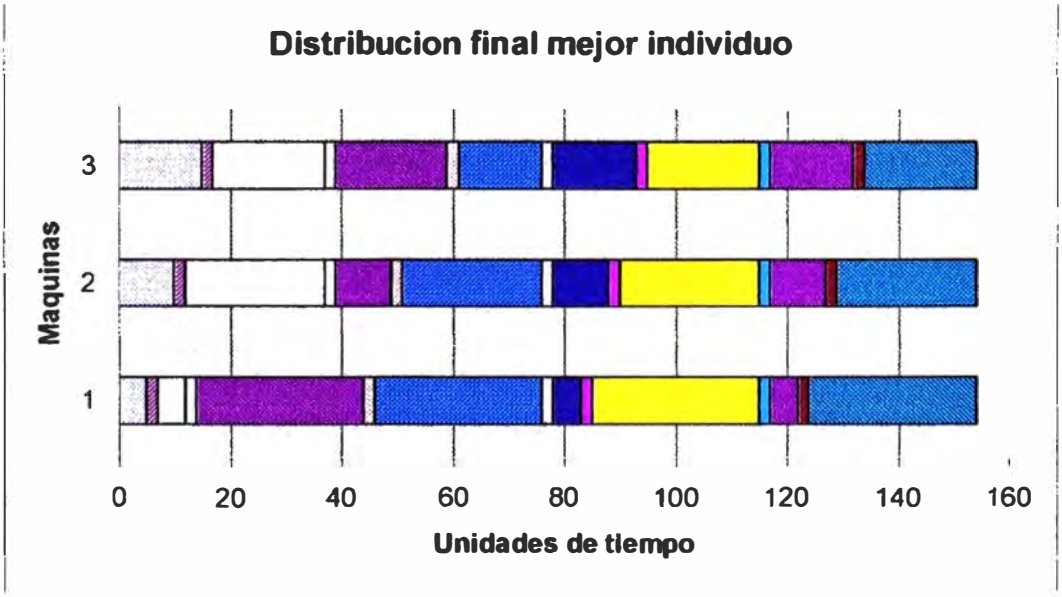


4.2.2 Resolución Caso 2

Ahora se probará la efectividad del método de algoritmos genéticos con el caso 2. Para este fin se usaron los mismo parámetros que en el caso 1.

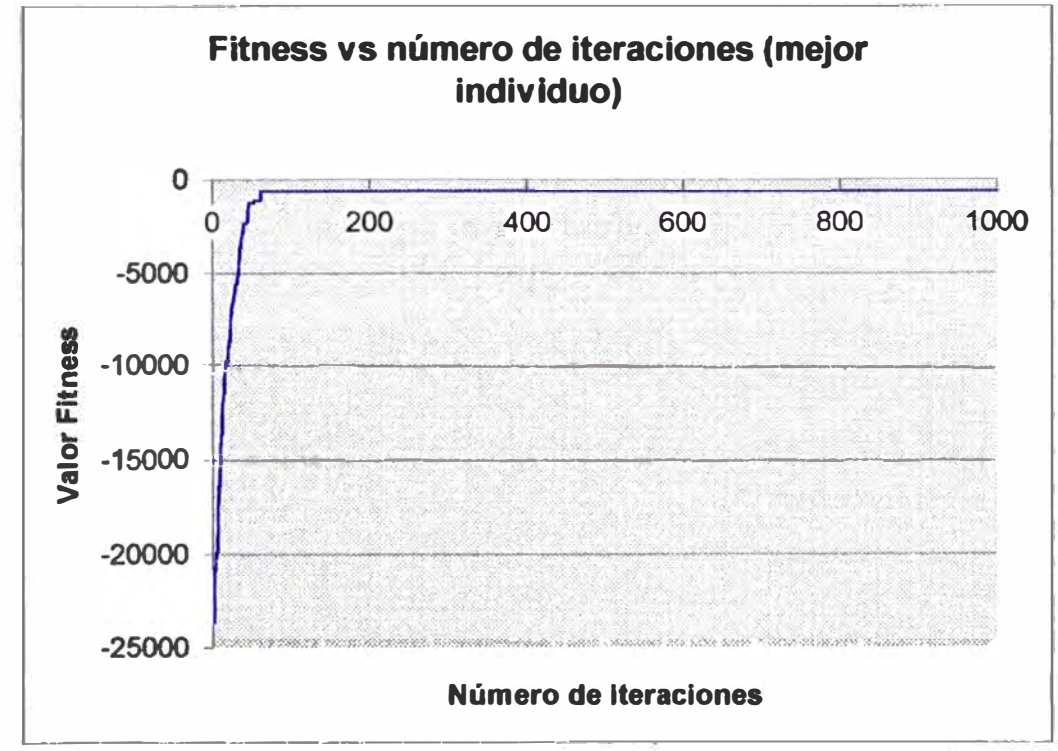
Población	200
Iteraciones	1000
Probabilidades	
Mutación	1%
Reproducción	10%
Crossover	89%

Ejecutando la simulación respectiva se llegó a la iteración 1000, sin que el método pudiera encontrar la solución óptima. El mejor individuo hallado hasta ese momento, aun mantenía una demora de 25 unidades en el tiempo de entrega:

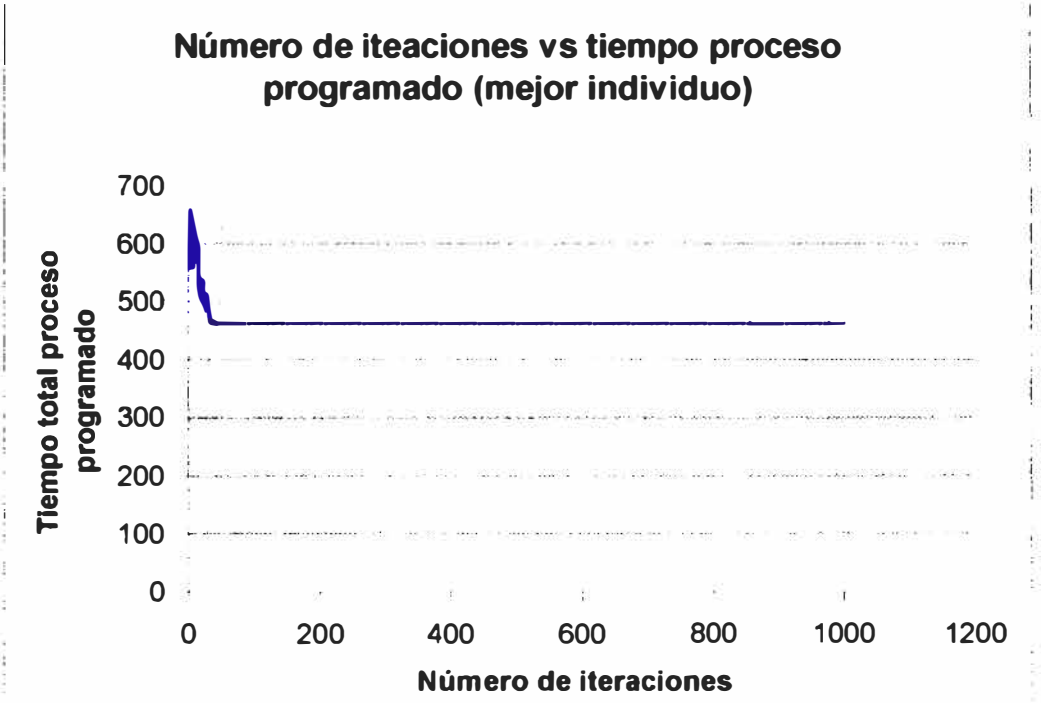


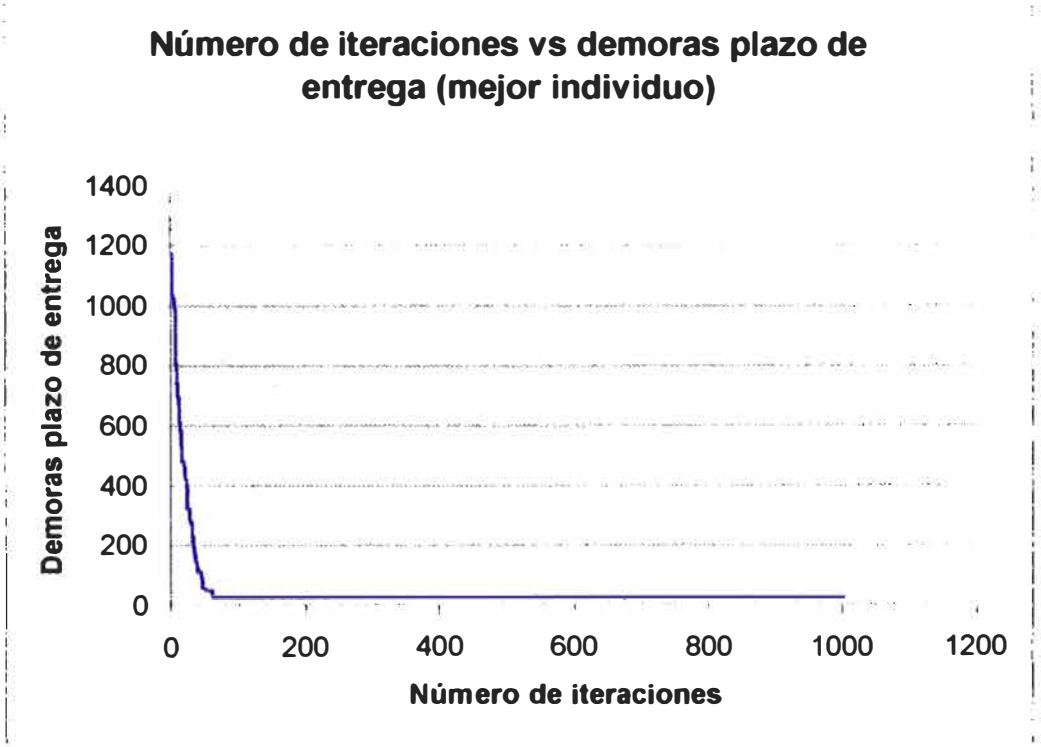
Se analizó este caso, descubriendo que la solución que se halló estaba muy cerca de la solución óptima, apenas hubiera sido necesario permutar un par de tareas en la máquina 1, y otro par en la máquina 3, sin embargo, el método no pudo encontrar esta solución.

La información que se analizó primero fue el valor fitness del mejor individuo. Al principio se presentó un comportamiento similar al estudiado en el caso 1, con mejoras en forma de pequeños saltos en el valor del fitness. Las mayores mejoras de la función fitness se dieron en las primeras 50 iteraciones, sin embargo, cuando llegó a la iteración 61 encuentro un individuo con valor fitness de –654, valor se mantuvo durante el resto de la simulación.

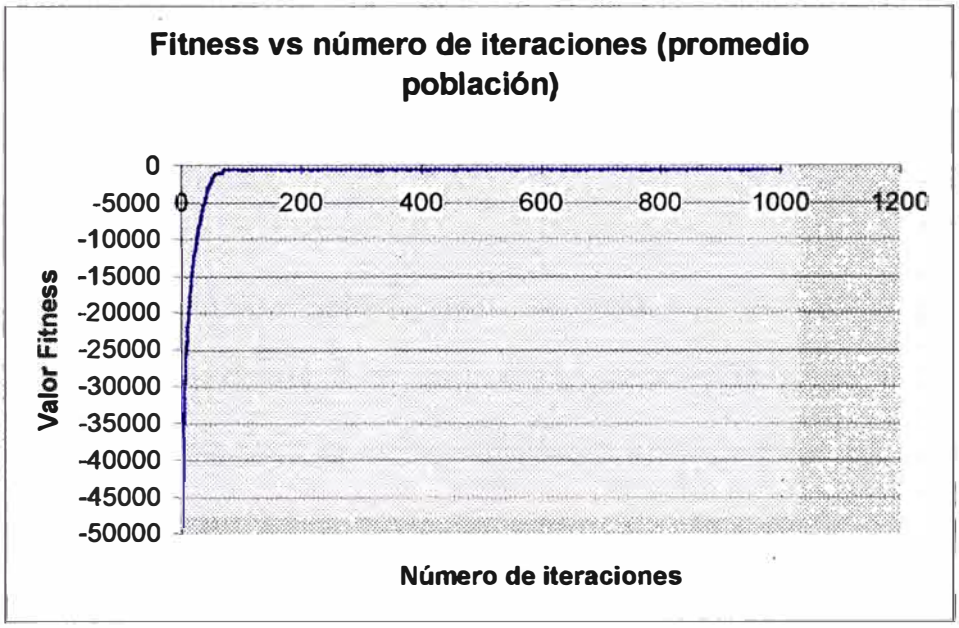


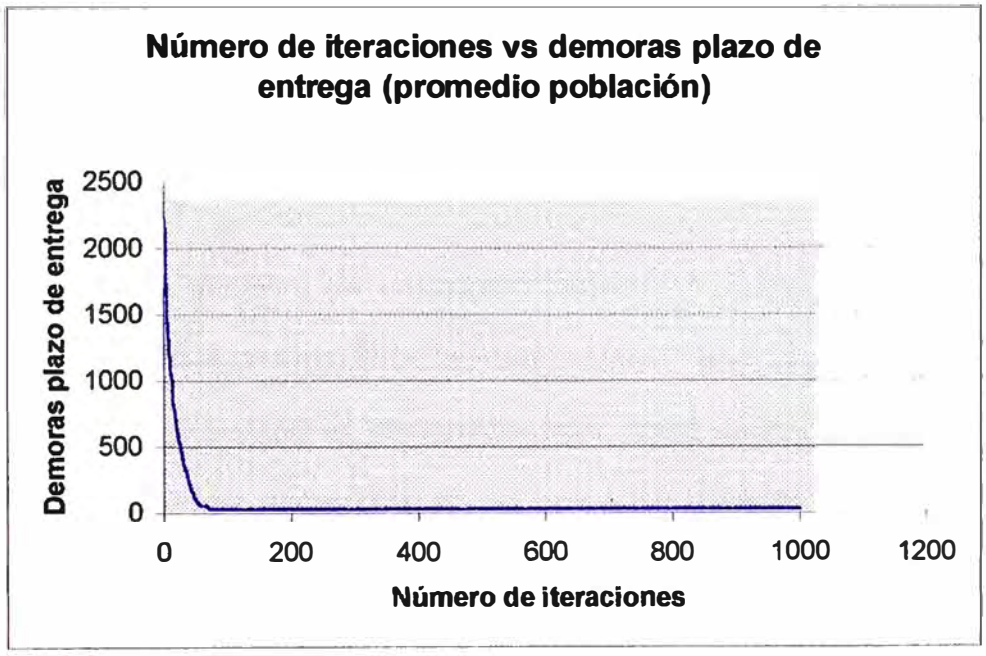
Un comportamiento similar se presentó en la gráfica de tiempo total programado vs el número de iteraciones y el de demoras en el plazo de entrega vs el número de iteraciones.





Entonces ¿qué factores intervinieron antes de la iteración 61, que pudieron generar individuos de mejor fitness en cada iteración, pero que después de la iteración 61 fueron incapaces de seguir generando individuos con mejor valor fitness que –654?. Para responder esta pregunta se analizaron los resultados de la simulación, pero esta vez, evaluando la población total:





En la gráfica de la población total, se pudo apreciar, que el valor de la función fitness promedio de la población aumentó visiblemente durante las 50 primeras iteraciones, al igual que el valor de la función fitness del mejor individuo. Sin embargo, esta tendencia, se mantuvo más tiempo en el valor promedio de la población, que en el mejor individuo de la población; llegando incluso a sobrepasar la iteración 61. Para ser exactos, esta tendencia se mantuvo, en el valor promedio del fitness de la población hasta la iteración 71, es decir, 10 iteraciones mas que la del mejor individuo, después del cual el valor promedio de la función fitness de la población se mantuvo oscilante con un valor promedio de –700.

Estos datos fueron interesantes, se notó que la función fitness se incremento rápidamente en las primeras iteraciones, luego de las cuales se mantuvo fija para el mejor individuo a partir iteración 61, pero siguió incrementándose en la población promedio, hasta aproximadamente la iteración 71, donde también se estabilizo con ciertas fluctuaciones. El factor que estuvo influenciando el valor del fitness promedio de la población hasta la iteración 71, y después ya no lo hicieron más era el operador genético de crossover.

El operador genético de crossover combina el código genético de dos individuos, este proceso tiene la tendencia de mantener el código genético bueno. Pero si todo el código genético fuera igual o muy similar las recombinaciones generadas por el operador genético de crossover no generarían individuos nuevos, sino generarán individuos ya existentes en la población. Entonces las primeras 70 iteraciones fue el tiempo necesario

para que el operador de crossover recombinara todo el código genético de la población, hasta uniformizarlo totalmente, en esta situación, ya no se produjo incrementos substanciales en el valor fitness promedio de la población, ni se generó un nuevo individuo con mejor valor fitness.

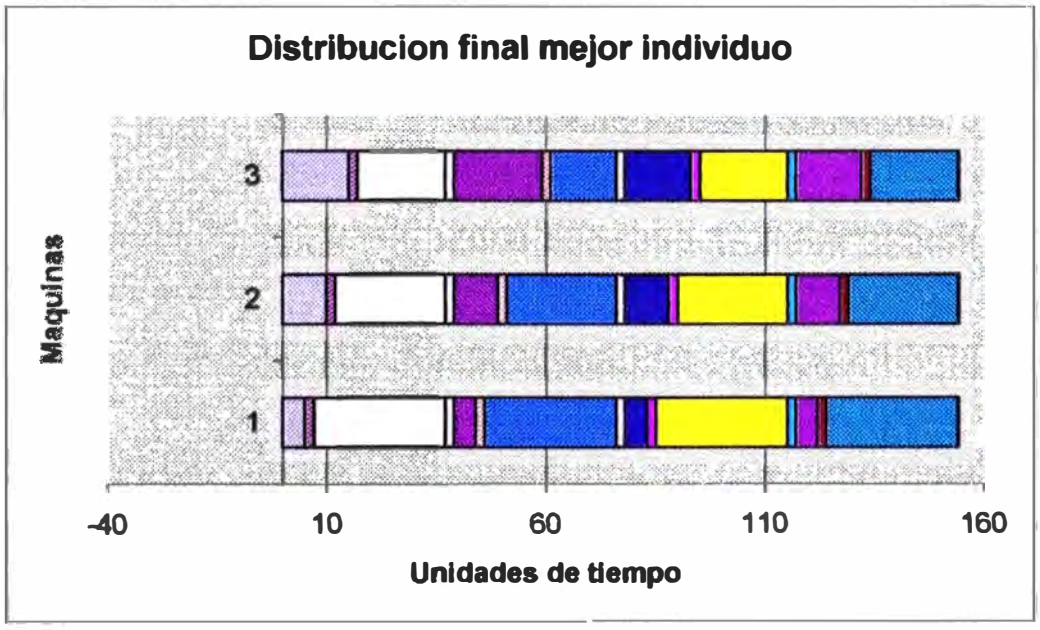
Sin embargo, las fluctuaciones en el valor fitness promedio se produjeron por el operador de la mutación. La mutación en cada iteración generó individuos aleatorios, que en nuestro caso, tuvieron un valor fitness inferior, al valor fitness de nuestro mejor individuo. Que tan inferiores puedan ser los valores fitness, dependió de los valores aleatorios que son asignados a sus genes, lo cual se vio reflejado como una oscilación en el valor del fitness promedio.

Entonces el problema de falta de convergencia a la solución óptima se debió a que todos los individuos en una determinada iteración presentaron código genético similar, entrampándose en un óptimo local. El hecho de que todos los individuos de una población presentaran códigos genéticos similares se llama "falta de diversidad genética", y es precisamente la causa por el cual no se pudo hallar la solución optima. Una de las formas de generar esta diversidad es el operador genético de la mutación. En este caso el operador no fue capaz de generar esta diversidad en más de 900 iteraciones restantes, simplemente, generó código genético con valor fitness inferior, que fue rechazado por el proceso evolutivo. Otra forma es incrementando la población inicial, como se genera aleatoriamente esto aseguraría una base genética superior a la que teníamos con una población menor.

De esta forma, se propuso una variación a los valores iniciales planteados para el problema, incrementando la población:

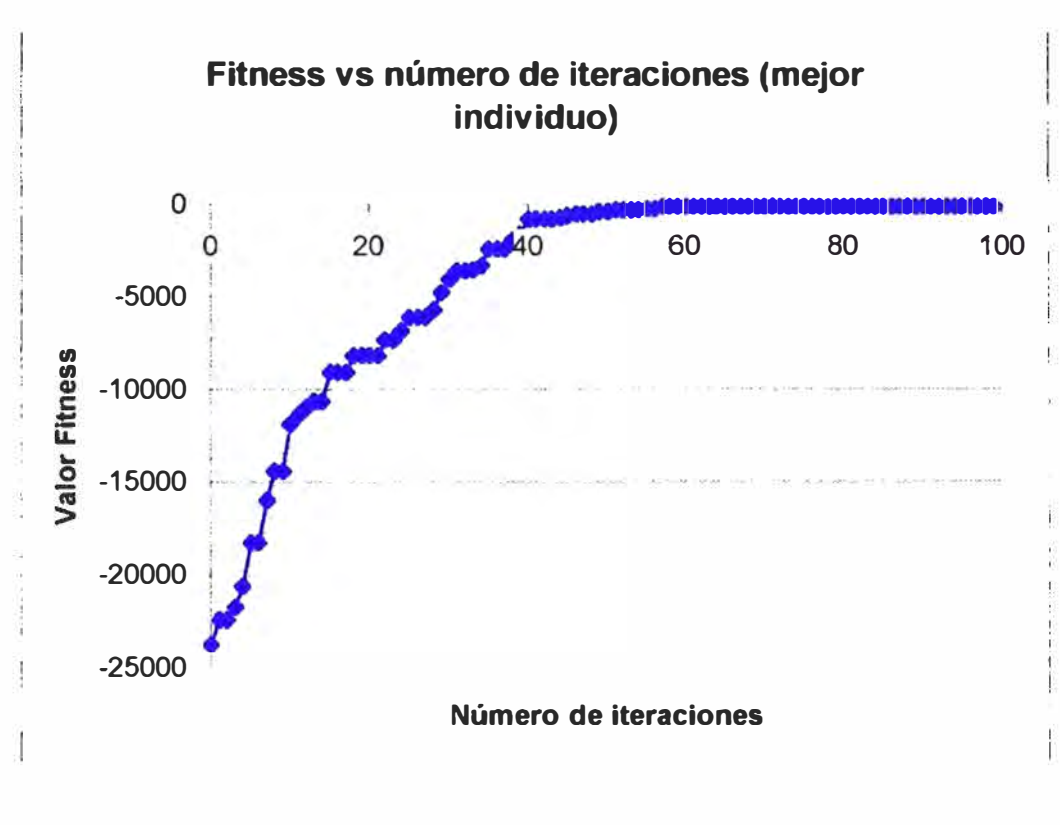
Población	400
Iteraciones	1000
Probabilidades	
Mutación	1%
Reproducción	10%
Crossover	89%

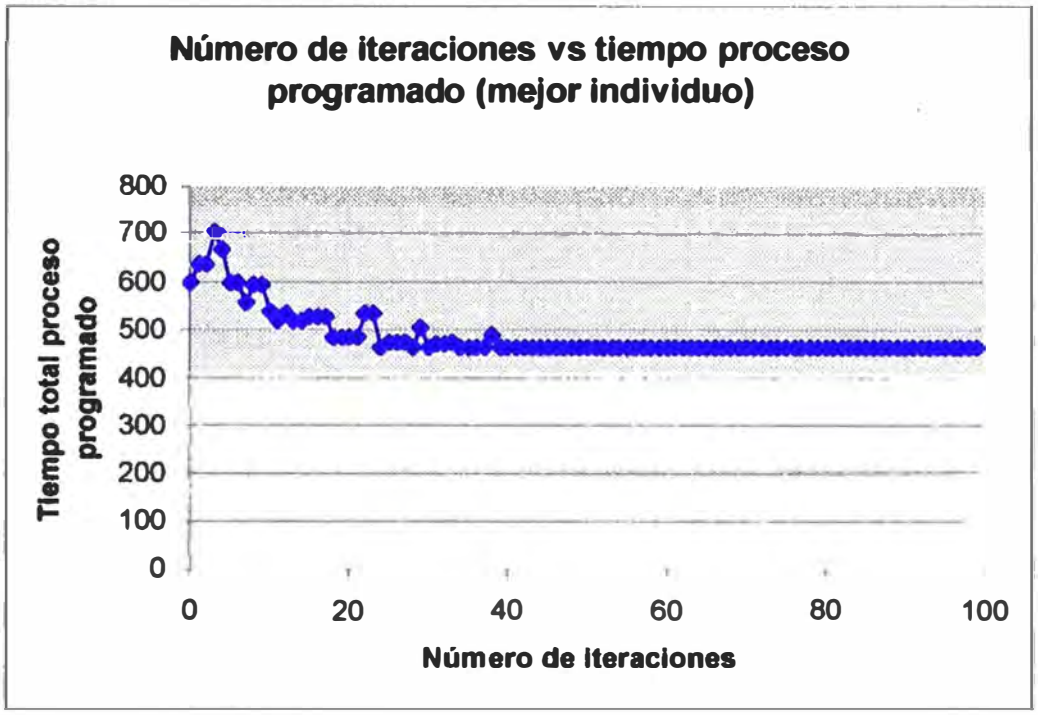
La simulación produjo como resultado que se encontrara la solución óptima en la iteración 57. La solución se presenta a continuación:

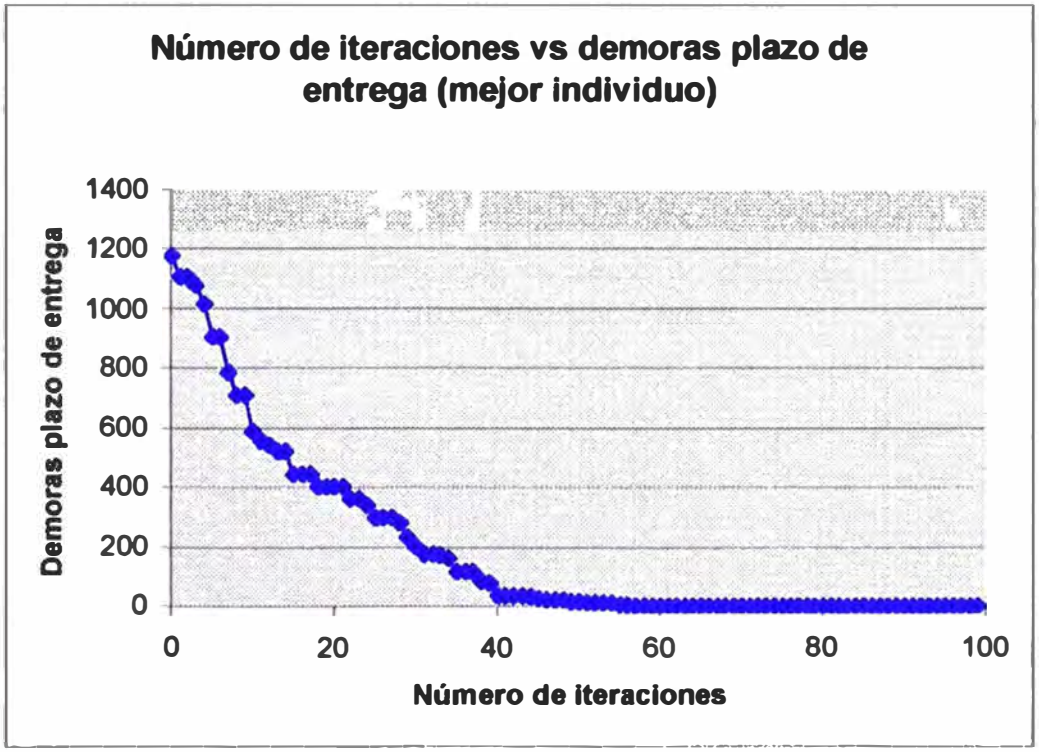


Se comprobó que era la solución óptima planteada del problema. Analizando las graficas de valor de la función fitness del mejor individuo vs

iteraciones, tiempo de proceso programado vs iteraciones y demoras vs iteraciones, se observó que esta vez el algoritmo presentaba un comportamiento similar al anterior con la diferencia que esta vez el método si fue capaz de llegar a la solución óptima.

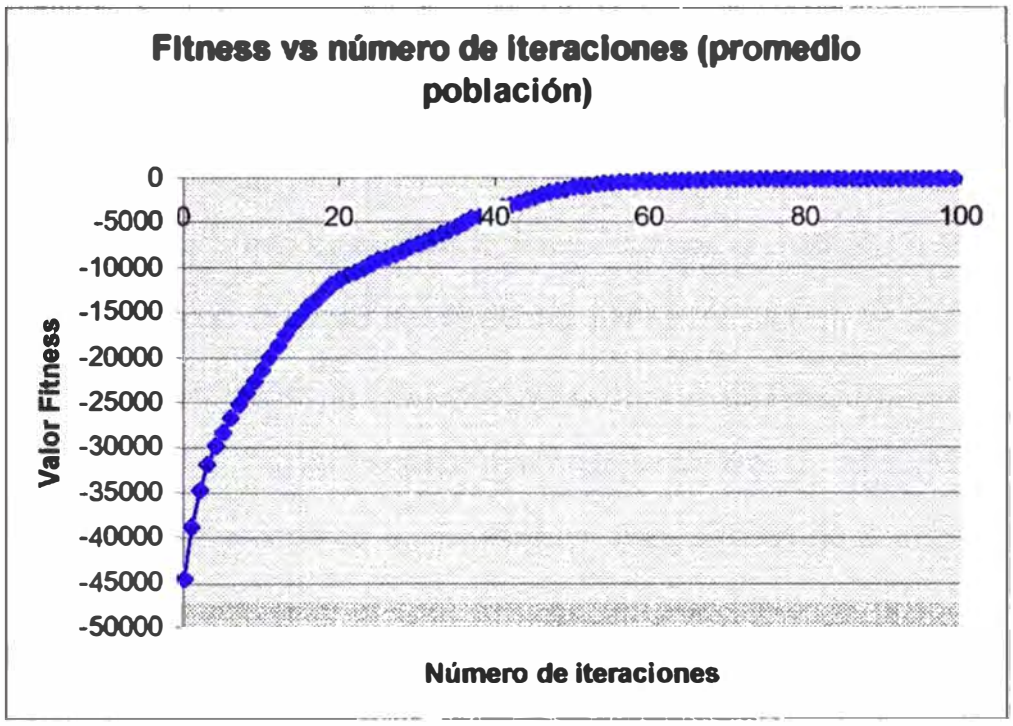


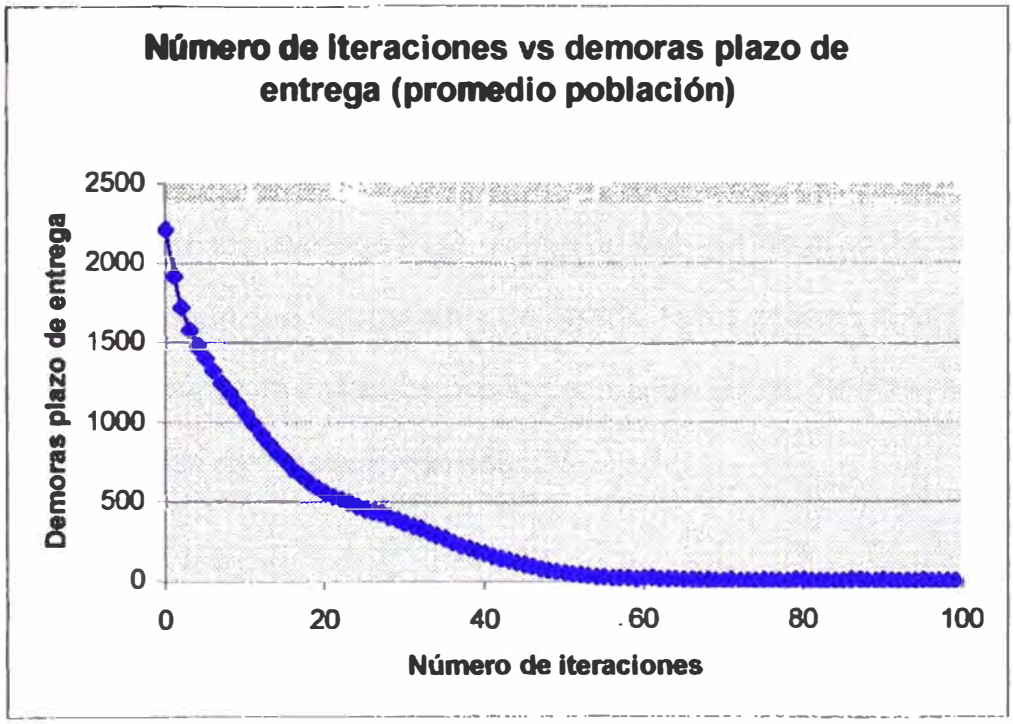




Se examinaron las gráficas para los valores poblacionales promedios; observando una pendiente pronunciada en las primeras iteraciones, para

luego presentar una pendiente más suave, hasta uniformizar los valores de la población, con las acostumbradas fluctuaciones causadas por el operador genético de la población.





Hasta ahora se demostró la eficiencia del método, para encontrar la solución, con un único reajuste en los parámetros iniciales, duplicando la población.

4.3 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES

En el siguiente punto se ejecutó la simulación con el método de sistemas multiagentes con sistemas clasificadores para la resolución de caso 1 y el caso 2. A diferencia de la simulación en el método de algoritmos genéticos anterior, el cual no se necesitó entrenamiento previo, este método si requiere de un periodo de entrenamiento. Este periodo de entrenamiento es el proceso mediante el cual, los agentes adquirieron el conocimiento necesario para resolver el problema, por medio de pruebas de ensayo error.

4.3.1 Resolución Caso 1

Para este caso los investigadores usaron un sistema inicial conformado por 12 agentes, cada uno represento a un pedido. Estos agentes fueron dotados de reglas de comportamiento generadas al azar basadas en las combinaciones de los siguientes estados y reglas:

Nº	Estado	Descripción
1	Cumple con el tiempo de entrega y no existe superposición	El agente percibe que esta cumpliendo sus restricciones
2	Existe superposición	El agente percibe que otro pedido se ejecuta en el mismo rango de tiempo en la misma máquina
3	No cumple tiempo de entrega	El agente percibe que no esta cumpliendo con su plazo de entrega.
4	No cumple con tiempo de entrega y existe superposición	El agente percibe que otro pedido se ejecuta en el mismo rango de tiempo, en la misma máquina y que no esta cumpliendo con su plazo de entrega

Nº	Acción	Descripción
0	Búsqueda izquierda	El agente realiza una búsqueda en un tiempo de inicio anterior al actual en la misma máquina, en caso de no encontrarlo no realiza desplazamiento
1	Búsqueda derecha	El agente realiza una búsqueda en un tiempo de inicio posterior al actual en la misma máquina, en caso de no encontrarlo no realiza desplazamiento.
2	Moverse a cualquier máquina	Realiza un salto aleatorio a cualquier máquina
3	Moverse a máquina de mejor desempeño	Busca un desplazamiento en una máquina de mejor desempeño y ubica su inicio en la posición más tarde posible para cumplir con su plazo de entrega.
4	Superponerse con el agente anterior más	Superpone el inicio del agente con el inicio del agente anterior más próximo, en caso de no

	próximo	encontrarlo no realiza desplazamiento.
5	No hacer nada	No desempeña ninguna acción
6	Búsqueda izquierda en todas las máquinas	El agente realiza una búsqueda en un tiempo de inicio anterior al actual en todas las máquinas, en caso de no encontrarlo no realiza desplazamiento
7	Búsqueda derecha en todas las máquinas	El agente realiza una búsqueda en un tiempo de inicio posterior al actual en la misma máquina, en caso de no encontrarlo no realiza desplazamiento.
8	Superponerse con el agente posterior más próximo	Superpone el inicio de agente con el inicio del agente posterior más próximo, en caso de no encontrarla se ubica al inicio de operaciones de la máquina.
9	Superponerse con uno de los agentes anteriores	Superpone el inicio del agente con el inicio del un agente anterior seleccionado aleatoriamente, en caso de no encontrarla se ubica al inicio de operaciones de la máquina.

Cada simulación estaba conformada de la siguiente forma:

Conocimiento	
Número de reglas por agente	12
Reglas que se mantienen por sistemas clasificadores	10
Reglas generadas por sistemas clasificadores	2
Máximo número de reglas heredadas a la siguiente generación	12
Número máximo de reglas generadas durante simulación	Solo si no existe acción para el estado

Simulación	
Número máximo de movimientos por corridas	1000
Número de corridas (Generaciones)	Los necesarios hasta obtener el conocimiento para solucionar el problema

Los agentes ejecutaron simulaciones continuas donde el conocimiento adquirido en una generación fue heredado a la generación de la siguiente simulación. Estas simulaciones se generaron durante dos horas tiempo en el cual obtuvo la solución óptima en tres corridas consecutivas. Las reglas generadas por los agentes en esta ultima corrida fueron tomadas como conocimiento inicial para los agentes en las siguientes simulaciones de este experimento. Dicho conocimiento inicial fue el siguiente:

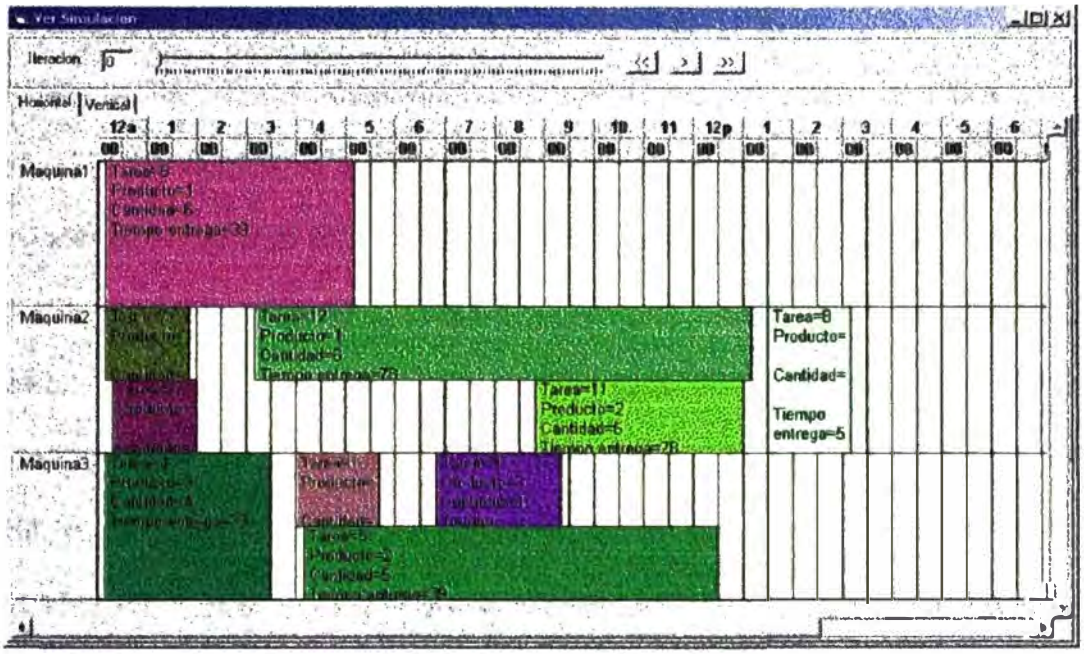
Conocimiento inicial obtenido en la fase de aprendizaje	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda izquierda
	No hacer nada
Existe superposición	Búsqueda izquierda
	Búsqueda derecha
	Moverse a máquina de mejor desempeño
No cumple tiempo de entrega	Búsqueda izquierda
	Moverse a máquina de mejor desempeño
	Superponerse con el agente anterior mas próximo
No cumple con tiempo de entrega y existe superposición	Búsqueda derecha
	Moverse a máquina de mejor desempeño

Las simulaciones mantuvieron el esquema de herencia del conocimiento de generación en generación. Realizando las simulaciones se obtuvo los siguientes resultados:

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	12	0	7
2	12	0	60
3	12	0	4
4	12	0	4
5	12	0	88
6	12	0	34
7	12	0	21
8	12	0	37
9	12	0	24
10	12	0	60

El número mínimo se iteraciones se encontró en la corrida 3 y 4 llegando a solamente 4 iteraciones. Se examino esta simulación con detenimiento, a fin de analizar el comportamiento de los agentes estaba de acuerdo con el conocimiento adquirido. Los autores aclararon que, el análisis siguió una secuencia ordenada por el orden de la tarea, pero en la simulación la secuencia de ejecución de los agentes fue aleatoria.

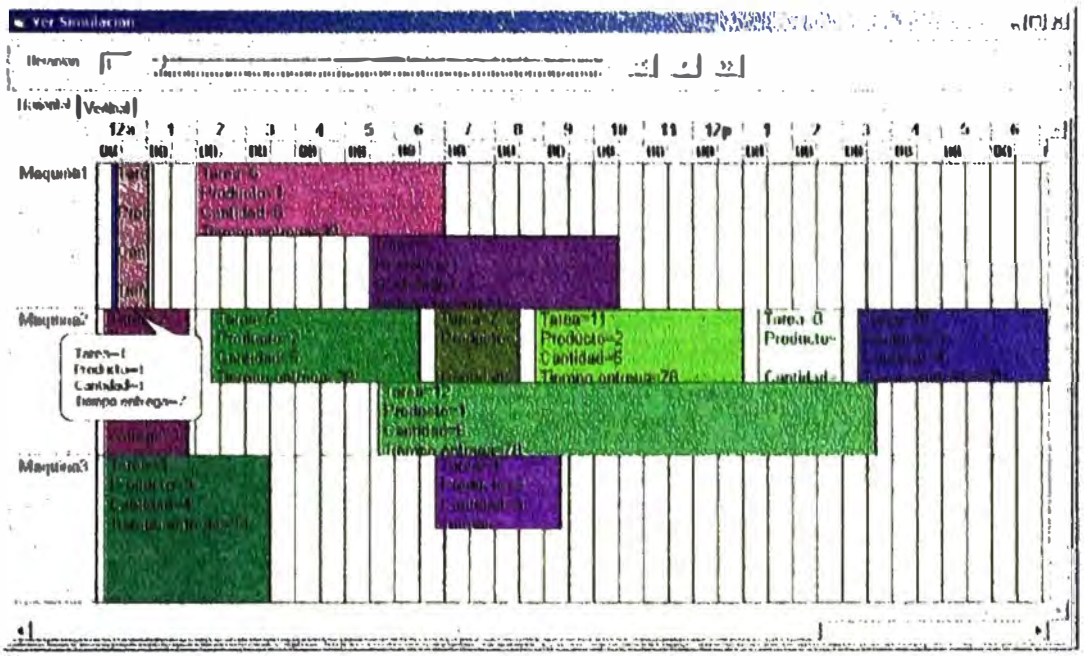
Usando el visor de simulaciones se observó que en la posición inicial se obtuvo la siguiente distribución aleatoria:



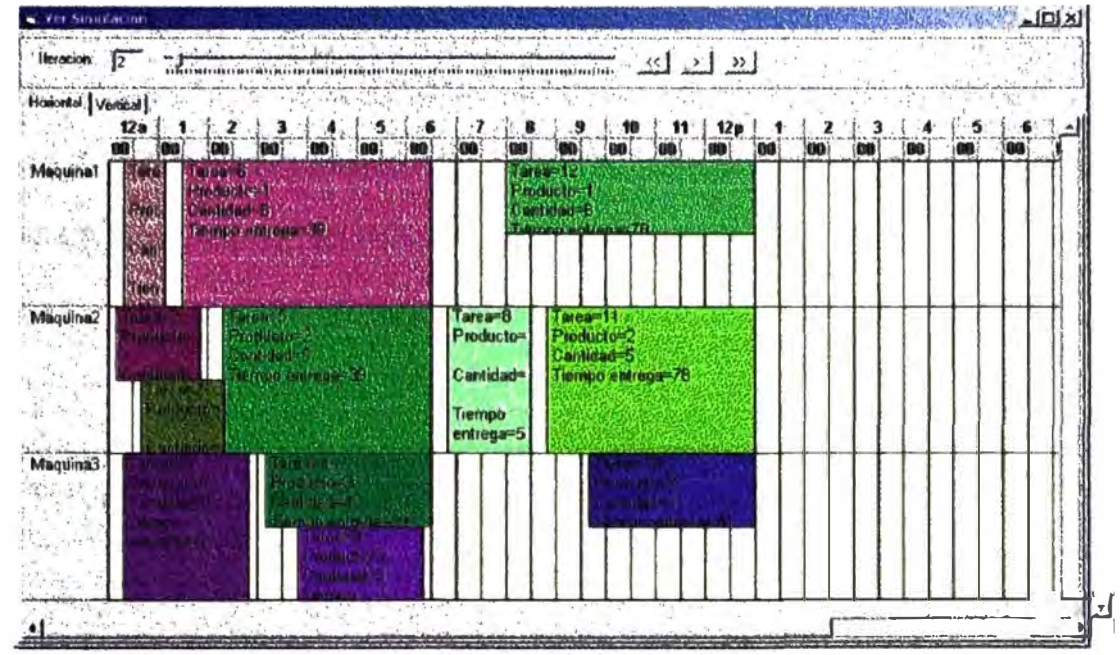
En este grafico se observó a todas las tareas con excepción de las tareas 3 y 10 debido a que su inicio escapaba del rango visual del visor. La tarea 3 se ubico en la máquina 1 y la tarea 10 en la máquina 2.

En la primera iteración se observó un comportamiento interesante el agente que representaba a la primera tarea respondió a su estado (no-cumplimiento del plazo de entrega) con la acción de desplazamiento a la máquina de mejor desempeño para el tipo de producto 1. Logrando de esta forma desplazarse a la primera máquina en una posición donde cumplía su plazo de entrega. El agente de la tarea 2 por su parte actuó optimizando su solución desplazándose a la izquierda en la misma máquina. El agente de la tarea 3, que no apareció en el grafico inicial, también se desplazo hacia la izquierda en su misma máquina tratado de esta forma cumplir con su plazo de entrega, objetivo que no pudo cumplir con esta acción. El agente de la tarea 4, sin embargo, cumplió con sus condiciones todas sus restricciones por lo cual ejecuto la acción de no hacer nada. El agente de

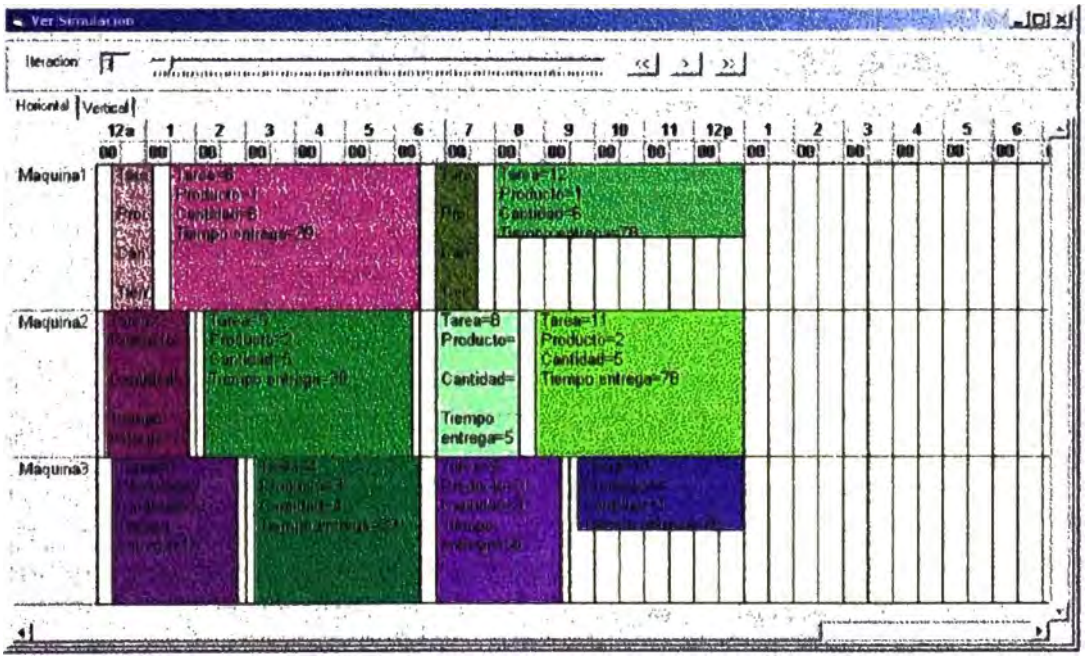
la tarea 5 que no cumplía con su plazo de entrega se desplazo a una máquina de mejor desempeño ocasionando de esta manera una superposición con la tarea 12. El agente de la tarea 6 ejecuto un desplazamiento a la derecha ocasionando una superposición con la tarea 3. El agente de la tarea 7 ejecuto un desplazamiento a la derecha para evitar la superposición con la tarea 2. Los agentes de las tareas 8, 11 ejecutaron la acción de desplazamiento a la izquierda pero debido a que no encontraron espacio libre se quedaron en sus posiciones iniciales. El agente de la tarea 9 por su parte ejecuto no hacer nada ya que cumplía con sus restricciones. El agente de la tarea 10 por su parte también ejecuto la acción de desplazamiento a la izquierda para ejecutarse después de la tarea 8. Por último el agente de la tarea 12 ejecuto un salto aleatorio a cualquier máquina que en esta caso coincidió con la máquina 2 provocando superposiciones con las tareas 8 y 10. En este último caso los autores aclararon, que si bien los agentes recibieron un conocimiento inicial, estos tenían la capacidad de modificar este conocimiento adicionando nuevas reglas. En este caso el desplazamiento aleatorio a cualquier máquina no era parte del conocimiento inicial sino fue generado y ejecutado por el agente de la tarea 12. El existo o fracaso de esta innovación fue evaluado posteriormente por el mismo agente mediante los mecanismos de sistemas clasificadores.



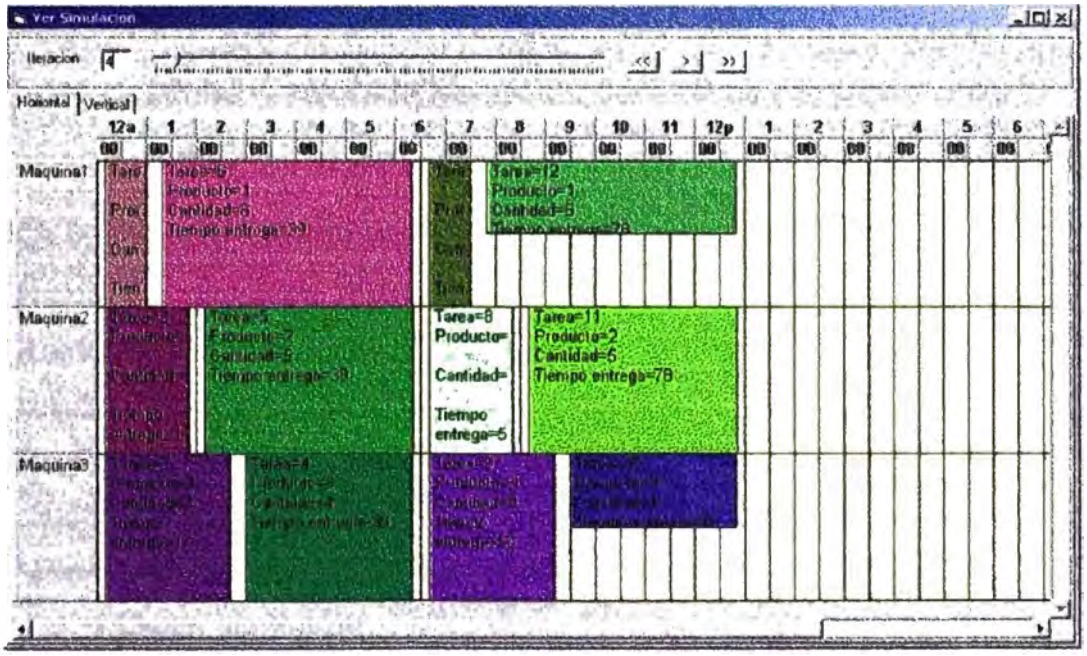
Se evaluó a continuación la iteración 2 en este caso se observó que los agente de las tareas 1 y 2 ejecutaron la acción de no hacer nada debido a que había satisfecho todas sus restricciones. El agente de la tarea 3 por su parte se desplazo a una máquina de mejor desempeño para su tipo de producto, en este caso, la máquina 3. El agente 4 que se encontraba en superposición con la tarea 3 se desplazo a la derecha. El agente de la tarea 5 ejecuto la acción no hacer nada. El agente de la tarea 6 optimizó su solución desplazándose a la izquierda. El agente de la tarea 7 se desplazo a cualquier máquina, en este caso su propia máquina. Los agentes de las tareas 8, 9 y 11 se desplazaron a la izquierda. Y por último los agentes de las tareas 10 y 12 se desplazaron a una máquina de mejor desempeño en este caso la 3 y 1 respectivamente.



El la tercera iteración el agente de la tarea 7 se desplazó a la máquina 1 de mejor desempeño para su tipo de producto. El agente de la tarea 9 se desplazó hacia la derecha para eliminar la superposición. Y el agente de la tarea 5 optimizó la solución desplazándose a la izquierda. En este punto se cumplieron todas las restricciones, encontrándose una solución, quedando únicamente el proceso de optimización, para llegar a la solución óptima.



En la iteración 4 los autores comprobaron que los agentes ejecutaron la acción de desplazamiento a la izquierda, optimizando de esta forma la solución encontrada en la iteración 3 obteniendo la solución óptima en solo 4 iteraciones.



Los autores aclararon que los agentes no fueron forzados ni incentivados de ninguna forma en la iteración 4, para optimizar la solución encontrada en la iteración 3. Si lo agentes actuaron de esta forma, se debió únicamente a que los mismo agentes, evaluando el conocimiento que poseían en ese momento, escogieron esa acción como la mas apropiada.

4.3.2 Resolución Caso 2

Debido a que el caso 2 guarda similitud al caso 1, diferenciándose en el aumento de la complejidad del problema, por el incremento de pedidos a

ser programados, los autores decidieron reutilizar el conocimiento adquirido en el caso 1, en las simulaciones en el caso 2. De esta manera los autores no solo determinarían la eficacia del método para resolver casos mas complejos, sino que, medirían la adaptabilidad de los agentes para resolver variaciones en el problema, usando el conocimiento adquirido en problemas menos complejos.

La simulación se realizo bajo los mismos parámetros que para el caso 1.

Conocimiento	
Número de reglas por agente	12
Reglas que se mantienen por sistemas clasificadores	10
Reglas generadas por sistemas clasificadores	2
Máximo número de reglas heredadas a la siguiente generación	12
Número máximo de reglas generadas durante simulación	Solo si no existe acción para el estado
Simulación	
Número máximo de movimientos por corridas	1000
Número de corridas (Generaciones)	Los necesarios hasta obtener el conocimiento para solucionar el problema

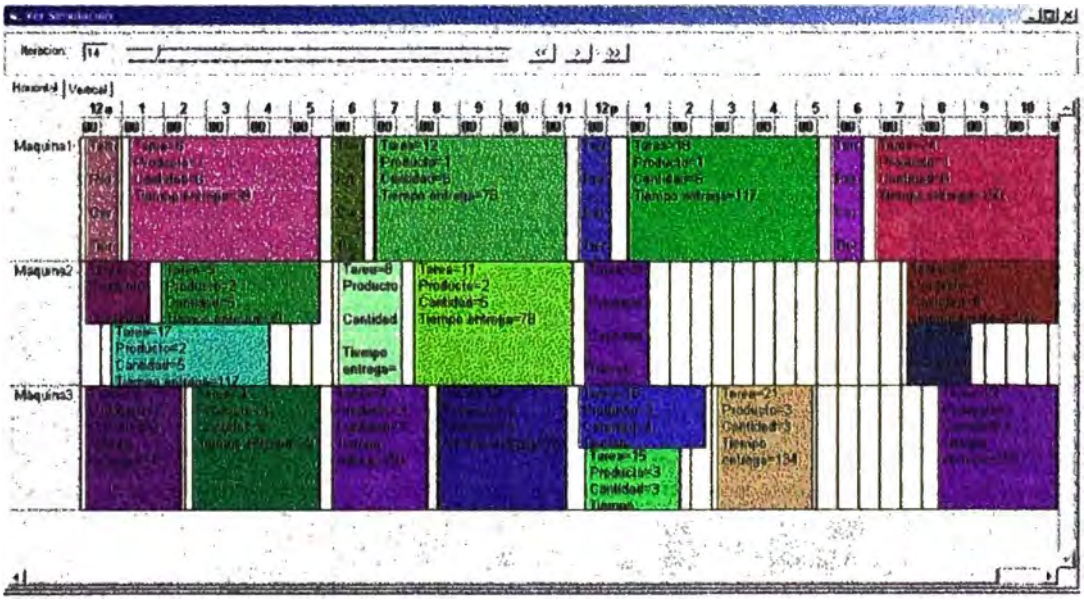
Conocimiento inicial heredado del caso 1	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda izquierda
	No hacer nada
Existe superposición	Búsqueda izquierda
	Búsqueda derecha
	Moverse a máquina de mejor desempeño
No cumple tiempo de entrega	Búsqueda izquierda
	Moverse a máquina de mejor desempeño
	Superponerse con el agente anterior mas próximo
No cumple con tiempo de entrega y existe superposición	Búsqueda derecha
	Moverse a máquina de mejor desempeño

La solución obtenida durante las simulaciones se muestra a continuación, en ella se observó, que si bien se produjo un aumento en el número de iteraciones el conocimiento heredado del caso 1, fue suficiente para garantizar la obtención de la solución óptima, llegando a un número mínimo de iteraciones de 18 para encontrar esta solución.

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	24	0	399
2	24	0	34
3	24	0	18
4	24	0	82
5	24	0	301
6	24	0	138
7	24	0	77

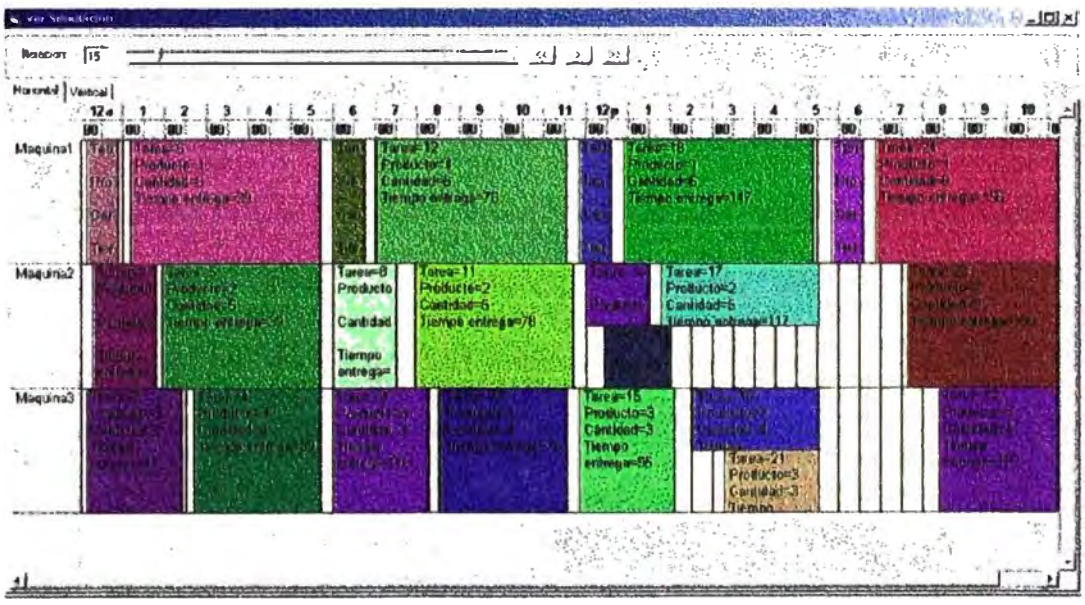
8	24	0	68
9	24	0	144
10	24	0	112

Los autores examinaron las 5 ultimas iteraciones de la solución de 18, encontrando situaciones interesantes. En la iteración 14 se observó que la solución para la máquina 1 estaba casi completa, faltando únicamente el proceso de optimización. En la máquina 2 sin embargo existían 2 superposiciones una con las tareas 17, 2 y 5 y otra con las tareas 20 y 23, a pesar de esto, se observaba un gran espacio libre en medio del programa. En la máquina 3 existía una superposición con las tareas 15 y 16. A pesar de estas superposiciones, las 12 primeras tareas estabas muy cerca de la solución del problema como en el caso 1.



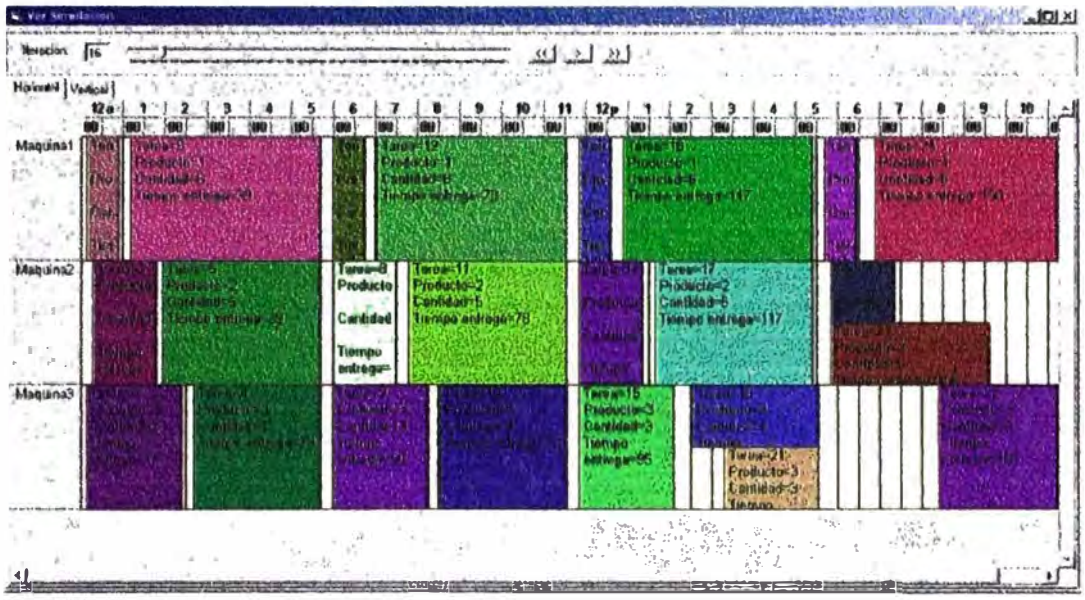
En la iteración 15 se observó, que el agente de la tarea 17 ejecuta un desplazamiento a la derecha al espacio libre mencionado anteriormente, eliminando de esta forma la superposición con las tareas 2 y 5. Cabe

destacar que a pesar de que los agentes de las tareas 2 o 5, pudieron ejecutar acciones que malograrían la solución, ambos no lo hicieron, dejando la acción al agente de la tarea 17. Asimismo el agente de la tarea 16 realizo una superposición con la tarea 21. El agente de la tarea 20 realizo un salto aleatorio dentro de la misma máquina, causando una superposición con las tareas 14 y 17, que hasta ese momento cumplían sus restricciones. En este punto se observó que la solución de las 12 primeras tareas se había encontrado, lo cual indico la forma de actuar de los agentes, para atacar problemas similares al problema en el cual habían sido entrenados: en este caso los agentes resolvieron primero el problema similar, atacando después la variación.

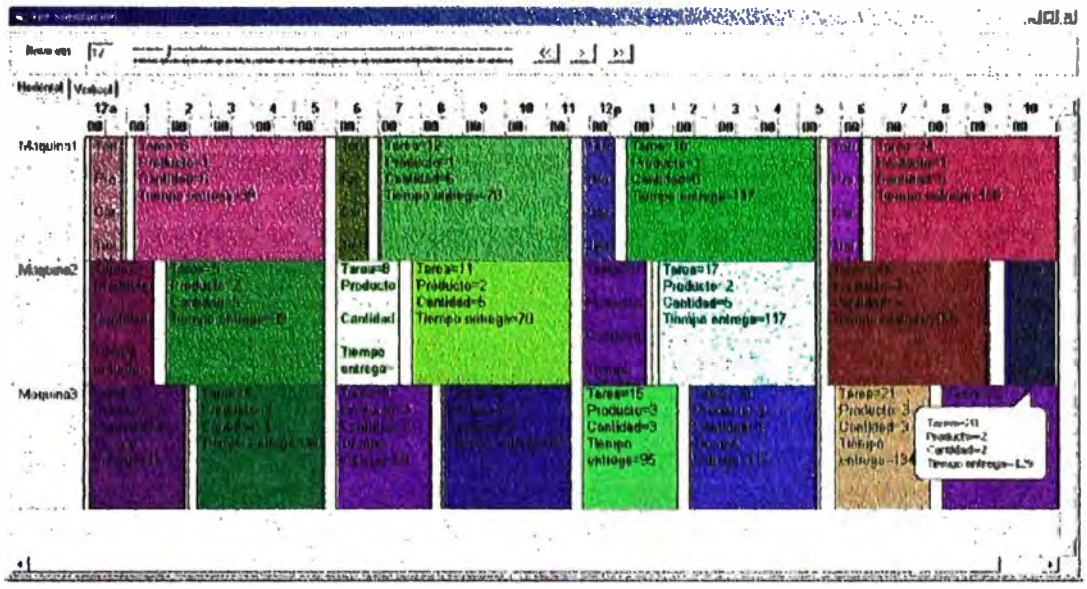


En la siguiente iteración, se observó que el agente de la tarea 20 se desplazó a la izquierda, liberando de esta forma a las tareas 14 y 17; un comportamiento muy similar al observado con el agente de la tarea 17, en la iteración 15. Por último el agente de la tarea 23 ejecuto una

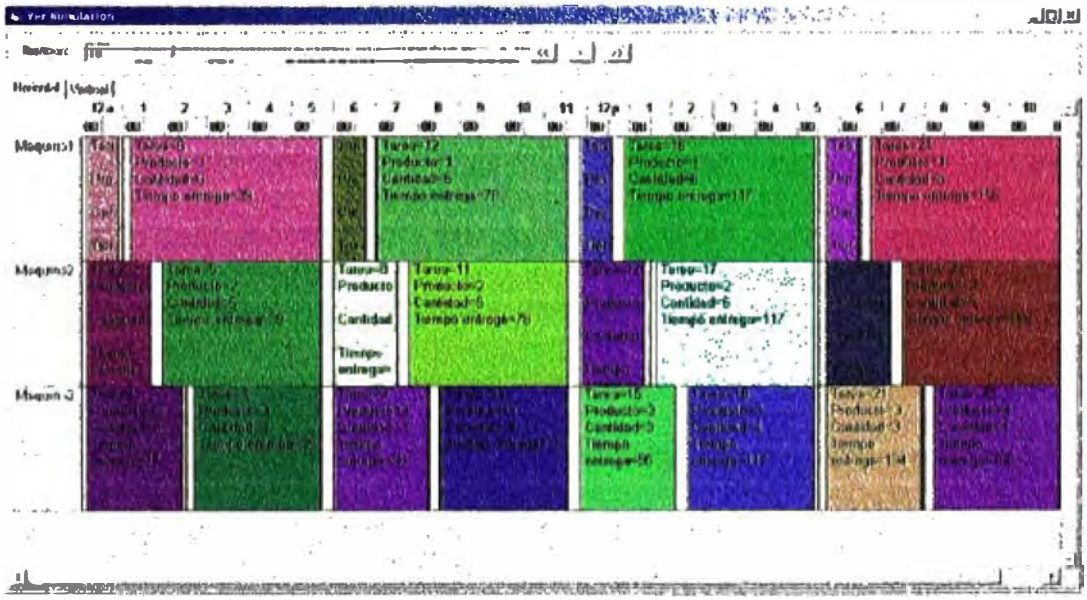
superposición con el la tarea 20, y algunos agentes realizaron optimizaciones menores.



En la iteración 17, el agente de la tarea 21 realizo un desplazamiento a la derecha, encontrando una solución para la máquina 3. El agente de la tarea 20 también realizo un desplazamiento a la derecha, lo cual elimino la superposición con la tarea 23, pero de esta forma incumplió su restricción de plazo de entrega . Cabe recalcar que una situación similar fue observaba en el método de algoritmos genéticos, en la resolución del caso 2 para una población de 200 agentes. En esa ocasión el método de algoritmos genéticos fue incapaz de resolver este problema sino hasta que su población aumento hasta 400 individuos.



El la iteración 18, se observó que el los agentes de las tareas 21 y 22 se desplazaron a la derecha, optimizando su solución. Asimismo el agente de la tarea 20 actuó generando una superposición con la tarea 23. Asimismo el agente de la tarea 23 respondió esta acción con un desplazamiento a la derecha, encontrando de esta forma la solución óptima.



Los autores recalcan que la operación de superposición solo demoro una iteración, esta misma solución debió ser aplicada por el método de algoritmos genéticos en la simulación de 200 individuos, pero en esa ocasión, el método de algoritmos no pudo resolver esta situación en mas de 900 iteraciones.

4.4 MÉTODO DE SISTEMAS MULTIAGENTES CON SISTEMAS CLASIFICADORES Y ESTRATEGIAS EVOLUTIVAS

Las principales dificultades en los sistemas multiagentes son: la determinación de la función de valoración, la forma de valoración de los agentes individualmente y la determinación de cuando es conveniente hacer las evaluaciones.

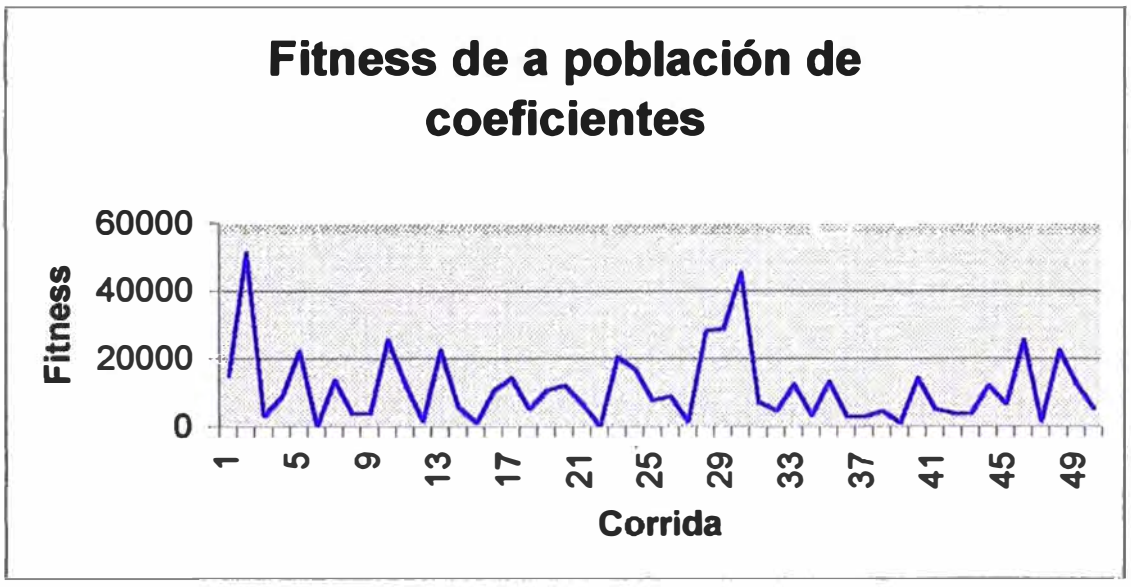
Una buena función de valoración, premiará y castigará adecuadamente las reglas de comportamiento de los agentes, permitiendo explotar eficazmente el conocimiento adecuado, así como explorar nuevas reglas de comportamiento. Esta función también influye en la eficiencia para resolver el problema. Por otro lado, una mala determinación de la función de valoración, puede hacer que los agentes nunca aprendan, y como resultado no resolverán el problema, por otro lado si a los agentes tienen un conocimiento adecuado, éste podría ser perdido.

Para abordar esta dificultad, se planteó este cuarto modelos, a fin de identificar los coeficientes adecuados para el modelo anterior, es decir el modelo de sistemas multiagentes con sistemas clasificadores. Las estrategias evolutivas fueron usadas para calcular los coeficientes que permiten al modelo aprender y resolver el problema con mayor eficiencia.

Debido a que la función de evaluación es la misma para todos los casos de estudio, solo el caso 1 es usado para la simulación. Los parámetros de las estrategias evolutivas fueron:

Población	4
Modelo de aprendizaje	$\mu+\lambda$
Padres	2
Iteraciones	25

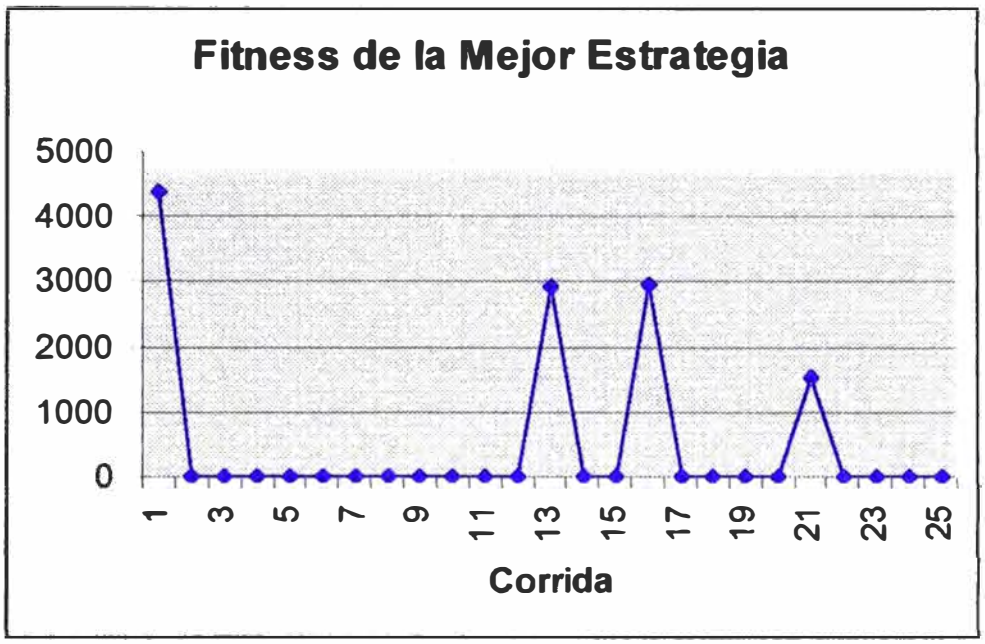
Inicialmente se intentó determinar los coeficientes adecuados para un aprendizaje más eficiente, donde el conocimiento es asignado en forma aleatoria, obteniéndose los siguientes resultados:



Como se puede apreciar, no se logró encontrar un conjunto de coeficientes que permiten al modelo aprender. Las valoraciones que se hacen a los agentes se propagan a través de todas las acciones realizadas que permitieron llegar hasta ese estado, premiándolos o castigándolos. En caso que un agente realizó una acción buena, pero otro agente influenció a que el sistema total tenga un resultado negativo, este primer agente es castigado, reduciendo los pesos de estas reglas, de esta forma el conocimiento adquirido puede ser perdido debido a la gran magnitud de los castigos.

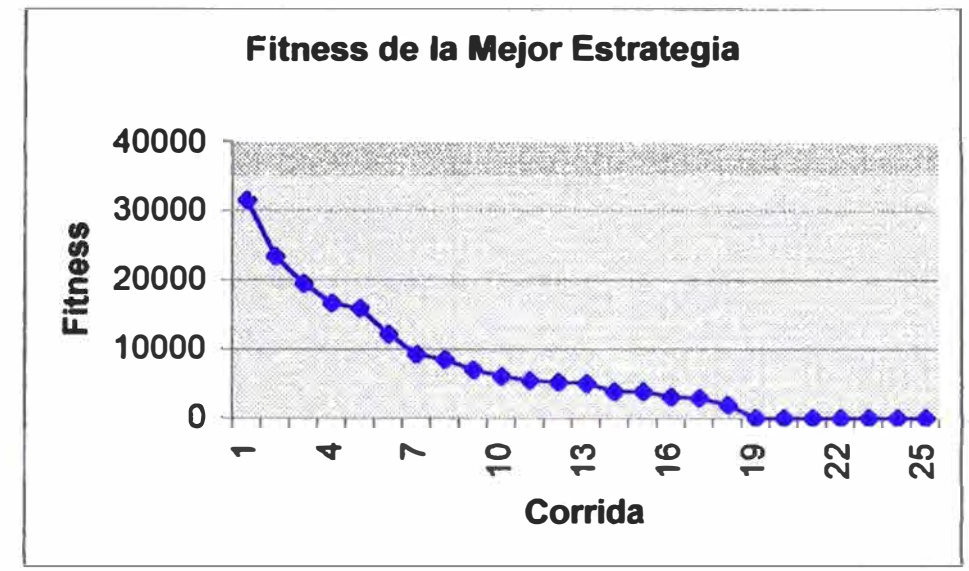
Por otro lado, en esta simulación el conocimiento inicial era nulo, es decir las reglas de comportamiento fueron generadas en forma aleatoria. Esto influenció negativamente en la simulación debido a que adicionaba más grados de libertad al sistema, complicando el estudio, ya que un resultado negativo podría ser resultado de una mala función de evaluación o un mal conocimiento inicial.

Para realizar una mejor evaluación del impacto de la función de evaluación, se tomó como conocimiento inicial las reglas obtenidas en el caso anterior, el cual será constante en el inicio de todas las corridas. En el siguiente gráfico se muestra los resultados obtenidos:



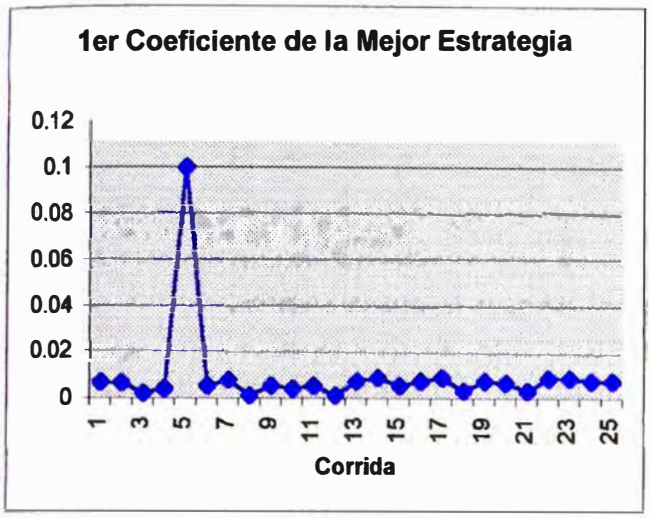
Al inicial la simulación, el sistema asigna en forma aleatoria valores a los coeficientes de la función de valoración. La mayoría de las veces, estos coeficientes iniciales hacen que los castigos sean muy fuertes y los agentes pierden el conocimiento necesario para poder resolver el problema. En esta simulación se observó un comportamiento interesante, debido a que el sistema cuenta con un conocimiento inicial eficiente, éste logra resolver el problema mucho antes de perder este conocimiento, es decir los coeficientes no jugaron el papel principal para resolver el problema. Razón el cual no se logró encontrar los coeficientes óptimos.

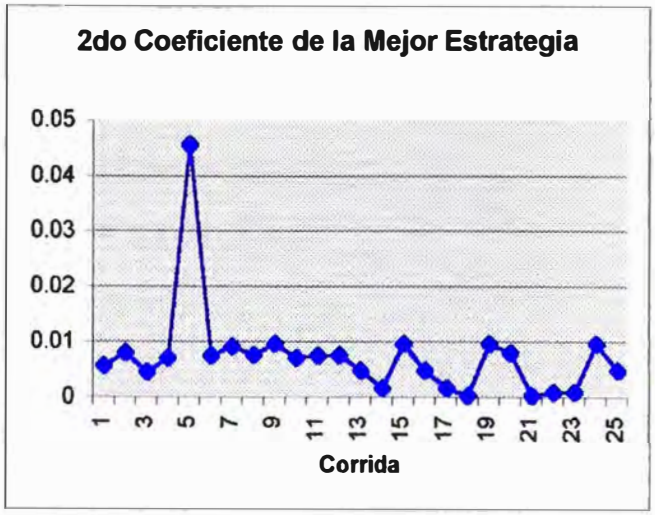
Estas simulaciones, llevaron a los autores a realizar una tercera simulación, en esta simulación se siguió tomando el conocimiento inicial anterior, pero se aplicó al problema de 24 pedidos, que es mucho más complicado, obteniendo los siguientes resultados:

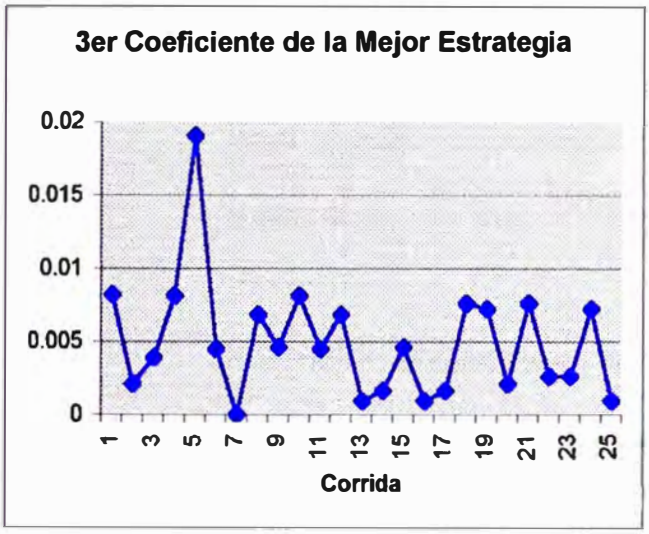


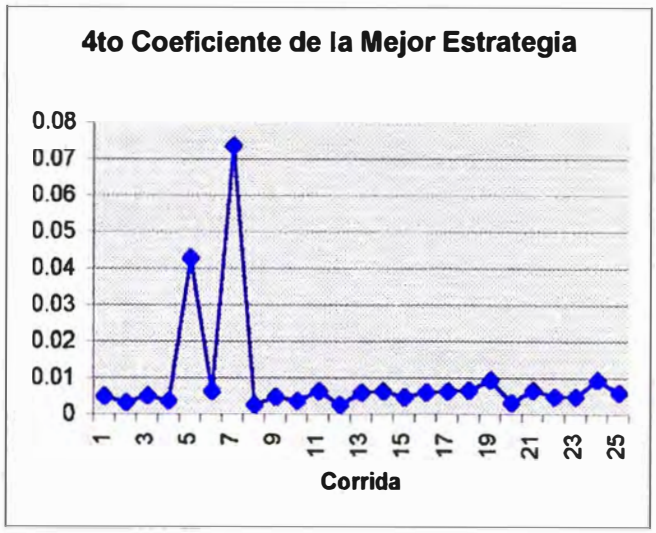
En esta simulación se pudo apreciar el efecto que tiene la función de evaluación, logrando encontrar los coeficientes mas adecuados para la resolución de problema planteado en esta tesis.

Los siguientes gráficos muestran las tendencias de los coeficientes de la función de valoración.









Al iniciar el programa, el sistema asigna a los coeficientes valores aleatorios, y conforme el sistema se va ejecutando, las estrategias evolutivas van encontrando un conjunto de coeficientes que le permiten resolver el problema más eficientemente.

El primer coeficiente, que representa la valoración de la mejora del tiempo total de ejecución, es el más importante, y concuerda con uno de los

objetivos del problema de asignación de recursos y tareas, que es la minimización del periodo total de ejecución.

El segundo coeficiente, que representa la influencia del tiempo muerto de las máquinas, tiene una gran importancia en el modelo, ya que esta variable obliga al sistema a optimizarse, y de esta forma reducir el tiempo total de ejecución.

De igual forma el tercer coeficiente, que castiga a las reglas por superposiciones, influencia en pequeña media la valoración total, esto se debe a que en muchos de los casos el sistema hace uso de la superposición para lograr encontrar la solución óptima.

Finalmente, el cuarto coeficiente muestra, que una reducción en la demora total influye positivamente a la valoración de las acciones del agente.

El conjunto de coeficientes que logró el mejor rendimiento fue:

Coeficiente 1	Coeficiente 2	Coeficiente 3	Coeficiente 4
0.0088	0.0076	0.0022	0.0012

Logrando resolver el problema en 25 iteraciones, estos coeficientes fueron utilizados, para las diferentes simulaciones de los modelos de sistemas multiagentes con sistemas clasificadores.

4.5 DETERMINACION DE LA EFICIENCIA DE LOS METODOS

En los puntos anteriores de este capitulo los autores evaluaron la eficacia de los métodos propuestos en esta tesis para hallar la solución de los casos de estudio planteados, encontraron que los métodos basados en algoritmos genéticos y sistemas multiagentes inteligentes presentaron un buen desempeño para resolver estos casos, ya que llegaron, con relativa facilidad, a encontrar la solución óptima propuesta para cada uno de los casos de estudio. Mientras la búsqueda exhaustiva representó una alternativa muy difícil o hasta imposible de realizar en la practica.

Sin embargo, los autores decidieron examinar la eficiencia de estos métodos, determinando de esta forma cual de los métodos resulta ser el más conveniente para resolver problemas de asignación de recursos y tareas. Decidieron realizar el planteamiento del tercer caso de estudio donde la complejidad del problema fue reducida al mínimo.

Caso 3:

Consideraron N pedidos de un único tipo de producto en cantidades y plazos de entrega iguales para todos los pedidos, los que serian ejecutados en N máquinas. Se asumió que la eficiencia de las máquinas eran iguales.

Como se puedo observar este caso es sumamente sencillo, la solución óptima para este problema solamente era designar que cada tarea se

ejecutara en una máquina diferente. Si bien la complejidad del problema fue mínima, este planteamiento permitió ir incrementando fácilmente el número de pedidos al problema, aumentando de esta manera, la complejidad de la búsqueda, obtuvieron de forma una herramienta de comparación entre los métodos propuestos.

Para realizar este análisis se usaron solamente los métodos de algoritmos genéticos y de sistemas multiagentes con sistemas clasificadores. No se evaluó el método de sistemas multiagentes con sistemas clasificadores y estrategias evolutivas debido a que este se trataba de una optimización del método de sistemas multiagentes con sistemas clasificadores; la estructura de aprendizaje de los agentes era la misma diferenciándose solamente en el uso de Estrategias evolutivas para la determinación de los coeficientes de la función de valoración. Además los autores consideraron que al tener un grado de complejidad mayor debido al proceso evolutivo para la determinación de los coeficientes, el método de sistemas multiagentes con sistemas clasificadores y estrategias evolutivas estaba en aparente desventaja contra el método de algoritmos genéticos.

Teniendo en cuenta estas consideraciones se evaluó la eficiencia de los métodos mencionados, resolviendo el caso 3. El objetivo de estas evaluaciones fue forzar a los métodos con la finalidad de encontrar la solución óptima variando las condiciones de los experimentos, haciendo un análisis de sensibilidad de los diversos factores que debieron considerarse para asegurar que el método encontrara la solución óptima.

4.5.1 Método de Algoritmos Genéticos, Resolución Caso 3

Para evaluar la eficacia del método de algoritmos genéticos los autores mantuvieron la estructura del genoma, la función fitness y las probabilidades de las operaciones genéticos usados en los experimentos anteriores:

Genoma:

Pedido1		Pedido2		Pedido3		...	Pedido(P-1)		PedidoP	
M1	S1	M2	S2	M3	S3	...	M(P-1)	S(P-1)	MP	SP

Función Fitness:

$$Fitness = \alpha\left(\sum_{i=1}^{M} UsoMaquina_i / M\right) + \beta\sum_{i=1}^{P} DemoraPedido_i + \delta MAX_{i=1}^{M}(UsoMaquina_i)$$

Constantes del algoritmo:

Probabilidades	
Mutación	1%
Reproducción	10%
Crossover	89%

Los factores que variaron fueron la población y el número máximo de iteraciones; ya que estos factores, en la metodología de algoritmos genéticos, ayudan a mejorar la calidad de la solución obtenida.

Consideraron un número de iteraciones lo suficientemente grande para abarcar una búsqueda promedio. Si una búsqueda demoraba más que este limite, simplemente se asumió que el algoritmo no pudo encontrar la solución. Se consideró 5,000 como el número máximo de iteraciones.

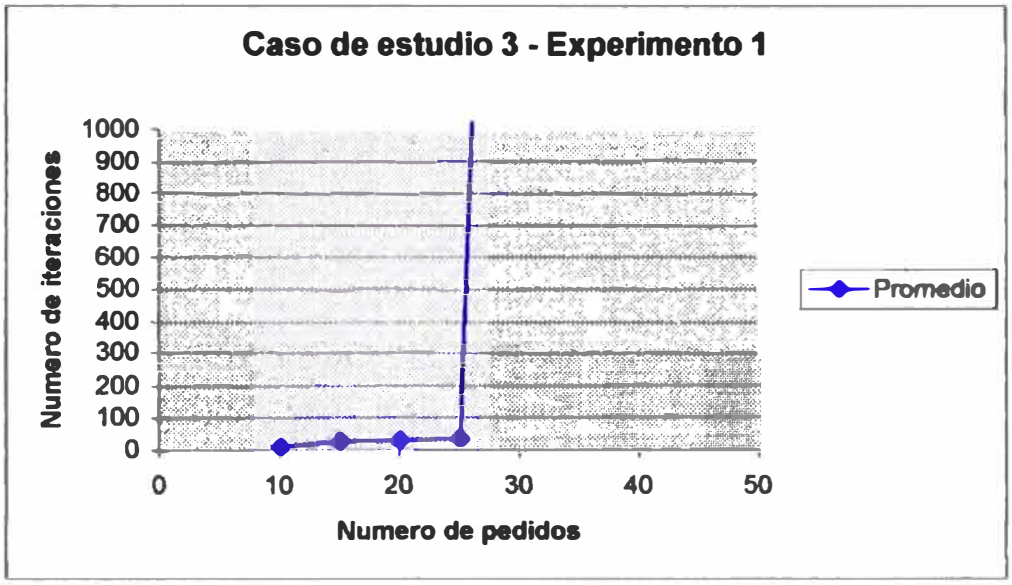
Experimento 1: Población constante y número de pedidos variable.

El primer experimento consideró el número de pedidos variable y a la población constante. Consideraron una población pequeña, de sólo 200 individuos, para agilizar el proceso de búsqueda. Usaron 3 corridas con distinta semilla de números aleatorios a fin de trazar la curva promedio "número de pedidos vs número de iteraciones necesarias para hallar la solución óptima".

Población	Número de Pedidos	Iteraciones semilla 1	Iteraciones semilla 2	Iteraciones semilla 3
200	10	9	9	10
200	15	20	27	26
200	20	29	2616	30
200	25	37	33	126
200	30	+5000	+5000	+5000
200	40	+5000	+5000	+5000

Como se pudo apreciar en los datos obtenidos, existió un punto en que la simulación generada con la segunda semilla ejecuto mas iteraciones antes de alcanzar la solución óptima con respecto a la primera semilla. Para la evaluación el resultado promedio se eliminaron estos resultados extraños a fin de obtener una gráfica más regular.

Población	Número pedidos	Iteraciones semilla 1	Iteraciones semilla 2	Iteraciones semilla 3	Iteraciones promedio
200	10	9	9	10	9.3
200	15	20	27	26	24.3
200	20	29	2616	30	29.5
200	25	37	33	35	35.0
200	30	+5000	+5000	+5000	+5000
200	40	+5000	+5000	+5000	+5000



De estos resultados se pudo apreciar que la eficacia del método es inversamente proporcional al número de pedidos o tareas asignadas. Así se vio que para 10, 15 y 20 pedidos el algoritmo respondió rápidamente obteniendo resultados en menos de 50 iteraciones. Sin embargo se apreció un cambio muy brusco en las simulaciones con 25 y 30 pedidos, simplemente el algoritmo dejó de encontrar la solución óptima para las simulaciones con 30 pedidos a pesar de que en la simulación 25 aun conservaba un buen desempeño. En las iteraciones con 35 y 40 pedidos,

las simulaciones siguieron sin poder encontrar la solución óptima llegando a alcanzar el máximo número de iteraciones fijado para estos experimentos.

Como se apreció en el grafico, en el intervalo de las simulaciones de 10 a 25 tareas programadas se produjo un incremento casi lineal del número de iteraciones necesarias para encontrar la solución óptima. Sin embargo en el intervalo de las simulaciones 25 a 30 ocurrió un fenómeno que impidió al método seguir hallando la solución óptima. Este fenómeno mereció ser analizado con mayor detenimiento por lo cual se realizaron experimentos para 26, 27, 28 y 29 pedidos, los resultados se mostraron el la siguiente tabla:

Población	Número de Pedidos	Iteraciones semilla 1	Iteraciones semilla 2	Iteraciones semilla 3
200	10	9	9	10
200	15	20	27	26
200	20	29	2616	30
200	25	37	33	35
200	26	76	519	126
200	27	37	+5000	4815
200	28	1230	+5000	+5000
200	29	+5000	+5000	1153
200	30	+5000	+5000	+5000
200	40	+5000	+5000	+5000

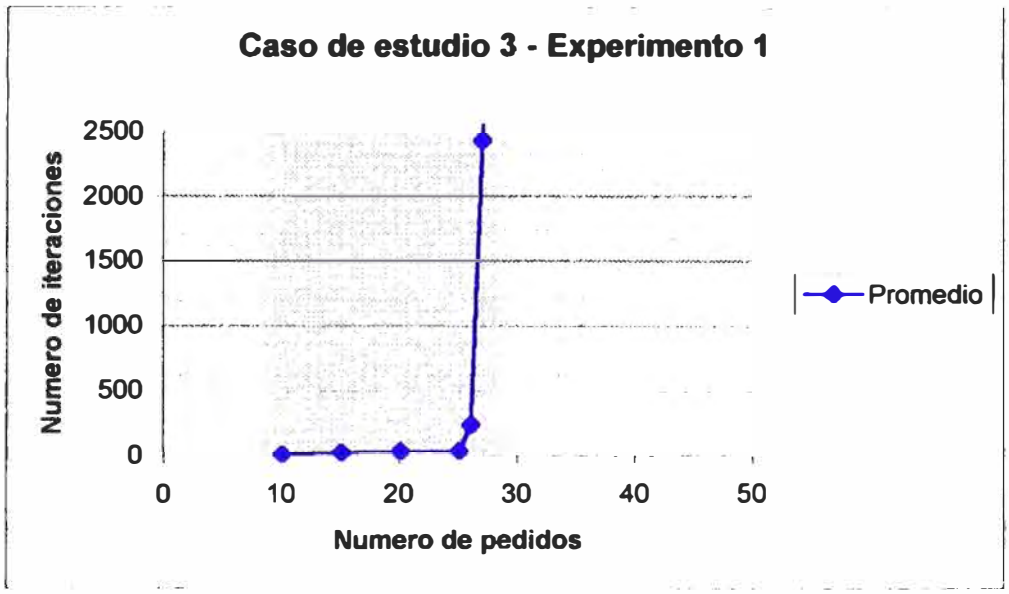
Nuevamente se vieron en algunas simulaciones generadas con la segunda y tercera semilla ejecutaron muchas iteraciones comparadas con el promedio de las semillas restantes, para una mejor evaluación los

resultados extraños fueron eliminados, obteniéndose de esta manera los siguientes resultados promedios:

Población	Número Pedidos	Iteraciones semilla 1	Iteraciones semilla 2	Iteraciones semilla 3	Iteraciones promedio
200	10	9	9	10	9.3
200	15	20	27	26	24.3
200	20	29	2616	30	29.5
200	25	37	33	35	35.0
200	26	76	519	126	240.3
200	27	37	+5000	4815	2426.0
200	28	1230	+5000	+5000	+5000
200	29	+5000	+5000	1153	+5000
200	30	+5000	+5000	+5000	+5000
200	40	+5000	+5000	+5000	+5000

Esta tabla mostró información muy interesante, revela un comportamiento que no se había apreciado en los experimentos anteriores debido al tamaño de los intervalos; se ve que, para el rango de 25 a 30 pedidos, el número de iteraciones sufrió un incremento muy pronunciado.

De estos resultados se podo apreciar que el número de pedidos y el número de iteraciones necesarias para hallar la solución óptima guardaban una proporción casi geométrica. Es decir llegó un momento en que el algoritmo genético requirió de un mayor incremento del número de iteraciones necesarias con respecto al incremento del número de pedidos. Graficando estos resultados con los anteriores se obtuvo la siguiente curva.



Esta gráfica hizo recordar a una asidota, aunque no fue posible determinar si efectivamente, en la simulación de un problema de asignación de recursos y tareas para una población dada, existió un punto en el cual el método de búsqueda de algoritmos genéticos encontraría la solución óptima en el infinito, si se pudo afirmar lo siguiente:

Al simular un problema de asignación de recursos y tareas mediante el método de búsqueda de algoritmos genéticos y para una población dada, existe un punto (número de pedidos), en el cual el número promedio de iteraciones requeridas, para que el método encuentre la solución óptima, empieza a incrementarse geométricamente proporcional al incremento del número de pedidos a ser ejecutados.

Esto nos reveló un comportamiento muy especifico del método de búsqueda de algoritmos genéticos que se evaluó. Ante esta evidencia los autores se formularon la siguiente interrogante: ¿Era posible afirmar que si

el número de tareas a asignar es lo suficientemente grande el método de algoritmos genéticos seria incapaz de hallar la solución?. Para dar respuesta a esta pregunta se retomo una de las consideraciones fundamentales del método explicado en la parte de fundamento teórico de esta tesis: el concepto de la población. Como se había indicado la población aseguraba la diversidad de la base genética necesaria para que el método de algoritmos genéticos realizara una búsqueda apropiada. Entonces, se pudo suponer que en los casos de las simulaciones de 30, 35 y 40 pedidos, no se encontró una solución debido a que la población de 200 individuos fue insuficiente para garantizar una base genética lo suficientemente diversa para que el método de algoritmos genéticos realizara una búsqueda apropiada. Además los autores no olvidaron que el código genético de los individuos de la población es más complejo cuando es mayor el número de tareas a ser programadas, requiriendo de esta forma una mayor diversidad genética.

Para dar respuesta a la interrogante de que si el método de algoritmos genéticos sería capaz de resolver el problema si se le incrementaba la población, se realizo el experimento 2

Experimento 2: Población variable número de pedidos constante.

Como se vio en el primer experimento, no se pudo encontrar la solución óptima al problema planteado en el caso 3 para 40 pedidos, llegando al limite de las 5,000 iteraciones. Se planteo el siguiente experimento, manteniendo el número de pedidos fijos en 40 y realizando incrementos de la población con la finalidad de analizar el comportamiento de la

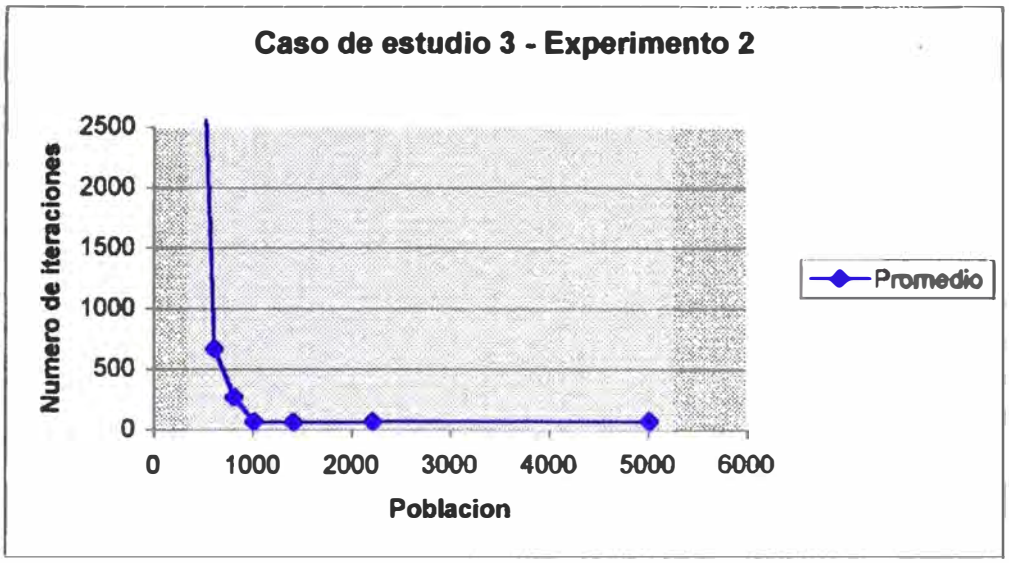
búsqueda al incrementar la población. En este experimento también se realizaron 3 corridas con semillas distintas (las mismas semillas que en el experimento anterior) para cada población con la finalidad de graficar la curva población vs número de iteraciones.

Población	Número de pedidos	Iteraciones semilla 1	Iteraciones semilla 2	Iteraciones semilla 3
200	40	+5000	+5000	+5000
400	40	+5000	+5000	+5000
600	40	1284	+5000	55
800	40	687	73	51
1000	40	64	77	58
1400	40	62	+5000	59
2200	40	53	966	75
5000	40	68	68	77

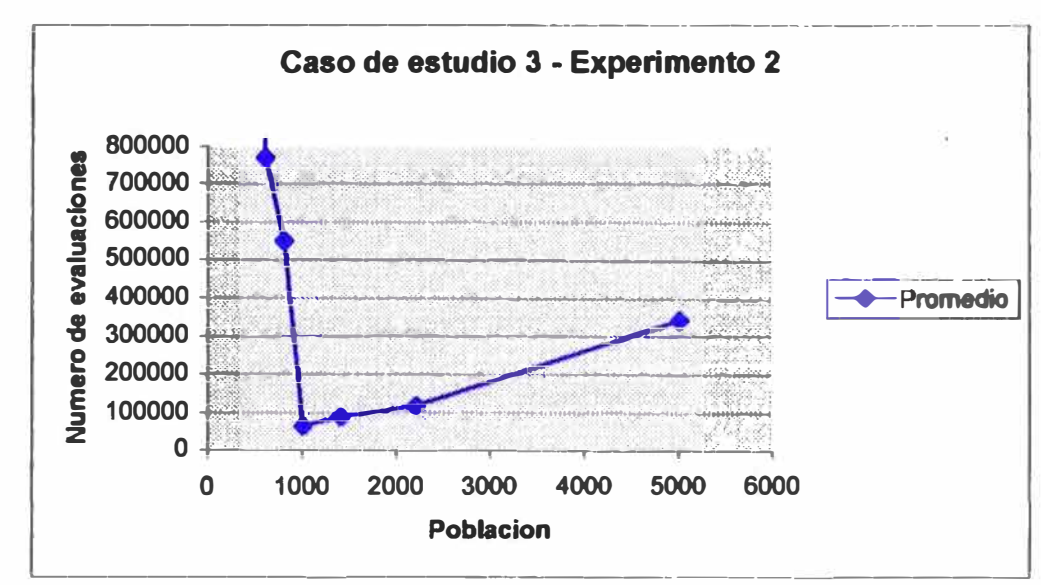
Se eliminaron nuevamente los resultados extraños, obteniendo los siguientes resultados promedios:

Población	Número pedidos	Iteraciones semilla 1	Iteraciones semilla 2	Iteraciones semilla 3	Iteraciones promedio
200	10	9	9	10	9.3
200	15	20	27	26	24.3
200	20	29	2616	30	29.5
200	25	37	33	35	35.0
200	30	+5000	+5000	+5000	+5000
200	40	+5000	+5000	+5000	+5000

El siguiente grafico mostró la relación entre el número de iteraciones y la población:



De esta gráfica se pudo deducir que el incremento de la población permitió que la simulación encontrara la solución óptima del problema de asignación de tareas planteado. Los autores plantearon otra interrogante al preguntarse si ¿Era posible afirmar que el método de búsqueda de algoritmos genéticos aumentaba su eficacia cuando trabaja con grandes poblaciones de individuos?. Los autores recalcaron que al tener una población más numerosa, el número de cálculos realizado en cada iteración se incrementaban; así que lo correcto era comparar el número de individuos que conforman la población vs el número de evaluaciones necesarias para encontrar la solución óptima. El número de evaluaciones se obtuvo multiplicando el número de individuos que conforman la población por el número de iteraciones necesarias para hallar la solución óptima del caso planteado.



Estudiando la gráfica obtenida se observó un comportamiento interesante se pudo apreciar que para pequeñas poblaciones el número de iteraciones necesarias para encontrar la solución óptima es muy grande, pero, conforme se iba incrementando la población, el número de evaluaciones realizadas disminuía, llegando a un mínimo de las evaluaciones realizadas. Después de este punto a mayores incrementos de la población el número de evaluaciones se incrementaba.

De estos resultados se llegó a la siguiente conclusión:

Al simular un problema de asignación de recursos y tareas mediante el método de búsqueda de algoritmos genéticos, existe una población óptima, en la cual el resultado del problema planteado, puede ser obtenido con el menor número de evaluaciones.

Regresando al grafico anterior, se pudo apreciar que una mayor población tiene mayor posibilidad de encontrar la solución óptima. Por lo cual afirmó

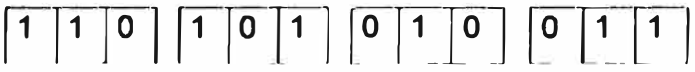
que efectivamente un incremento de la población, trae consigo un incremento de la base genética necesaria para que el método de algoritmos genéticos realizara la búsqueda exitosamente.

Sin embargo, también se pudo apreciar otro comportamiento interesante en esta gráfica: para pequeñas poblaciones, el número de iteraciones necesarias para hallar la solución óptima era inversamente proporcional al número de individuos de la población; pero llegaba un punto a partir del cual el número de iteraciones promedio se mantenía casi constante, a pesar de que la población experimentaba grandes incrementos. Esto se debía a que para poblaciones pequeñas (como las de 200 y 400 individuos) la diversidad genética era insuficiente para contener individuos con características favorables que permitieran encontrar la solución óptima. Cuando la población se incrementaba se llegó a alcanzar la base genética necesaria para realizar la búsqueda; fue por esta razón que a pesar de que la población seguía incrementándose considerablemente no se apreció una disminución substancial del número de iteraciones de simulación. Fue lógico que no se observara esta disminución ya que el algoritmo había alcanzado la base genética necesaria para realizar la búsqueda. Un incremento de la población en este caso no aumentaba considerablemente la diversidad de la base genética. Los autores ilustraron esta idea con un ejemplo.

Si se tenia una población genética conformada por cuatro individuos de la siguiente forma:

Y el individuo solución era el siguiente:

Como se pudo apreciar, el individuo solución poseía un valor 1 en su primer gen, mientras que ningún individuo en la población inicial lo poseía. Entonces las posibilidades de que el método de algoritmos genéticos encontrara la solución, eran basabas únicamente en la probabilidad de que el valor 1 en el primer gen de un individuo fuera generado por medio del operador de mutación, la cual era básicamente estocástico; y en nuestro experimento el porcentaje de ocurrencia del operador de mutación era solo del 1%. Sin embargo, si se hubiera contado con individuos con un valor 1 en el primer gen, la probabilidad de que este código se extienda era mayor, ya que también estaría afectada por el operador genético de crossover, que en nuestro experimento tenia un porcentaje de ocurrencia del 90%. Una vez que el método de algoritmos genéticos alcanzara la diversidad necesaria, el aumento de la población no afectaría significativamente la base genética, así se demuestra en el ejemplo:



Ambas poblaciones dependían de igual manera, de los operadores genéticos para encontrar la solución óptima a pesar de que la segunda era 50% más grande que la primera.

De estos resultados se pudo concluir que la diversidad de la base genética era un factor muy importante en la eficiencia del método de algoritmos genéticos.

Al simular un problema de asignación de recursos y tareas, mediante el método de búsqueda de algoritmos genéticos, existe una población mínima para proveer la diversidad genética necesaria, a fin de que el algoritmo converja hallando la solución óptima al problema planteado. El valor de esta población mínima es directamente proporcional a la complejidad de la cadena genética de los individuos.

El operador de mutación es otra alternativa para generar una mayor diversidad de la base genética. Se analizaron los resultados incrementando el porcentaje de ocurrencia del operador de mutación, de 1% a 5% y posteriormente 10%. Tal como se apreció en el siguiente cuadro.

Población	Número de pedidos	Mutación 1%	Mutación 5%	Mutación 10%
200	10	9	3	13
200	15	20	47	18
200	20	29	26	46
200	25	37	302	1126
200	26	76	48	47
200	27	37	1104	714
200	28	1230	+5000	+5000
200	29	+5000	238	64
200	30	+5000	+5000	61
200	40	+5000	+5000	+5000

Del resultado se observó que existe una ligero aumento del número de iteraciones necesarias para encontrar la solución óptima. Pero también se notaron simulaciones en las cuales el número de iteraciones bajó considerablemente, esto se debió a que al tener porcentajes de mutación muy altos, se incrementaron las posibilidades de que un mejor código genético aparezca. El resultado fue que estas simulaciones presentaron un número de iteraciones considerablemente más bajo.

Se analizó una corrida en detalle, observando que el proceso de convergencia en general se retrasaba, debido a que los porcentajes de ocurrencia de los demás operadores genéticos (crossover y reproducción) se habían reducido. Fue interesante una comparación con la Biología, donde si bien el proceso de mutación es aleatorio, la probabilidad de ocurrencia se eleva cuando el código genético es similar. Así, la

descendencia de padres consanguineos, tienen una mayor probabilidad de generar una mutación, que los padres que no tienen parentesco, es decir, código genético diferente.

En el método de algoritmo genéticos, existe una gran probabilidad de que los padres tengan el mismo código genético, cuando un individuo se ha reproducido lo suficiente como para que su código genético conforme una buena parte de la población o incluso toda, en otras palabras, cuando la base genética se ha reducido. Si en estas circunstancias, se elevaría el porcentaje de mutación, se generarían individuos con nuevo código genético, restableciendo de esta manera, la diversidad genética. De esta forma podríamos mejorar el proceso evolutivo creando un operador de mutación que incremente sustancialmente su probabilidad cuando el código genético de los padres sean iguales, y se mantenga en menor porcentaje cuando el código genético de los padres sean diferentes, optimizando de esta forma el método de algoritmos genéticos.

4.5.2 Método de Sistemas Multiagentes con Sistemas Clasificadores Resolución Caso 3

Para evaluar la eficacia del método de sistemas multiagentes con sistemas clasificadores, se usó un método de evaluación similar al de algoritmos genéticos, es decir, el proceso de entrenamiento se llevo a cabo primero con un número de tareas reducido (10 pedidos), y se

incrementó paulatinamente el número de pedidos hasta llegar a un número de 40 pedidos.

Los estados y las acciones definidas fueron las mismas que se usaron para resolver los casos 1 y 2.

Nº	Estado	Descripción
1	Cumple con el tiempo de entrega y no existe superposición	El agente percibe que esta cumpliendo sus restricciones
2	Existe superposición	El agente percibe que otro pedido se ejecuta en el mismo rango de tiempo en la misma máquina
3	No cumple tiempo de entrega	El agente percibe que no esta cumpliendo con su plazo de entrega.
4	No cumple con tiempo de entrega y existe superposición	El agente percibe que otro pedido se ejecuta en el mismo rango de tiempo, en la misma máquina y que no esta cumpliendo con su plazo de entrega

Nº	Acción	Descripción
0	Búsqueda izquierda	El agente realiza una búsqueda en un tiempo de inicio anterior al actual en la misma máquina, en caso de no encontrarlo no realiza desplazamiento
1	Búsqueda derecha	El agente realiza una búsqueda en un tiempo de inicio posterior al actual en la misma máquina, en caso de no encontrarlo no realiza desplazamiento.
2	Moverse a cualquier máquina	Realiza un salto aleatorio a cualquier máquina
3	Moverse a máquina	Busca un desplazamiento en una máquina de

	de mejor desempeño	mejor desempeño y ubica su inicio en la posición más tarde posible para cumplir con su plazo de entrega.
4	Superponerse con el agente anterior más próximo	Superpone el inicio del agente con el inicio del agente anterior más próximo, en caso de no encontrarlo no realiza desplazamiento.
5	No hacer nada	No desempeña ninguna acción
6	Búsqueda izquierda en todas las máquinas	El agente realiza una búsqueda en un tiempo de inicio anterior al actual en todas las máquinas, en caso de no encontrarlo no realiza desplazamiento
7	Búsqueda derecha en todas las máquinas	El agente realiza una búsqueda en un tiempo de inicio posterior al actual en la misma máquina, en caso de no encontrarlo no realiza desplazamiento.
8	Superponerse con el agente posterior más próximo	Superpone el inicio de agente con el inicio del agente posterior más próximo, en caso de no encontrarla se ubica al inicio de operaciones de la máquina.
9	Superponerse con uno de los agentes anteriores	Superpone el inicio del agente con el inicio del un agente anterior seleccionado aleatoriamente, en caso de no encontrarla se ubica al inicio de operaciones de la máquina.

La ejecución de estas reglas se realizó sin previo entrenamiento de los agentes. Cada agente contó con 12 reglas iniciales, generadas aleatoriamente, y posteriormente, durante el desarrollo de la corrida, se generarían mas reglas por medio de la metodología de sistemas clasificadores, eliminando las reglas que tengan una valoración baja. Una vez terminada la corrida, el conocimiento pasó a formar parte del conocimiento histórico de la población y se heredaron las mejores 10

reglas a la nueva generación de la siguiente corrida. El conocimiento histórico fue almacenado hasta dos generaciones anteriores.

Conocimiento	
Número de reglas por agente	12
Reglas que se mantienen por sistemas clasificadores	10
Reglas generadas por sistemas clasificadores	2
Máximo número de reglas heredadas a la siguiente generación	12
Número máximo de reglas generadas durante simulación	Solo si no existe acción para el estado
Simulación	
Número máximo de movimientos por corridas	1000
Número de corridas (Generaciones)	10

Se inició la evaluación de la metodología examinando el problema del caso 3 para 10 agentes.

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	10	0	211
2	10	0	2
3	10	0	1
4	10	0	3
5	10	0	3

6	10	0	2
7	10	0	5
8	10	0	4
9	10	0	1
10	10	0	5

Como se observó, el método resolvió muy fácilmente el problema. Sin considerar la primera corrida, en promedio el problema fue resuelto en 3 iteraciones. Como se había mencionado, los agentes no recibieron entrenamiento previo, es lógico pensar que se entrenaron en la primera corrida y luego trabajaron con el conocimiento adquirido en las siguientes corridas, se trato de comprobar esta suposición en las siguientes corridas.

Corrida	Número de Pedidos /máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	15	0	58
2	15	0	5
3	15	0	19
4	15	0	13
5	15	0	10
6	15	0	15
7	15	0	12
8	15	0	6
9	15	0	13
10	15	0	6

En este caso, si bien el número de iteraciones promedio había aumentado, aun se observaba un comportamiento similar al observado con 10 tareas, en el cual la primera iteración sirvió de entrenamiento para los agentes y a

partir de la segunda se observó un comportamiento promedio de 11 iteraciones

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	20	0	94
2	20	12	745
3	20	12	151
4	20	6	837
5	20	6	877
6	20	24	564
7	20	24	376
8	20	6	995
9	20	6	614
10	20	25	556

En este caso, no se ajustó al comportamiento habitual que se presento en las dos simulaciones anteriores. A pesar de que se halló la solución óptima en la primera iteración, el método no pudo hallarlas en las siguientes corridas. Antes de investigar más detenidamente este caso, se decidió realizar las siguientes simulaciones para 25, 30 y 40 pedidos.

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	25	6	288
2	25	6	215
3	25	6	89
4	25	0	222
5	25	0	387
6	25	21	43

7	25	0	468
8	25	21	267
9	25	13	2
10	25	23	2

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	30	18	247
2	30	6	242
3	30	24	663
4	30	6	291
5	30	6	242
6	30	0	237
7	30	274	3
8	30	96	131
9	30	169	932
10	30	107	51

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	40	56	221
2	40	491	828
3	40	0	9
4	40	0	37
5	40	0	14
6	40	0	27
7	40	0	17
8	40	0	14
9	40	0	23
10	40	0	35

Las simulaciones para 25 y 30 pedidos, aparentemente confirmaron la ineficacia del modelo para resolver problemas del caso 3 con número de pedidos mas altos que 15. Sin embargo, la simulación del problema con 40 pedidos volvió a mostrar un comportamiento similar al de las simulaciones de 10 y 15 pedidos, es decir, una etapa de entrenamiento en las primeras corridas y luego una clara facilidad para resolver el problema en las siguientes corridas.

Se presentó la interrogante: ¿el método era verdaderamente ineficaz o estas diferencias de resultados se debían a alguna diferencia entre las simulaciones de 10, 15 y 40 pedidos contra las simulaciones de 20, 25 y 30 pedidos?. Para aclarar esta duda, se basó en uno de los conceptos fundamentales de los sistemas de multiagentes inteligentes, la definición de autonomía en un agente. Como se había indicado en el fundamento teórico, la conducta de un agente puede ser basada en su propia experiencia y en el conocimiento preprogramado, el cual es generado en la construcción del agente, para un ambiente particular en el cual opera. Gracias a esta definición, se pudo determinar que la diferencia fundamental entre los sistemas que encontraban la solución óptima y los que no, debía ser la conducta de los agentes del sistema. En la simulación del sistema en este caso de estudio, habíamos establecido que los agentes en cada simulación no tenían entrenamiento previo, por lo cual su conducta estaba basada únicamente en su propia experiencia, adquirida durante la simulación. Se decidió examinar la experiencia adquirida por los agentes en cada simulación a fin de encontrar posibles diferencias o similitudes.

A continuación se muestra el conocimiento de las simulaciones que lograron resolver el problema:

Conocimiento adquirido al finalizar la simulación para 10 pedidos	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas
Existe superposición	Búsqueda izquierda
	Búsqueda derecha
	Moverse a máquina de mejor desempeño
	Superponerse con el agente anterior más próximo
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas
No cumple tiempo de entrega	Búsqueda izquierda en todas las máquinas

Conocimiento adquirido al finalizar la simulación para 15 pedidos	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda izquierda
	Superponerse con el agente anterior más próximo
	Búsqueda izquierda en todas las máquinas
	Superponerse con el agente posterior más próximo
Existe superposición	Búsqueda izquierda
	Búsqueda derecha
	Moverse a máquina de mejor desempeño
	Superponerse con el agente anterior más próximo
	Búsqueda izquierda en todas las máquinas

No cumple tiempo de entrega	Moverse a máquina de mejor desempeño
	Búsqueda izquierda en todas las máquinas
No cumple con tiempo de entrega y existe superposición	Búsqueda izquierda

Conocimiento adquirido al finalizar la simulación para 40 pedidos	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda derecha
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Superponerse con el agente posterior más próximo
Existe superposición	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas
No cumple tiempo de entrega	Búsqueda izquierda
	Búsqueda derecha
	Moverse a cualquier máquina
	Moverse a máquina de mejor desempeño
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas
No cumple con tiempo de entrega y existe superposición	Superponerse con el agente anterior más próximo

La experiencia adquirida por los agentes en las simulaciones en las cuales no pudieron resolver el problema satisfactoriamente es:

Conocimiento adquirido al finalizar la simulación para 20 pedidos	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Superponerse con el agente posterior más próximo
Existe superposición	Búsqueda izquierda
	Búsqueda derecha
	Moverse a cualquier máquina
	Moverse a máquina de mejor desempeño
	Superponerse con el agente anterior más próximo
	No hacer nada
	Búsqueda izquierda en todas las máquinas
No cumple tiempo de entrega	Moverse a cualquier máquina
	No hacer nada
	Búsqueda derecha en todas las máquinas

Conocimiento adquirido al finalizar la simulación para 25 pedidos	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda derecha
Existe superposición	Búsqueda izquierda
	Búsqueda derecha
	Moverse a cualquier máquina
	Moverse a máquina de mejor desempeño
	Superponerse con el agente anterior más próximo
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas

No cumple tiempo de entrega	No hacer nada
	Búsqueda derecha en todas las máquinas
No cumple con tiempo de entrega y existe superposición	Moverse a máquina de mejor desempeño
	Búsqueda izquierda en todas las máquinas

Conocimiento adquirido al finalizar la simulación para 30 pedidos	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda derecha
	Superponerse con el agente anterior más próximo
	Superponerse con el agente posterior más próximo
Existe superposición	Búsqueda derecha
	Moverse a cualquier máquina
	Moverse a máquina de mejor desempeño
	Superponerse con el agente anterior más próximo
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas
No cumple con tiempo de entrega y existe superposición	Búsqueda derecha

Se observó que el conocimiento es variado en los agentes, sin embargo no se pudo asegurar que algún patrón de conducta similar fue establecido por el conjunto de reglas, para estudiar este fenómeno, se decidió retroalimentar la simulación de 15 pedidos con el conocimiento generado por la simulación de 10 pedidos, los resultados se muestran a continuación:

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	15	0	6
2	15	0	7
3	15	0	3
4	15	0	4
5	15	0	1
6	15	0	1
7	15	0	2
8	15	0	3
9	15	0	3
10	15	0	6

De igual forma, sé retroalimento la experiencia adquirida como resultado de la simulación de 10 pedidos a los demás casos.

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	20	0	5
2	20	0	10
3	20	0	11
4	20	0	76
5	20	0	8
6	20	0	5
7	20	0	13
8	20	0	7
9	20	0	17
10	20	0	24

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	25	0	30
2	25	0	25
3	25	0	210
4	25	0	21
5	25	0	4
6	25	0	344
7	25	0	57
8	25	0	41
9	25	0	23
10	25	0	61

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	30	0	9
2	30	0	246
3	30	0	11
4	30	0	74
5	30	0	176
6	30	0	10
7	30	0	12
8	30	0	137
9	30	0	32
10	30	0	201

Corrida	Número de Pedidos / máquinas	Demoras (0 = óptima)	Iteración mejor solución
1	40	0	9
2	40	0	11
3	40	0	74

4	40	0	176
5	40	0	10
6	40	0	12
7	40	6	327
8	40	0	137
9	40	0	23
10	40	0	45

El resultado de este cambio fue muy claro, con esta adaptación, casi todas las corridas de las simulaciones encontraron la solución óptima en menos de 100 iteraciones en promedio. El conocimiento obtenido como resultado final en la ultima iteración de la simulación de 40 tareas fue:

Conocimiento adquirido al finalizar la simulación para 40 pedidos	
Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda izquierda
Existe superposición	Búsqueda izquierda
	No hacer nada
	Búsqueda derecha en todas las máquinas
No cumple tiempo de entrega	Búsqueda izquierda
	Búsqueda derecha
	Moverse a cualquier máquina
	Superponerse con el agente anterior más próximo
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas

Este resultado también fue diferente al obtenido en la anterior simulación de 40 tareas, que se muestra a continuación:

Estado	Acción
Cumple con el tiempo de entrega y no existe superposición	Búsqueda derecha
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Superponerse con el agente posterior más próximo
Existe superposición	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas
No cumple tiempo de entrega	Búsqueda izquierda
	Búsqueda derecha
	Moverse a cualquier máquina
	Moverse a máquina de mejor desempeño
	No hacer nada
	Búsqueda izquierda en todas las máquinas
	Búsqueda derecha en todas las máquinas
No cumple con tiempo de entrega y existe superposición	Superponerse con el agente anterior más próximo

En este caso, solo existe una acción para el estado "cumple con el tiempo de entrega y no existe superposición", la cual es: "desplazamiento a la izquierda en la misma máquina", que fue interpretado como una acción de optimización, a diferencia de las cuatro acciones que se tenían en la simulación anterior. En cuanto a las acciones en el estado de "superposición", existe una acción en común, "la búsqueda a la derecha en todas las máquinas". Para el estado de "no cumplir el plazo de entrega" es donde se encontraron más acciones en común, las acciones de

"desplazarse hacia la izquierda y derecha en la misma máquina", "moverse a otra máquina", "deslazarse a la derecha e izquierda en otras máquinas" y por último "no hacer nada".

Un comportamiento interesante fue el que se observó en la séptima corrida, de la ultima simulación del problema para 40 tareas, simplemente no pudo encontrar la solución óptima en esa corrida, aunque en la siguiente corrida si fue capaz de hallarla, analizando el conocimiento resultante de esa corrida, se descubrió que de alguna manera los agentes habían perdido las acciones de "desplazamiento hacia la izquierda en cualquier máquina" para el estado de "incumplimiento con el plazo de entrega".

Analizando este caso mas detenidamente, se observó que las 6 unidades de tiempo de retraso se debía a que dos agentes se encontraban ejecutándose en una misma máquina, dejando otra máquina libre. La solución seria muy sencilla, el segundo agente debió desplazarse hacia la máquina libre, es decir, la acción de búsqueda a la izquierda en cualquier máquina. Sin embargo, al perderse el conocimiento de las acciones de desplazamiento, solo si esta regla fuese generada como resultado del proceso evolutivo de los sistemas clasificadores, se encontraría la solución a este problema. En el caso de la corrida 7, no se encontró esta regla, pero en la corrida 8 se volvió a generar y se transmitió como conocimientos a las siguientes generaciones.

Este fenómeno se presento repetidas veces durante la simulación, por lo cual concluimos que en las primeras simulaciones, en las cuales no se encontró la solución óptima, se debió fundamentalmente a que esta regla clave, para la resolución del problema, no fue encontrada. Mientras que las simulaciones que si encontraron esta regla, resolvieron el problema mucho más rápido que cualquier método analizado. Este conocimiento adquirido, fue la fortaleza del método multiagentes inteligentes, si bien requirió de un periodo de entrenamiento hasta conseguir este conocimiento, una vez que el agente lo adquirió, el método llegó a ser sumamente eficiente, resolviendo el problema tal como lo haría un experto. A diferencia del método de algoritmos genéticos, el cual no fue capaz de diferenciar la complejidad del problema, dependiendo únicamente en la complejidad del código genético de los individuos, el método de multiagentes inteligentes, logró adaptar su conocimiento de acuerdo a la complejidad del problema.

La metodología de multiagentes inteligentes resultó ser la más eficiente para resolver problemas similares al caso 3, fue capaz de identificar el conocimiento necesario para resolver fácilmente el problema y si este conocimiento era heredado a las siguientes simulaciones, los agentes serian capaces de adaptar el nuevo conocimiento a las nuevas situaciones, es decir, adaptarse a un incremento de la complejidad del problema.

CAPITULO V

CONCLUSIONES Y RECOMENDACIONES

5. CONCLUSIONES Y RECOMENDACIONES

En la presente tesis se desarrollaron cuatro modelos haciendo uso de diferentes metodologías de inteligencia artificial, para la resolución de problemas de asignación de recursos y tareas. Las metodologías son:

- Búsquedas Exhaustivas.
- Algoritmos Genéticos.
- Sistemas Multiagentes con Sistemas Clasificadores.
- Sistemas Multiagentes con Sistemas Clasificadores y Estrategias Evolutivas.

Las conclusiones a las cuales se llego con la tesis fueron las siguientes:

1. Se demostró que la búsqueda exhaustiva a pesar que garantiza encontrar la solución óptima, es inaplicable para este tipo de problemas, debido al enorme número de evaluaciones de las posibles soluciones que se tiene que realizar. Evaluando problemas medianamente complejos de asignación de recursos y tareas se demostró que este método requiere años de procesamiento ininterrumpido con los mejores computadores disponibles.

2. Se demostró que la metodología de algoritmos genéticos es una metodología eficaz para resolver problemas de pequeñas a mediana complejidad, es decir, aquellos que se puedan codificar en una cadena genética de mediana longitud. Nos brinda una solución bastante aceptable, en la mayoría de casos la solución óptima.

3. Se demostró que la metodología de algoritmos genéticos es una metodología eficiente para resolver problemas de pequeña y mediana complejidad. Tomando como base la metodología de búsqueda exhaustiva un problema calculado para ser resulto en 50 días fue resuelto en menos de 10 segundos.

4. Para problemas complejos donde se requiere una cadena genética de gran longitud el modelo basado en algoritmos genéticos requiere de una base genética (población) muy grande para garantizar una solución aceptable.

5. Cualquier cambio en la definición del problema hace necesaria volver a ejecutar la simulación del modelo basado en algoritmos genéticos nuevamente.

6. El método de algoritmos genéticos tiene una gran dependencia de la definición de la función fitness y la representación del problema en el código genético.

7. Los Sistemas Multiagentes con Sistemas Clasificadores en cambio demostraron ser mucho menos eficiente en problemas de mediana y

pequeña complejidad que algoritmos genéticos y medianamente eficiente para problemas de mayor complejidad y ciertos problemas de mediana complejidad. La disminución de la eficiencia de debe básicamente a que esta metodología requiere de un periodo de aprendizaje que depende directamente del número de agentes que tenga el sistema.

8. El modelo basado en sistemas multiagentes y sistemas clasificadores, demostró ser capaz de resolver problemas como los planteados en la presente tesis; haciendo uso de un conjunto de reglas simples, obteniéndose como resultado un comportamiento bastante complejo del sistema, gracias a la interacción entre ellos.

9. La metodología de sistemas multiagentes nos permite modelar fácilmente el problema planteado en esta tesis de una manera más comprensible y se asemeja más al sistema real.

10. Cuando el modelo basado en sistemas multiagentes adquiere un comportamiento adecuado, ya sea predefinido o por el proceso de entrenamiento, éste encuentra rápidamente la solución y en algunos casos específicos, mucho más rápidos que el modelo basado en algoritmos genéticos.

11. El modelo basado en sistemas multiagentes es adaptativo, es decir, permite reorganizar la asignación de recursos y tareas, partiendo de soluciones previas y basándose en el conocimiento adquirido. En este sentido para

pequeños cambios en el problema, no es necesario volver a ejecutar la simulación desde cero.

12. La eficiencia y eficacia del modelo de multiagentes inteligentes tiene una gran dependencia de la función de valoración, así como, de las condiciones iniciales y variables estocásticas del modelo, debido a que el modelo es un sistema no lineal.

13. Los modelos multiagentes requieren de un tiempo de aprendizaje para poder determinar el comportamiento más adecuado de cada uno de los agentes para la solución del problema. Este tiempo puede reducirse significativamente asignándole ciertas reglas de comportamiento que se consideran adecuadas para el problema.

14. El modelo basado en sistemas multiagentes, sistemas clasificadores y estrategias evolutivas, son una mejora del modelo basado en sistemas multiagentes y sistemas clasificadores atacando su desventaja de dependencia de la función de valoración, haciendo uso de estrategias evolutivas.

15. Este modelo basado en sistemas multiagentes, sistemas clasificadores y estrategias evolutivas no presenta dependencia de la función de valoración, ya que esta función es configurada automáticamente mediante las estrategias evolutivas, logrando resolver más eficientemente el problema planteado que el modelo basado en sistemas multiagentes y sistemas clasificadores.

16. Finalmente concluimos que los sistemas basados en inteligencia artificial ofrecen una muy buena alternativa para la solución de problemas de asignación de recursos y tareas, obteniendo los resultados óptimos en un tiempo aceptable. Así como una mayor adaptabilidad a los cambios, los que implican una reducción de costos al optimizar el uso de los recursos disponibles, cumpliendo satisfactoriamente las demandas de los clientes y de esta manera mejorado el servicio.

Recomendaciones:

1. Debido a que el problema planteado en la tesis es sumamente complejo a causa del gran número de restricciones que hace muy difícil resolver estos problemas por los métodos tradicionales recomendamos hacer uso de métodos de inteligencia artificial para asegurar una mayor eficiencia y eficacia de las soluciones de problemas complejos como el desarrollado en esta tesis.

2. En caso de problemas de asignación de recursos y tareas de pequeña y mediana complejidad, es decir, problemas con un número de tareas a programar no muy grande, recomendamos el uso de modelos basados en técnicas de computación evolutiva como algoritmos genéticos o estrategias evolutivas. Ya que estos demostraron ser los más eficientes para estos casos.

3. En el caso de problemas con mayor grado de complejidad recomendamos el uso de modelos basados en agentes inteligentes teniendo en consideración la importancia del aprendizaje además de garantizar un buen desempeño.

APENDICES

APENDICES

TEOREMA DE SCHEMADATA

Un esquema es una plantilla de similitud que describe un subconjunto de cadenas con las similitudes a ciertas posiciones de la cadena. Consideremos el alfabeto binario {0, 1}. Introduciremos un esquema añadiendo un símbolo especial a este alfabeto, el # o símbolo "no importa" que representa indistintamente al 0 o 1 en una posición particular. Con este alfabeto extendido podemos crear las cadenas con el alfabeto ternario (el Schemadata).

{0, 1, #}

Un esquema representa una cadena particular si a cada situación en el esquema 1 representa un 1 en la cadena, un 0 representa un 0, un # representa ambos. Como un ejemplo, consideremos las cadenas y Schemadata de longitud 5. El esquema

#101#

Describe un subconjunto con cuatro miembros

01010, 01011, 11010, 11011,

Consideremos una población de individuos (las cadenas) A_j, j = 1, 2,..., n contenido en la población A(t) a un instante (o generación) dado t (t = 0,1, 2...). Además la anotación para describir poblaciones, cadenas y alelos, Denotaremos como H para describir el Schemadata contenido en las cadenas individuales y poblaciones.

***V*:={0,1, #}.**

Para los alfabetos de cardinalidad k, hay $(k + 1)^l$ schemadatas dónde l es la longitud de la cadena. Para una población de cadenas con n miembros, existe a lo mas $n * 2^l$ schemadatas contenidos. Estos argumentos nos dan una idea de la magnitud de información que se procesa por los algoritmos genéticos.

Todos los schemadatas no se crean igual. Algunos son más específicos que otros. El esquema **011#1##** es una declaración más precisa sobre la similitud que el esquema **0#######**. Además, ciertos schemadatas se expanden más en la longitud de la cadena total que otros. El esquema **1####1#** expande una porción más grande de cadena que el esquema **1#1#####**, para cuantificar estas ideas, se introducen dos propiedades del esquema: el orden del esquema y la longitud definida.

Definición. *El orden de un esquema H, denotado por o(H), es el número de posiciones fijas (en un alfabeto binario, el número de 1's y 0's) presentes en la plantilla.*

Ejemplo. El orden del esquema **011#1##** es 4, considerando que el orden del esquema **0#######** es 1.

Definición. *La longitud definida de un esquema* ***H****, denotado por* δ***(H), es la distancia entre la primera y última posición especifica de la cadena.*

Ejemplo. El esquema **011#1##** tiene definiendo longitud δ = 4 porque la última posición específica es 5 y la primera posición específica es 1. Así, δ(**H**) = 5 - 1 = 4.

El Schemadata provee los medios básicos para analizar el efecto neto de reproducción y los operadores genéticos en los bloques de construcción contenidos dentro de la población. Permitiéndonos considerar al individuo y combinar los efectos de reproducción, crossover, y mutación en los schemadatas contenidos dentro de una población de cadenas. Supongamos que después de un tiempo *t* dado hay *m*(***H***,*t*) ejemplos de un esquema particular **H** contenido dentro de la población **A**(*t*). Durante la reproducción, una cadena se copia según su fitness, o más precisamente una cadena Ai se selecciona con la probabilidad:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

Después de escoger a una población no superpuesta de tamaño ***n*** con el reemplazo de la población **A**(*t*), nosotros esperamos tener *m*(***H***,*t*+1) representantes del esquema **H** en la población al instante *t*+1, esta cantidad esta dada por:

$$m(H, t+1) = \frac{m(H,t) n f(H)}{\sum_{j=1}^{n} f_j(t)}$$

Donde el *f*(**H**) es el fitness promedio de las cadenas que representan al esquema **H** al instante *t*. el fitness promedio de la población entera se define como:

$$\bar{f} = \frac{1}{n} \sum_{j=1}^{n} f_i$$

Así nosotros podemos escribir la ecuación de crecimiento de esquema reproductor como sigue:

$$m(H, t+1) = m(H,t) \frac{f(H)}{\bar{f}(t)}$$

Asumiendo ese *f*(***H***)/ $\bar{f}$ permanece relativamente constante para la *t* = 0,1,..., la ecuación precedente es una ecuación de diferencia lineal *x*(*t*+1) = ***a****x*(*t*) con coeficiente constante que tiene la solución *x*(*t*) = $a^t x(0)$. Así un esquema particular crece con el ratio del fitness promedio del esquema entre el fitness promedio de la población. El Schemadata con un valor de

fitness superior al promedio de la población recibirá un número creciente de muestras en la próxima generación, mientras el Schemadata con un valor de fitness promedio debajo del promedio de la población recibirá un número decreciente de muestras. Este comportamiento se lleva a cabo con cada esquema ***H*** contenido en una población particular **A** en paralelo. En otras palabras, todos los Schemadata en una población crecen o se deterioran según su esquema promedio exclusivamente bajo el funcionamiento de la reproducción. Los Schemadata sobre el promedio crecen y los Schemadata debajo del promedio se extinguen. Suponga que nosotros asumimos que un esquema particular ***H*** mantiene una cantidad $c\ \bar{f}$ sobre el promedio con el c una constante. Bajo esta asunción nosotros encontramos:

$$m(H,t+1) = m(H,t)\frac{(f+cf)}{f} = (1+c)m(H,t)$$

Empezando en t = 0 y asumiendo un valor estacionario de c, obtendremos la ecuación:

$$m(H,t) = m(H,0)(1+c)^t$$

Esta es una progresión geométrica o discreta análoga de una forma exponencial. La reproducción asigna aumentando exponencialmente (disminuyendo) el número de individuos sobre(bajo) el Schemadata promedio. El teorema fundamental de algoritmos genéticos es como sigue:

Teorema. *Usando la selección, crossover, y mutación del algoritmo genético normal, el corto, bajo-orden, y el superior al Schemadata*

promedio reciben incrementos exponencialmente de sus individuos en las poblaciones subsecuentes.

Los corto, bajo-orden, y superior al Schemadata promedio del promedio son llamados bloques de construcción, y el teorema fundamental indica que se espera que los bloques dominen la población. ¿Es esto bueno o malo por lo que se refiere a la meta original de optimización de la función? El teorema precedente no contesta esta pregunta. Más bien, la conexión entre el teorema fundamental y las propiedades de perfeccionamiento del algoritmo genético son proporcionadas por lo siguiente conjetura.

Hipótesis. *La Hipótesis del bloque de Construcción. La cadena óptima global en* Ω

$$f : \Omega \rightarrow R \qquad con \qquad \Omega = \{0,1\}^n$$

Puede ser dividida en subcadenas que son dadas por los bits de las posiciones fijas de los bloques de construcción.

BIBLIOGRAFÍA

1. BANZHAF WOLFGANG, NORDIN PETER. Genetic Programming. Morgan Kaufman Publishers, Inc, Primera impresión, 1998. San Francisco, California, USA.
2. FLOREANO DARIO, NICOUD JEAN-DANIEL, MONDADA FRANCESCO. Advances in Artificial Life. 5th European Conference, ECAL'99 Lausanne, Switzerland, September 1999, Proceedings. Lecture Notes in Artificial Intelligence 1674. Springer 1999. Berlin Alemania.
3. HERRERA VIEDMA E. Sistemas Clasificadores de Aprendizaje. Aproximaciones Difusas. Universidad de Granada, Trabajo de doctorado, 1995. Granada, España.
4. KALLEL L., NAUDTS B., ROGERS A. Theoretical Aspects of Evolutionary Computing. Springer-Verlag Co, Primera impresión, 2001. Heidelberg, Berlin, Alemania.
5. MELANIE MITCHEL. An Introduction to Genetic Algorithms. MIT Press, Sexta impresión, 1999. Cambridge, Massachusetts, USA.
6. MUSCETTOLA NICOLA, CHIEN STEVE. 2nd NASA INTL. WORKSHOP Plannig and Scheduling for Space. NASA Ames Research center, Segunda Conferencia, March 2000. Ames Research center, Moffett Feild CA, USA.

7. STEBB WILLI-HANS. The nonlinear workbook, Chaos, Fractals, Celular Automata, Neural Networks, Genetic Algorithms and Fuzzy logic. Wold Cientific Pubishing Co, Primera impresión, 1999. Farrer Road, Singapur.
8. TESSIER CATHERINE, CHAUDRON LAURENT, MULLER HEINZ-JURGEN. Conflicting Agents, Conflict Management in Multi-Agent Systems. Kluwer Academic Publishers, 2001. Massachusetts, USA.