Capítulo 1

Introducción

1.1 Motivación

El aprendizaje es una cualidad única en los seres vivos, se puede definir como la capacidad de adquirir conocimiento (experiencia) para modificar un comportamiento. Con el origen de la inteligencia artificial como ciencia, se han propuesto modelos computacionales que tratan de explicar e implantar el aprendizaje así como otros procesos antes exclusivos de los seres vivos como la visión, la planeación y el reconocimiento de patrones. Cuando surgió la inteligencia artificial, empezó con dos vertientes. Una consistiría en emular el aprendizaje que ocurre en el cerebro por medio del procesamiento de símbolos y usando la lógica formal. Otra vertiente emularía el funcionamiento de los componentes básicos del cerebro; es decir, las neuronas y los sistemas nerviosos (Graubard, 1993). Con el tiempo, ambas verticales se han desarrollado y otras han surgido; tal es el caso de la computación evolutiva.

El aprendizaje puede aplicarse con computación evolutiva ya que existe una base teórica que relaciona el aprendizaje con los procesos evolutivos. Piaget fue el primer autor en sugerir que la adquisición de conocimiento puede ser un proceso evolutivo (Piaget, 1980). Fue Donalt P. Cambell quien acuñó el término "epistemología evolutiva" que es simplemente el estudio del conocimiento desde un punto de vista biológico y evolutivo (Campbell, 1974). Teniendo como fundamento la epistemología evolutiva, se han sugerido varios modelos computacionales evolutivos que funcionan como mecanismos de apredizaje. Las primeras referencias que se tienen sobre el desempeño de sistemas evolutivos son los trabajos de Lawrence Fogel con su programación evolutiva que estaban basados en máquinas de estados finitos (Fogel, Owens, y Walsh, 1966) y John Holland y sus trabajos sobre sistemas inductivos o de clasificadores (Holland, 1975).

Con el aumento en la capacidad del procesamiento de las computadoras y su facilidad de acceso, se ha permitido la implantación de varios métodos de aprendizaje ya sea usando redes neuronales (Rumelhart, McClelland, y PDP Research Group, 1986), sistemas difusos (Nguyen y Walker, 1997) y la fusión de ambos que da lugar a los sistemas neuro difusos. Las reglas de aprendizaje que usan estos sistemas se basan en métodos numéricos como el método del gradiente o por mínimos cuadrados. En la

actualidad, la computación evolutiva ha contribuido con algoritmos que pueden usarse en las redes neuronales y en los sistemas difusos, en lugar de los métodos convencionales.

Ahora revisaremos algunos problemas de los mecanismos de aprendizaje. El aprendizaje se caracteriza por asociar o relacionar un conjunto de situaciones con otro conjunto de situaciones; entonces podemos decir que en los mecanismos de aprendizaje generalmente se buscan relaciones entre situaciones en forma autónoma y se retienen a través del tiempo. Por ejemplo, si tenemos tres conjuntos de situaciones A, B y C entonces podemos asociar A con B por medio de un mecanismo de aprendizaje. La relación se puede llamar R_{AB}. Si se presentan situaciones del conjunto A, entonces podemos asociarlo al conjunto B por medio de la relación R_{AB}; sin embargo, si por alguna razón es necesario asociar las situaciones del conjunto A con las del conjunto C, la relación R_{AB} no será útil porque obtendremos situaciones del conjunto B ya que la relación así lo indica. Es necesario aplicar nuevamente el mecanismo de aprendizaje para obtener la relación funcional entre los conjuntos A y C; es decir, R_{AC}. En la mayoría de los mecanismos de aprendizaje, la relación funcional anterior se pierde; es decir, para aprender la relación R_{AC} se pierde la relación R_{AB}.

Continuando con el ejemplo, tenemos tres conjuntos de situaciones A, B y C con dos relaciones entre los conjuntos R_{AB} y R_{AC}. Si suponemos que no tenemos la relación R_{AC} y existe una similitud entre los conjuntos B y C entonces debe existir una similitud entre las relaciones R_{AB} y R_{AC}. Usando esta suposición, podemos usar la relación R_{AB} para encontrar la relación entre los conjuntos A y C, es decir R_{AC}. Al establecer una similitud entre los conjuntos B y C, estamos reutilizando la relación R_{AB} para encontrar R_{AC} evitando empezar sin información que generalmente es la forma en que funcionan los mecanismos de aprendizaje.

Tenemos dos propiedades deseables en un mecanismo de apendizaje, el primero es la retención de relaciones aprendidas con anterioridad, el segundo es la reutilización de relaciones aprendidas que podrían acelerar el mecanismo de aprendizaje. Existen mecanismos que permiten el aprendizaje de varias relaciones como son los sistemas propuestos por Grossberg basados en la teoría de resonancia adaptiva o ART (Freeman y Skapura, 1992). El mecanismos usado en el aprendizaje por analogía, también permite el aprendizaje de varias relaciones; además, las retiene (Carbonell, 1983).

Desde los orígenes de la teoría evolutiva, se ha propuesto la universalidad de la misma (Cziko, 1995), lo cual indica que no sólo da una explicación al origen de las especies, sino que puede explicar el origen del lenguaje, la propagación de las ideas y también el aprendizaje. Si usamos computación evolutiva en los mecanismos de aprendizaje, entonces es posible crear un sistema con propiedades de retención y reutilización.

Gould y Lewontin han propuesto el término *exaptación* para explicar porqué algunos organismos tienen estructuras adaptadas que originalmente tenían otros usos. También explica porqué existen muchas redundancias en las estructuras de los seres vivos sin ninguna utilidad aparente, y después en algún momento se utilizan para adaptarse a un uso específico (Gould y Vrba, 1982; Gould, 1991). La exaptación permite la retención de estructuras y además permite su reutilización en nuevas funciones similares o totalmente distintas a las que originalmente fueron adaptadas.

Dado que la computación evolutiva ha propuesto algoritmos evolutivos para resol-

ver problemas de optimización y aprendizaje, es interesante tenerlos con capacidades exaptivas ya que podrían retener estructuras útiles y reutilizarlas en el momento en que las condiciones del problema cambien.

1.2 Estructura general de la investigación

La investigación realizada en esta tesis muestra la forma de crear e implantar sistemas evolutivos con propiedades exaptivas. Un sistema exaptivo o con propiedades exaptivas reutiliza y retiene conocimiento para enriquecer su comportamiento. Un sistema exaptivo puede utilizarse como un mecanismo de aprendizaje o como un sistema de optimización de funciones dinámicas. Es por eso que se divide la tesis en tres partes. La primera parte es una revisión de temas relacionados con la computación evolutiva, el aprendizaje, los problemas dinámicos y la exaptación; dicha parte está dividida en cinco temas que aportan propiedades, características, ideas y algoritmos para la creación e implantación de sistemas exaptivos. El primer tema es la computación evolutiva, en donde se analizan distintos algoritmos que aplican los conceptos de la teoría evolutiva. Los algoritmos que se revisarán son las estrategias evolutivas, la programación genética y los algoritmos genéticos. Cada uno aporta ideas de implantación ya que han resuelto problemas de optimización y aprendizaje. El segundo tema es el aprendizaje. Es importante revisar las propiedades de un mecanismo de aprendizaje ya que permitirán determinar sus componentes con la finalidad de usarlos en un sistema exaptivo. El tercer tema es el aprendizaje por analogía. Se demostrará que existe una relación entre el aprendizaje por analogía y la exaptación ya que ambos retienen y reutilizan conocimiento. El cuarto tema se enfoca a los problemas dinámicos que se dividen en la optimización de funciones dinámicas y en el aprendizaje en un medio ambiente dinámico. Tanto la optimización como el aprendizaje se trabaja desde un punto de vista evolutivo. El quinto tema tiene dos propósitos. El primero es la explicación a detalle de la exaptación y el segundo es la posible implantación de un sistema exaptivo o con características exaptivas basándose en los tres primeros temas.

La segunda parte revisa la forma de reutilizar y retener soluciones en algoritmo evolutivos. Primero se analizan y clasifican las técnicas de sembrado en algoritmos genéticos ya que es un manera de reutilizar soluciones. Para los algoritmos que se basan en la información extraída de un conjunto de individuos, se explica y se prueba una forma de reutilización. Finalmente se revisa la forma de implantar sistemas con características exaptivas analizando los mecanismos que existen para reconocer cambios en un problema dinámico, retener soluciones y reutilizarlas. Los sistemas propuestos son sistemas seudoexaptivos porque tiene mecanismos explícitos de retención y reutilización. Los sistemas propuestos son puestos a prueba con otro algoritmo evolutivo de otro autor que se especializa en la optimización de funciones dinámicas.

La tercera parte está dedicada al aprendizaje. Se escogerá un sistema neuronal sencillo para aprender funciones no lineales y se usará en un sistema seudoexaptivo. El sistema mostrará dos cualidades deseables en un sistema exaptivo que es la retención de estructuras útiles y la reutilización para aprender nuevas funciones. Finalmente se

propone un agente con propiedades exaptivas para que se desempeñe adecuadamente en tareas de logística. Se demostrará que el sistema le brinda una ventaja sobre otro agente que no utiliza características exaptivas.

1.3 Preguntas clave de la investigación

El principal objetivo de la investigación es la creación e implantación de sistemas evolutivos exaptivos que son sistemas que puedan retener y reutilizar información para que puedan incrementar su desempeño. Para cumplir con el objetivo propuesto de esta tesis, es necesario contestar las siguientes preguntas:

- ¿Es posible crear un sistema exaptivo?
- ¿Cuál es la estructura de un sistema exaptivo?
- ¿Cuáles son sus elementos que lo componen y cuáles son sus funciones?
- ¿Qué tan factible es la implantación de un sistema exaptivo?
- ¿Realmente los sistemas con características exaptivas propuestos retiene y reutiliza conocimiento?
- ¿Es posible extender el sistema exaptivo para resolver problemas de optimización y de aprendizaje, sobre todo cuando las condiciones de los problemas cambian en el tiempo?
- ¿Cómo puede crearse un agente inteligente con propiedades exaptivas?

1.4 Contribuciones

La principal contribución de esta tesis es la creación e implantación de un sistema con características exaptivas para resolver problemas de optimización de funciones cambiantes y problemas de aprendizaje en un medio ambiente dinámico. El sistema exaptivo también puede usarse en la estructura de un agente inteligente, para que tenga más habilidades en la interacción con un medio ambiente cambiante e inestable.

Los problemas de optimización y aprendizaje que se proponen en la tesis tienen una finalidad ilustrativa, es decir, permitirán mostrar la forma de implantar un sistema seudoexaptivo que solucione estos problemas. Otra contribución importante es la extensión que pueden tener las ideas propuestas ya que es posible diseñar sistemas seudoexaptivos que solucionen problemas más complejos que los propuestos en la tesis. Los sistemas exaptivos pueden ser una opción muy conveniente cuando se tengan problemas que impliquen optimización de funciones semejantes o funciones que cambien en el tiempo, modificación de procesos en forma gradual, diseño de controladores autoadaptables, optimización y control de procesos que sean semejantes, aprendizaje

autónomo autoadaptable e incremental y aprendizaje multifuncional. Una contribución importante es la propuesta de procedimientos para la reutilización de una o varias soluciones de manera eficiente.

1.5 Estructura de la tesis

La tesis se divide en tres partes. La primera parte se compone de cinco capítulos. La introducción es el presente capítulo. La computación evolutiva es el título del capítulo 2. Los mecanismos de aprendizaje y el aprendizaje por analogía son los contenidos de los capítulos 3 y 4 respectivamente. El capítulo 5 se revisan los problemas dinámicos. El capítulo 6 explica el origen del término exaptación; también indica la forma de implantar sistemas con propiedades exaptivas.

En la segunda parte se revisa la reutilización del conocimiento en los algoritmos evolutivos en problemas de optimización. La segunda parte está dividida en cuatro capítulos. En el capítulo 7 se analizan las técnicas de sembrado en los algoritmos genéticos. El capítulo 8 explica la forma de reutilizar soluciones en la mutación dirigida (otro algoritmo evolutivo). El capítulo 9 muestra la implantación de dos sistemas seudoexaptivos para la optimización de una función dinámica.

La tercera y última parte muestra cómo se aplican los sistemas seudoexaptivos en problemas que requieren mecanismos de aprendizaje. En el capítulo 10 se revisan los problemas de aprendizaje de varias funciones con redes neuronales. El capítulo 11 se dedica a los agentes seudoexaptivos. Se explica la arquitectura de un agente seudoexaptivo que se usa para resolver problemas relacionados a la reprogramación de tareas. Las conclusiones y el trabajo futuro se incluyen en el capítulo 12.

Capítulo 3

Mecanismos de aprendizaje

3.1 Introducción

Siempre se ha trabajado con procesos para obtener algún beneficio; por ejemplo del proceso de la descomposición, podemos obtener productos lácteos, vino, cerveza, etc. Todos los procesos toman materias primas y dan productos. Si queremos tener una calidad en los productos, entonces es necesario tener un control en el proceso. Una forma de establecer un control es por medio de una política para el proceso, es decir, una forma de actuar en el proceso. Podemos decir que una política es un conjunto de acciones que permiten alcanzar un fin, un objetivo o una meta. Muchos procesos son manejados por sistemas autónomos o por operadores y ambos tienen en su "memoria" la política que debe aplicarse. El mecanismo para adquirir una política se le puede llamar aprendizaje.

La entidad que usa aprendizaje busca obtener una relación entre las acciones que aplica al proceso (para cumplir con los objetivos) y lo que se observa del proceso. La entidad debe aplicar ciertas acciones si los objetivos no se cumplen. Las observaciones realizadas a los procesos corresponde a sus estados. Podemos decir que el aprendizaje es un procedimiento que permite obtener las relaciones entre los estados que se observan del proceso y las acciones que son aplicadas. Si una política es una forma de actuar, entonces podemos decir que también es una relación. Podemos plantear algoritmos para obtener políticas o relaciones. Los algoritmos planteados son los mecanismos de aprendizaje.

3.1.1 Implantación de un algoritmo general de aprendizaje

Para establecer un mecanismo de aprendizaje se describen dos conjuntos E y A que representan respectivamente a los estados y las acciones posibles. R_F relaciona un conjunto de estados E con un conjunto de acciones A. R_F está formado por varios componentes que determinan su comportamiento. El mecanismo de aprendizaje determina los componentes adecuados para encontrar la relación funcional deseada. Para encontrar los componentes adecuados es necesario tener una referencia de lo deseado a lo que existe; por medio de esta referencia se modifica, elige o desecha un componente.

El siguiente ejemplo explica la formación de relaciones funcionales. Tenemos tres componentes C_1, C_2 y C_3; según el diagrama de la figura 3.1, el conjunto de las entradas E tiene dos subconjuntos E_1 y E_2. El conjunto de las acciones A también tiene dos subconjuntos A_1 y A_2. Para formar la relación funcional entre $E1 \rightarrow A_1$ usamos los componentes C_1 y C_2. Para la relación funcional $E_2 \rightarrow A_2$ usamos el componente C_3. Un algoritmo general de aprendizaje se muestra en la figura 3.2.

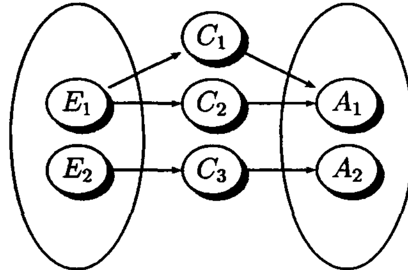


Figura 3.1: La relación funcional se forma estableciendo una relación entre los estados y las acciones con los componentes.

1) Lectura del estado $e(t)$.
2) Obtención de la acción $a(t)$ usando la relación funcional R_F.
3) Aplicación de la acción $a(t)$ al proceso.
4) Obtención del estado $e'(t+1)$ del proceso.
5) Se usa el estado $e'(t+1)$ para verificar el objetivo.
6) Si se cumple con el objetivo, entonces retener los componentes de R_F.
7) Si no cumple con el objetivo, la acción $a(t)$ no es la adecuada, entonces se busca otra acción modificando los componentes de la relación funcional R_F.
8) Si es fin de aprendizaje ir a (1), sino terminar.

La variable t corresponde al tiempo. R_F es la relación funcional que asocia los estados $e(t)$ con las acciones $a(t)$.

Figura 3.2: Algoritmo general de aprendizaje.

La función del algoritmo es aprender una política para un proceso que toma acciones y responde con estados que cumplan con un objetivo preestablecido. La guía que existe para encontrar las acciones adecuadas es el objetivo que tenga la entidad. Tenemos una entidad de aprendizaje que genera una relación funcional eligiendo, eliminando o modificando sus componentes. La relación entre los estados (e) y las acciones (a) se representa en la relación funcional R_F. El aprendizaje siempre se limita por el tiempo en que existe la entidad o por la capacidad de memoria que tenga.

3.1.2 Algunos ejemplos de relaciones funcionales

Los componentes de las relaciones funcionales pueden ser muy variados. Se pueden usar parámetros de ecuaciones, funciones, estructuras, reglas, conexiones, procedimientos, etc. Los siguientes ejemplos ilustrarán cómo se obtienen relaciones funcionales usando diversos componentes. El primer ejemplo es sobre la representación de una relación funcional como una ecuación matemática. Tenemos tres estados (argumentos de entradas) x_1, x_2 y x_3. Hay una sola acción (salida) $f(x_1, x_2, x_3)$. La relación funcional es como sigue:

$$f(x_1, x_2, x_3) = a\ \text{sen}(x_1) + b\cos(x_1\, x_2) + d\cos(x_3)$$

Los parámetros a, b, c y d modifican el comportamiento de la relación funcional. Las funciones sen(), cos(), + y $*$ también modifican el comportamiento de la relación funcional. La multiplicación la representaremos con $*$ como ilustración.

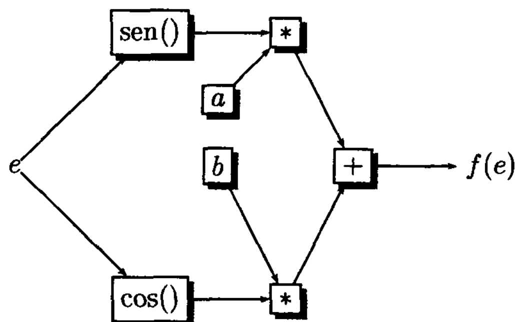


Figura 3.3: Una relación funcional representada como una ecuación matemática.

Una relación funcional con estructuras interconectadas es como sigue. Si representamos las funciones sen(), cos(), + y $*$ como estructuras o bloques de procesamiento y las conectamos entre sí, entonces podemos obtener relaciones funcionales muy complejas. La estructura de la figura 3.3 corresponde a la relación funcional $f(e) = a\text{sen}(e) + b\cos(e)$. Si cambiamos las conexiones, obtendremos una relación funcional totalmente distinta, como se observa en la figura 3.4 ya que corresponde a la ecuación $f(e) = a\, b\cos(\text{sen}(e) + e))$.

Otra forma de representar relaciones funcionales es por medio de reglas o procedimientos. Podemos decir que tanto las reglas como los procedimientos están compuestos por ecuaciones o funciones. Un ejemplo muy simple consiste en un sistema basado en

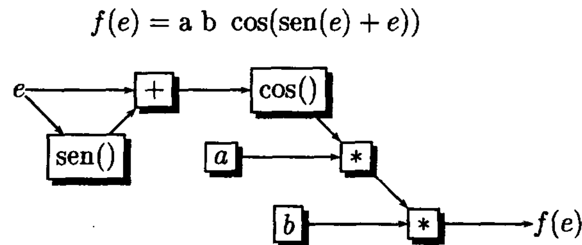


Figura 3.4: Una relación funcional representada como una ecuación matemática; mismos componentes, diferente estructura.

sólo dos reglas, que toma como argumento de entrada el estado de una variable e y a la salida el sistema brinda la acción a.

Usando reglas:
Si el estado e es alto entonces la acción a es baja
Si el estado e es bajo entonces la acción a es alta

Cuando se usan procedimientos, la acción a la obtenemos a partir del estado e como a = procedimiento$_B$(procedimiento$_A(e)$). En resumen, los mecanismos de aprendizaje buscan alterar los componentes de las relaciones funcionales para cumplir con un objetivo específico.

3.1.3 Aprendizaje supervisado y no supervisado

Existen dos formas para realizar el aprendizaje. El aprendizaje busca o modifica los componentes de una relación funcional para dar una acción adecuada. Si existe un tutor que brinde la acción correcta como referencia, entonces se dice que se está realizando un aprendizaje *supervisado* ya que la entidad de aprendizaje debe preocuparse solamente en buscar o modificar sus componentes. En cambio, cuando no existe un tutor que brinde la acción correcta, es necesario medir de alguna manera el efecto que tiene las acciones con el proceso. La finalidad es la modificación adecuada de sus componentes por medio de esta medida. Este tipo de aprendizaje se llama *no supervisado*. En los mecanismos de aprendizaje con algoritmos evolutivos, generalmente no se utiliza la supervisión, sin embargo puede usarse cuando sea requerida, sobre todo cuando se diseñan sistemas que imiten comportamientos.

3.2 Algoritmos evolutivos en los mecanismos de aprendizaje

En esta sección se explicará la forma de aplicar algoritmos evolutivos para obtener políticas o relaciones funcionales. El algoritmo evolutivo fue descrito en el capítulo anterior y tiene procesos de selección, evaluación y variación (reproducción). En la sección anterior obtuvimos la forma de representar relaciones funcionales y se observó la manera en que el mecanismo de aprendizaje las modifica para cumplir con un objetivo predeterminado. Podemos tener un algoritmo de aprendizaje evolutivo al aplicar los procedimientos evolutivos en la búsqueda o modificación de componentes de las relaciones funcionales (Holland, 1975). En la implantación de un algoritmo de aprendizaje evolutivo, se usan los procesos de evaluación, selección y variación; sin embargo, cada proceso debe cubrir las siguientes consideraciones:

1. El proceso de evaluación se aplica en las relaciones funcionales para tener un valor de aptitud; es decir, se determina el desempeño que tiene una relación funcional para cumplir con los objetivos establecidos. La evaluación es compleja ya que la representación del cromosoma requiere codificaciones muy complicadas (en algoritmos genéticos sobre todo). El proceso de evaluación inicia dejando a la entidad (de donde reside la relación funcional o política) funcionando en un lapso de tiempo predefinido; después se adquiere una medida de desempeño que es comparada con los requerimientos deseables y con la diferencia que exista, obtenemos el valor de evaluación. Es común que el proceso de evaluación se ejecute lentamente; por lo tanto, lo más adecuado es contar con modelos de procesos reales cuando sea necesario acelerar la evaluación.

2. El mecanismo de selección es prácticamente el mismo que el aplicado en otros algoritmos y no requiere de consideraciones especiales.

3. El mecanismo de variación básicamente permanece igual. Lo que es importante destacar es cómo afectan los mecanismos de variación a las estructuras y componentes de las relaciones funcionales ya que pueden encontrarse formaciones inválidas; sin embargo, si incluimos un mecanismo de validación o usamos una codificación o representación adecuada, evitamos estos problemas.

El algoritmo de aprendizaje evolutivo inicia con un aprendizaje no supervisado. En la implantación del algoritmo de aprendizaje evolutivo general, es necesario definir un conjunto de estados de un proceso e, un conjunto de acciones válidas que puedan aplicarse al proceso a y una relación funcional R_F donde $a = R_F(e, C)$ con los componentes C. La relación funcional interactúa con el proceso en un periodo de tiempo predeterminado, tomando sus estados e y dando las acciones a. El algoritmo de aprendizaje evolutivo se muestra en la figura 3.5.

En las siguientes secciones revisaremos tres mecanismos de aprendizaje que usan algoritmos evolutivos. Los sistemas adaptables de clasificadores, las redes neuro evolutivas y los sistemas difusos evolutivos.

1) Lectura de un estado del proceso $e(t)$.
2) Obtención de una acción $a(t)$ de la relación funcional $a(t) = R_F(e(t), C)$
3) Aplicación de la acción $a(t)$ al proceso.
4) El proceso reacciona a la acción $a(t)$ dando un estado $e'(t+1)$.
5) El estado $e'(t+1)$ se usa para verificar el cumplimiento del objetivo.
6) Si es fin de interacción entonces ir a (7) sino ir a (1).
7) Evaluación del desempeño de la relación funcional R_F en el proceso.
8) Modificación de los componentes C de la relación funcional
por medio de la evaluación.
9) Si no es fin de aprendizaje ir a (1), sino terminar.

Figura 3.5: Algoritmo de aprendizaje evolutivo.

3.3 Sistemas adaptables de clasificadores

Los sistemas adaptables de clasificadores (SACs) fueron creados por Holland con la intención de emular el aprendizaje como un procedimiento inductivo, es decir, el procedimiento de adquirir conocimiento (relaciones funcionales) por medio de ejemplos evaluados en el proceso (Holland, Holyoak, Nisbett, y Thagard, 1986; Holland, 1986). Los sistemas adaptables de clasificadores se componen principalmente por reglas de producción (si-entonces) o clasificadores y usualmente están codificadas en cadenas ternarias (0, 1, #). El aprendizaje gira en torno a los clasificadores.

3.3.1 Componentes de un sistema adaptable de clasificadores

El sistema adaptable de clasificadores tiene como componentes una lista de clasificadores, detectores de estados, efectores de acciones, mecanismo de administración de clasificadores y un algoritmo genético. La función principal de cada clasificador es detectar estados y dar acciones. Cada clasificador tiene un peso de utilidad. El peso depende de la evaluación o recompensa que tenga el sistema en el momento. Los detectores y efectores son los que tienen una interacción con el medio ambiente o el proceso. Los detectores tienen la función de codificar los estados a mensajes para los clasificadores. Los efectores toman los mensajes de los clasificadores que darán acciones y las decodifican para que puedan aplicarse al proceso.

El mecanismo de administración de clasificadores tiene dos funciones, la primera es la selección de clasificadores y la segunda es la actualización de pesos. La selección es necesaria para determinar el clasificador que dará la acción. En la actualización de pesos es usado un mecanismo para repartir la utilidad total recibida del exterior (por medio de la evaluación) entre todos los clasificadores que contribuyeron para lograr esta utilidad. El algoritmo del sistema adaptable de clasificadores se muestra en la figura 3.6.

El SAC tiene un algoritmo evolutivo para crear nuevos clasificadores y eliminar

1) Lectura de un estado por medio del detector.
2) Activación de varios clasificadores.
3) Selección del mejor clasificador por criterios de utilidad o
por probabilidades.
4) Establecimiento de la salida que propone el mejor clasificador por medio
del efector.
5) Se recibe ocasionalmente una evaluación de las acciones efectuadas.
6) Se determina la utilidad de cada clasificador, generalmente
basada en la evaluación.
7) Se aplica ocasionalmente un algoritmo genético.
8) Si no es fin de algoritmo, ir a (1) sino terminar.

Figura 3.6: Algoritmo general del sistema adaptable de clasificadores.

los de baja utilidad. Los nuevos clasificadores son creados a partir de los clasificadores con mayor grado de utilidad. Existen varios tipos de SACs, unos difieren por los tipos de clasificadores que usan, hay SACs que usan redes neuronales (Smith y Cribbs, 1994) o sistemas difusos (Valenzuela-Rendón, 1991). Otra diferencia que existe es la forma de repartir la utilidad a los clasificadores. Podemos colocar un algoritmo evolutivo encima de un sistema adaptable de clasificadores, para modificar las propiedades de un SACs tal como la forma de repartir la utilidad (Kawakami, 1996).

3.4 Redes Neuro evolutivas

Las redes neuronales son sistemas que simulan las estructuras del sistema nervioso. La unidad básica de estas estructuras son las neuronas. Una red neuronal artificial consiste en la interconexión de pequeñas unidades de procesamiento o neuronas artificiales.

Una neurona natural tiene miles de entradas llamadas dendritas. Todas las señales de entrada son integradas en el cuerpo de la neurona (soma) y si superan un umbral predeterminado, se obtiene una señal de salida, es decir, la neurona se activa. La señal de salida es trasladada a otras neuronas por medio del axón. Las neuronas se conectan unas con otras por medio de sinapsis (punto de transmisión entre el axón de una neurona y la dendrita de otra u otras); existen dos tipos de conexiones, una es por medio de una excitación; la otra forma de conexión es la inhibición.

El procesamiento de cada neurona se puede simular con una función no lineal con varios argumentos (entradas) y una salida. El grado de conexión entre una neurona u otra se realiza por medio de un peso. Si el peso de conexión toma un valor de cero, entonces no existe conexión, pero si toma un valor positivo o negativo, existirá una excitación o una inhibición, respectivamente (Haykin, 1999). Los componentes que influyen en el aprendizaje de una red neuronal son los pesos o grados de conexión, el tipo de neurona usada, la forma de conexión, la regla de aprendizaje y los parámetros de la regla de aprendizaje.

Un algoritmo evolutivo puede ser utilizado para modificar uno o más de estos componentes. El trabajo más complejo es la representación de los componentes de las redes neuronales. Cuando se modifican los pesos entonces se usan estructuras fijas (número fijo de neuronas, con las mismas características y conexiones, la regla de aprendizaje también permanece sin cambio) y pueden usarse algoritmos de programación evolutiva o las estrategias evolutivas (Saravanan y Fogel, 1995; Montana y Davis, 1989).

Cuando se modifican las estructuras, ya sea conexiones o el tipo de neurona usada (generalmente la regla de aprendizaje es la misma), entonces se presta atención a la codificación de las conexiones o de los tipos de neuronas ya que para este tipo de problemas son usados los algoritmos genéticos (Kitano, 1990). Por último tenemos algoritmos que modifican la regla de aprendizaje y la forma de conexión, es decir, si es excitación o inhibición (Mondada y Floreano, 1995).

Para simplificar la implantación, tenemos una red neuronal R_F con los parámetros que indican el tipo de neurona, la estructura que se deduce al número de neuronas por capa, el número de capas, la reglas de aprendizaje y los pesos. Los mecanismos de variación de un algoritmo evolutivo modifican estos componentes y los adapta para encontrar una relación funcional particular. La red neuronal tiene varios componentes para representar una relación funcional $R_f(t_n, e_s, r_a, p)$ donde t_n es el tipo de neurona. e_s es la estructura que se refleja en el número de neuronas por capa. r_n corresponde a la regla de aprendizaje. p son los pesos que determinan el grado de conexión.

Considerando que las mayúsculas son parámetros constantes y las minúsculas son parámetros variables tenemos que la relación funcional (red neuronal) $R_f(T_N, E_s, R_A, p)$ se puede modificar alterando los parámetros de los pesos p. Otra modificación consiste es la variación en la estructura de la red neuronal e_s y en el tipo de neurona t_n, así la relación funcional toma la forma $R_f(t_n, e_s, R_A, P)$ y finalmente tenemos la variación de la regla de aprendizaje: $R_f(T_N, E_s, r_a, P)$. El objetivo es aplicar el algoritmo evolutivo para encontrar el valor más adecuado para cada variable. Se toma el mismo algoritmo mostrado en la figura 3.5, pero sólo se usan los componentes de la red neuronal. Todos los demás procedimientos evolutivos permanecen igual.

3.5 Sistemas difusos evolutivos

Los sistemas difusos están formados por una serie de reglas expresadas en términos de conjuntos difusos. Los conjuntos difusos son discretizaciones del universo de estados. Cada estado de un proceso puede variar desde una magnitud muy baja hasta tener magnitudes muy altas. Por ejemplo, el estado de la temperatura de un proceso puede tener desde temperaturas muy bajas hasta temperaturas muy altas. Como se verá, no se manejan valores numéricos específicos tal como 33.6 grados centígrados sino términos difusos como temperatura baja, muy alta, etc. Con los términos difusos podemos crear reglas del tipo:

"Si la temperatura es alta entonces la válvula debe cerrarse mucho. "

Generalmente se producen señales con valores numéricos en los procesos y para poder usar las reglas difusas, es necesario convertir estos valores a términos difusos.

Los términos de baja, medio, muy alta, tienen una representación numérica que abarca discretizaciones de un universo en particular y dan como resultado un grado de verdad. A cada argumento o término difuso, le podemos asignar un conjunto de valores con un grado de verdad o membresía.

Si tenemos una temperatura x entonces podemos obtener un grado de verdad para baja temperatura, y otro para muy alta temperatura. Si definimos varios conjuntos difusos y con ellos formamos reglas entonces al tener varios grados de verdad tendremos varias reglas activas. Para tener una consecuencia concreta son aplicados mecanismos de inferencia similares a los aplicados en la lógica formal. La representación de un sistema difuso se reduce a un conjunto de ecuaciones tanto para los conjuntos difusos como para los mecanismos de inferencia (Nguyen y Walker, 1997). Un sistema difuso puede aproximarse a cualquier relación funcional (Mendel, 1995); entonces si modificamos los componentes de estos sistemas, tendremos distintas relaciones funcionales. Los componentes de un sistema difuso son los conjuntos difusos, las reglas y los mecanismos para la inferencia difusa.

Los conjuntos difusos son representados con funciones que toman como argumento un estado del proceso y dan a la salida un grado de verdad o membresía. Cada función tiene parámetros que determinan las características de un conjunto difuso, es decir, si es bajo, muy alto, medio, etc. Las reglas tienen antecedentes y consecuentes compuestos por uno de los distinto conjuntos difusos que pueden declararse.

Por ejemplo, tenemos la siguiente regla: si la temperatura es "muy alta" entonces la apertura de la válvula es "baja". Si el término difuso "muy alta" lo cambiamos por "media", entonces la regla cambia, si la temperatura es "media" entonces la apertura de la válvula es "baja". También tenemos varios mecanismos en un sistema difuso, como los desdifusificadores y los mecanismos de inferencia. El mecanismo de desdifusificación es necesario porque el resultado del sistema difuso debe de ser nítido, es decir, no podemos dar como salida el conjunto difuso "muy alto", sino un valor más concreto para que pueda ser útil.

Una relación funcional puede desarrollarse con sistemas difusos y para aplicar un algoritmo de aprendizaje evolutivo es necesario definir sus componentes. El sistema difuso tiene reglas r, parámetros de los conjuntos difusos c_d, mecanismos de inferencia m_t y mecanismos de desdifusificación m_d. La relación funcional con sus componentes toma la siguiente forma: $R_f(r, c_d, m_t, m_d)$. Las variaciones pueden existir en las reglas $R_f(r, C_D, M_T, M_D)$, variación de los parámetros de los conjuntos difusos $R_f(R, c_d, M_T, M_D)$ variación de los mecanismos de inferencia: $R_f(R, C_D, m_t, M_D)$ y variación de los mecanismos de desdifusificación $R_f(R, C_D, M_T, m_d)$. Es posible realizar variaciones a más de un componente a la vez.

3.6 Conclusiones

El capítulo muestra al mecanismo de aprendizaje desde un punto de vista general. Es posible formar una relación funcional con infinidad de estructuras (reglas, funciones, procedimientos) que son modificadas por medio del aprendizaje. Se estableció una

forma de aprendizaje usando computación evolutiva ya que de alguna forma existe una modificación guiada de estructuras de una relación funcional. Se estableció un algoritmo genérico de aprendizaje evolutivo. Finalmente se mostraron tres sistemas que usan alguna forma de aprendizaje evolutivo. Ya tenemos dos herramientas que son básicas para tener mecanismos de aprendizaje o para optimizar una función. Cada una de estos mecanismos de aprendizaje evolutivo permiten implantar sistemas con un alto grado de autonomía; sin embargo, en todos los sistemas existe el riesgo de empezar con relaciones funcionales aleatorias que sean ineficientes y dañinas para la aplicación en donde se requieren. El uso del conocimiento previo y de modelos que emulan el ambiente en que funcionarán, permitirán tener sistemas más estables y confiables.

SISTEMAS EXAPTIVOS: RETENCIÓN Y REUTILIZACIÓN DE CONOCIMIENTO EN ALGORITMOS EVOLUTIVOS

por

M. C. Luis Martín Torres Treviño

Tesis

Presentada al Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

del

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey

como requisito parcial para obtener el grado académico de

Doctor en Ciencias

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Monterrey, N.L. Mayo de 2004

Capítulo 4

Aprendizaje por analogía

4.1 Introducción

Una de las funciones de los seres vivos que se han querido emular, es la adquisición de nuevo conocimiento. Si consideramos un conocimiento particular como una relación funcional, entonces se desea un mecanismo de aprendizaje que pueda adquirir nuevas relaciones funcionales sin olvidar lo aprendido previamente. Los mecanismos de aprendizaje vistos en la sección 1 del capítulo 3, sólo permiten la adquisición de una relación funcional a la vez. El aprendizaje de otra relación funcional implica olvidar la anterior. Como se analizó en el capítulo anterior, una relación funcional requiere de una estructura. Esto implica una estructura por cada relación funcional. Si requerimos aprender más relaciones funcionales es necesario una estructura nueva por cada nueva relación funcional. El aprendizaje por analogía permite que una entidad adquiera nuevas relaciones funcionales en una misma estructura.

Para encontrar la solución a un problema o satisfacer los objetivos planteados sobre un proceso, es necesaria una política o relación funcional. Esta relación funcional es obtenida por medio de un mecanismo de aprendizaje. Si el problema o proceso cambia, entonces es necesario adaptar la relación funcional para poder seguir manteniendo las condiciones de satisfacción, es decir, que la relación funcional siga siendo válida. Si el proceso cambia mucho, es necesario tener una nueva relación funcional. El aprendizaje por analogía permite aprovechar las relaciones funcionales encontradas en otros procesos para adaptarlos en nuevos procesos similares o análogos a los anteriores. Para definir el procedimiento, es necesario una memoria con descripciones de problemas o procesos con su correspondiente relación funcional. Cuando se tiene un problema o proceso nuevo, es necesario establecer una similitud con alguna descripción de la memoria. Si existe una similitud, entonces se usa la relación funcional que le corresponde, adaptándola al nuevo problema. En el caso de no existir alguna similitud, entonces es necesario encontrar una relación funcional con algún mecanismo de aprendizaje.

4.2 Métodos del aprendizaje por analogía

De acuerdo a la sección anterior, para implantar el aprendizaje por analogía son necesarios varios componentes y mecanismos. Primero es necesaria una memoria para retener relaciones funcionales con su descripción. Esta memoria puede ser una estructura más compleja. Los demás componentes son los mecanismos de aprendizaje de relaciones funcionales, los mecanismos de adaptación o modificación de relaciones funcionales, los mecanismos de identificación de similitudes y finalmente el mecanismo de almacenamiento.

El procedimiento general del aprendizaje por analogía empieza con la lectura de una nueva descripción; se establece una similitud entre las descripciones de la memoria y la nueva descripción. Si existe una similitud, entonces se extrae la relación funcional de la memoria y la adapta para formar un nueva relación funcional. Cuando no existe una similitud entonces se aplica un mecanismo de aprendizaje para encontrar una nueva relación funcional. El ciclo del aprendizaje por analogía termina cuando se almacenan en la memoria la descripción del nuevo problema con la relación funcional aprendida. Cada uno de los procesos merece revisarse por separado.

4.2.1 Descripción y detección de similitudes

Se puede realizar la descripción de un problema o proceso de varias maneras, ya sea usando una descripción con palabras binarias, descripciones textuales o de tipo numérico. La detección de similitudes es un procedimiento más complicado. El problema radica en determinar la similitud entre dos procesos. Una forma consiste en identificar las características de un proceso y verificarlas con las características de otro proceso. La forma de representación mencionada se usa en el razonamiento basado en casos que revisaremos en la siguiente sección.

Otra forma de representación consiste en usar las acciones aplicadas a un proceso, tomar sus estados y relacionarlos con la descripción del proceso. Podemos crear relaciones funcionales entre las entradas y las acciones de un proceso con su descripción (Thrun y O'Sullivan, 1995; Thrun, 1995). Generalmente la descripción se traduce a una medida cuantitativa. Thron ha propuesto una forma de establecer similitudes entre problemas; lo que hace es generar categorías de tareas que permitan solucionar algunos problemas. Cada categoría resuelve un grupo particular de problemas que son similares entre sí.

El algoritmo considera a un grupo de tareas n, y un grupo de conocimiento m. Cada tarea n puede ser resuelta usando algún conocimiento m. Se realiza un agrupamiento en categorías de las tareas que tengan un conocimiento común (que todas las tareas pueden ser resueltas con el mismo conocimiento); si se tiene una nueva tarea, entonces se determina la categoría de tarea a la que pertenece; es decir, se verifica la existencia de alguna similitud. Si existe similitud, entonces se usa el conocimiento que está ligado a la categoría de tareas.

Como ejemplo, tenemos tres tareas t_1, t_2 y t_3 y dos grupos de conocimiento c_1, c_2. Evaluamos cada tarea en cada grupo de conocimiento. Si el conocimiento resuelve la

tarea, entonces se asigna una alta calificación (0.8) de lo contrario, se asigna una baja calificación (0.2). Para todas las tareas, obtenemos la siguiente tabla:

Tarea	C_1	C_2
t_1	0.2	0.8
t_2	0.2	0.8
t_3	0.8	0.1

La tabla nos indica que las tareas t_1 y t_2 pertenecen a una misma categoría ya que obtienen una alta calificación en el conocimiento C_2. La tarea t_3 no pertenece a la categoría porque el conocimiento C_2 no la resuelve eficientemente, es decir, recibe baja calificación. En cambio, el conocimiento C_1 resuelve la tarea t_3 por lo cual pertenece a otra categoría. Si tenemos una nueva tarea t_4 y es similar a las tareas t_1 y t_2, entonces pertenece a la categoría formada por estas tareas. Podemos usar el conocimiento que le corresponde a la categoría, es decir, C_2. No es necesario calificar la tarea ya que esto implicaría resolverla con cada uno de los conocimientos, lo cual equivale a empezar sin ninguna información. Cuando usamos categorías de problemas, podemos determinar de antemano el grupo de conocimiento más apropiado para el problema sin tener que revisarlos todos.

4.2.2 Modificación de soluciones

La modificación se puede realizar de distintas maneras, una es la siguiente. Si la relación funcional consiste en una secuencia de acciones, entonces es posible aplicar operadores que permitan alterar esta secuencia. J. G. Carbonell describió una serie de procedimientos que son útiles en el aprendizaje por analogía. Para aplicar los procedimientos es necesario describir los siguientes componentes. Tenemos un conjunto de posibles estados, un estado es designado como estado inicial; uno o más estados son estados meta o finales; un conjunto de operadores que permiten desplazarse de un estado a otro; una función que calcula la diferencia entre el estado actual y el estado final o meta; un conjunto de restricciones globales que deben satisfacerse para dar soluciones viables. Es posible la aplicación de un proceso llamado *Means End Analysis* (MEA). Los pasos del procedimiento MEA implican la búsqueda de una secuencia de operadores donde es posible alcanzar el estado final o meta a partir del estado inicial. Para aplicar el aprendizaje por analogía se requiere la recuperación de un problema previamente resuelto y que sea similar a uno nuevo. También se requiere de la transformación de la solución (compuesta por una secuencia de acciones) en una que satisfaga los objetivos del nuevo problema (Carbonell, 1983).

La solución de un problema anterior puede ser usada como punto de partida de un nuevo problema que es similar al anterior. Como la solución consiste en una secuencia de acciones, existen operadores para modificar esta secuencia. Los operadores son llamados T (de transformación) y su función es alcanzar el estado meta o final desde cualquier otro propuesto por la solución del problema anterior. Los operadores T alteran la solución por medio de inserciones o eliminaciones de operadores de la secuencia. Otros

operadores unen una solución como parte de una secuencia mayor; también es posible el uso del procedimiento MEA al principio o al final de la secuencia. Existe un operador T que invierte las acciones de las secuencias en forma parcial o total. Finalmente existe el operador T que realiza un reordenamiento de las acciones de la secuencia. Todos los operadores pueden aplicarse evitando secuencias inválidas, es decir, que violen las restricciones. El objetivo es reducir las diferencias entre el estado actual y el estado meta. Cuando la relación funcional no esté compuesta por una secuencia de acciones, entonces podemos modificarla alterando sus componentes. Lo anterior indica que es posible aplicar un mecanismo de aprendizaje de cualquier índole (capítulo 3).

4.2.3 Almacenamiento

El almacenamiento de descripciones con sus respectivas relaciones funcionales (soluciones) puede ser una impresión en la memoria en donde quedará en forma indefinida. El problema que conlleva un funcionamiento así es el uso excesivo de memoria porque cada vez que sea necesario el almacenamiento de una decripción con su relación funcional, será necesario reservar un nuevo espacio.

Otra forma de almacenamiento consiste en la administración de los espacios de memoria de forma que los espacios que contengan información de poco uso, tenderán a ser ocupados por información nueva. La información de la memoria que tenga mucho uso difícilmente cederá su espacio a nueva información. Por medio de un mecanismo administrativo se puede tener un uso eficiente de los espacios de la memoria.

4.3 Razonamiento basado en casos

La aplicación más común del aprendizaje por analogía es el razonamiento basado en casos. Un caso es una sugerencia para encontrar soluciones a un problema; un caso brinda formas para entender una situación. Los casos ayudan a resolver problemas mostrando sus partes importantes y resalta sus puntos relevantes. Un caso está compuesto de una descripción del problema y tiene ligado la solución o la forma para obtenerla. Los casos son de muchas formas y tamaños, cubriendo grandes o cortos lapsos de tiempo, asociando soluciones con problemas y situaciones.

El razonamiento basado en casos adapta una solución anterior para cumplir con las nuevas demandas (Watson, 1997; Kolodner, 1993; Christopher K. Riesbeck, 1989). Usa casos anteriores para explicar nuevas situaciones o para criticarlas. También aplica un mecanismo de razonamiento sobre casos precedentes para interpretar una nueva situación; finalmente crea una solución para el nuevo problema. El mecanismo para entender un nuevo problema requiere recordar casos anteriores e interpretar la nueva situación en términos de esos casos. La localización de un problema, significa encontrar en memoria la experiencia más cercana a la nueva situación. La adaptación es el proceso de arreglar una solución que fue usada en problemas anteriores. Se tienen cuatro procesos generales en el razonamiento basado en casos, el primero es la recuperación de los casos que sean similares; el segundo es la reutilización de la información y conoci-

miento de un caso que resuelva el problema actual; en el tercero se realiza una revisión de la solución propuesta; en el cuarto se retiene la solución del caso reutilizado para futuras referencias.

En la implantación del razonamiento basado en casos (figura 4.1), existe una base de datos que contienen varios casos. Cada caso tiene ligada su solución. La solución puede ser una secuencia de acciones o relaciones funcionales. Los casos tienen un índice de acceso que facilita la recuperación.

1) Lectura de un caso nuevo.
2) Recuperación de un caso de la base de datos que es similar al caso nuevo.
3) La solución es extraída.
4) La solución es modificada para el nuevo caso.
5) La solución es revisada para verificar que es la adecuada para el nuevo caso.
6) El caso nuevo es almacenado junto con su solución, previamente modificada y revisada.

Figura 4.1: Algoritmo general del razonamiento basado en casos.

El razonamiento basado en casos se ha implantado en infinidad de aplicaciones (Leake, 1996) y es una buena alternativa para generar sistemas con aprendizaje incremental, es decir, que su conocimiento crece conforme el sistema se use.

4.4 Sistema adaptable de clasificadores con memoria

Zhou (1985) incluyó una memoria de largo plazo al sistema adaptable de clasificadores para aprender muchas tareas. El sistema es útil cuando el dominio de aprendizaje es amplio ya que es necesario aprender muchas tareas o los cambios en el medio ambiente son ligeros o cuando es importante tener alta eficiencia en un mismo sistema cuando se tienen tareas similares. Para implantar estos sistemas, se requiere de una descripción de cada tarea. Los componentes de un SAC con memoria son:

1. Memoria de episodios que almacena descripciones de tareas.
2. Una base de datos con clasificadores.
3. Un sistema adaptable de clasificadores.

El funcionamiento general del SAC (sin memoria) se describió en el capítulo 3, sección 3. Los procedimientos del SAC con memoria consisten en un reconocedor de tareas en donde se aplican las siguientes funciones. Se realiza la lectura de alguna tarea. Si la tarea es similar, el SAC con memoria empieza con clasificadores de la base de datos, en caso contrario, empieza con clasificadores aleatorios. El SAC con memoria

tiene un mecanismo de almacenamiento de los mejores clasificadores con su respectiva descripción. El algoritmo se muestra en la figura 4.2.

1) Se adquiere la descripción de una tarea nueva para encontrar una solución.
La solución generalmente es una relación funcional.
2) Se determina una similitud con las descripciones de las tareas guardadas en la memoria. Cada descripción tiene ligada un conjunto de clasificadores.
3) Si existe en la memoria una descripción similar a la descripción de la tarea nueva, entonces se usa el conjunto de clasificadores ligados a la descripción similar.
4) Si no existe una similitud, entonces el SAC empieza con un conjunto de clasificadores aleatorios.
5) Se almacena la descripción de la tarea nueva así como el conjunto de clasificadores óptimos que resulte del desempeño del SAC.

Figura 4.2: Algoritmo general del sistema adaptable de clasificadores con memoria.

El SAC con memoria encuentra un conjunto de clasificadores óptimos que corresponden a la relación funcional o política que soluciona la nueva tarea. Tanto el conjunto de clasificadores óptimos como la descripción, se almacenan en la base de clasificadores y en la memoria de episodios respectivamente. El SAC ha sido usado para retener rutas óptimas en agentes móviles (Zhou y Grefenstette, 1989).

4.5 Redes neuronales de aprendizaje incremental

Las redes neuronales artificiales representan relaciones funcionales usando patrones o ejemplos de entradas con sus salidas; sin embargo, las redes neuronales que usan mecanismos basados en el gradiente son estructuras estáticas porque cuando es necesario modificar la relación funcional o se requiere añadir más conocimiento, el mecanismo de aprendizaje se repite sin aprovechar el conocimiento previo. Revisaremos dos redes neuronales que permitan aprender nuevo conocimiento de forma incremental, sin necesidad de que eliminen el conocimiento previo, sino que lo aprovechan.

Adaptive resonance theory

ART es una red neuronal de aprendizaje incremental. ART significa Adaptive Resonance Theory que es un modelo desarrollado por Grossberg y Carpenter (Carpenter y Grossberg, 1987a, 1987b, 1990) para representar la forma como los humanos reconocen información e incorporan nuevo conocimiento de forma incremental, es decir, sin olvidar el conocimiento previo. Las primeras versiones del ART son dos, una es el ART-1 que es una red neuronal que reconoce patrones de números binarios. La otra versión es ART-2 que es una red neuronal que reconoce patrones con números reales. Se han desarrollado otras redes neuronales que se basan en el mismo principio de la teoría

de resonancia adaptiva como ARTMAP (Carpenter, Grossberg, y Reynolds, 1991) y FUZZY ART (Carpenter, Grossberg, y Rosen, 1991).

ART trabaja con categorías. Definimos a una categoría como el espacio ocupado por un conjunto de entradas que comparten ciertas similitudes. ART le asigna a cada entrada una categoría; sin embargo, en caso de no existir alguna similitud de la entrada con las categorías existentes, se genera una nueva categoría. Las categorías tienen un tamaño específico que es controlado por medio de un parámetro que también determina el número de categorías generadas en la red neuronal. ART es una red neuronal de aprendizaje incremental, esto quiere decir que puede aprender nuevo conocimiento sin necesidad de olvidar el anterior, por lo cual, usa el conocimiento previo. Cuando una red neuronal aprende nueva información, reteniendo lo previamente aprendido, se dice que resuelve el dilema de la estabilidad-plasticidad.

Red neuronal de respuesta por máxima sensibilidad

La red neuronal de respuesta por máxima sensibilidad es similar a ART, porque trabaja con el principio de la máxima sensibilidad que existe entre las estructuras neuronales de los seres vivos cuando son estimuladas por una misma señal. Las neuronas que son más sensibles al estímulo, siempre son las más activas y tienen una mayor influencia en la respuesta de la estructura neuronal (Torres-Treviño, 1998). La red neuronal de respuesta por máxima sensibilidad tiene los siguientes procedimientos. Primero existe un mecanismo para la detección de entradas, luego se distribuye la señal de entrada a otras neuronas. La neurona más sensible es la que tendrá un grado alto de activación. Para que una neurona brinde su respuesta deberá superar un margen de sensibilidad. Si no existe alguna neurona que supere el margen (que es común cuando es una señal de entrada nueva), entonces todas las neuronas contribuyen para dar una respuesta; además, se asigna una nueva neurona para que aprenda la nueva señal de entrada. Cada neurona tiene un factor de sensibilidad que determina la sensibilidad de una neurona respecto a un conjunto específico de entradas. El factor de sensibilidad es el equivalente al parámetro de control del tamaño de la categoría usada por ART. Tanto ART como la red neuronal de respuesta por máxima sensibilidad son redes neuronales que utilizan las relaciones funcionales previamente aprendidas para agregar nuevo conocimiento sin eliminar el previo. Esta particularidad es importante ya que podemos incluir varias relaciones funcionales teniendo como límite a la memoria.

4.6 Conclusiones

Por medio de los mecanismos y sistemas basados en el aprendizaje por analogía se han logrado crear sistemas que retienen y reutilizan el conocimiento. Las herramientas que se han usado son básicamente para administrar la memoria de los sistemas, identificar similitudes o reconocerlas y modificar soluciones previas para adaptarlas a nuevos problemas. Todos los pasos son necesarios para reutilizar y retener soluciones. El cambio a lo que existe con lo que se propone es la forma de retener y reutilizar soluciones ya que

los sistemas con características exaptivas que se proponen, se basa en algoritmos evolutivos. En los siguientes capítulos se analizarán mecanismos para reutilizar y retener soluciones.

Capítulo 5

Solución de problemas dinámicos

Recientemente ha surgido un interés por resolver problemas dinámicos porque existen problemas reales que generalmente cambian en el tiempo. Los problemas dinámicos se pueden dividir en dos ramas, la optimización de funciones dinámicas y el aprendizaje en un medio ambiente dinámico. En el presente capítulo se realiza una revisión general de los problemas dinámicos y los algoritmos que se han propuesto para resolverlos. Existen diversos algoritmos, arquitecturas y métodos para resolver estos problemas, sin embargo, nos interesan los algoritmos basados en la computación evolutiva.

5.1 Optimización de funciones dinámicas

Muchos de los problemas reales son dinámicos, es decir, sus características, restricciones y variables pueden cambiar en el tiempo. Cuando un problema cambia a otro puede existir alguna relación entre ellos, es decir puede existir alguna dependencia entre ambos problemas que permita reutilizar la solución que se usó en el primer problema antes del cambio al segundo problema. Cada problema tiene asociado un conjunto de variables, un conjunto de restricciones y una función objetivo. Cada variable tiene una dimensión predefinida y puede cambiar en un periodo de tiempo corto. El número de variables generalmente permanece sin cambio por largos periodos de tiempo. Las restricciones pueden cambiar en periodos cortos de tiempo. La función objetivo tiene asociado un modelo que se liga al número de variables y restricciones del problema. Si cambia el número de variables, sus dimensiones así como las restricciones, entonces existe un cambio en la función objetivo. El grado de cambio en la función objetivo puede presentarse a traves del tiempo.

Considerando que $f_o(t_1)$ y $f_o(t_2)$ es la función de optimización en el tiempo t_1 y t_2 respectivamente, entonces si $|f_o(t_1) - f_o(t_2)| < d$ donde d es la distancia máxima de la vecindad y $t_1 < t_2$, entonces el cambio en la función objetivo f_o es ligero y por lo tanto, puede considerarse a ambas funciones como vecindades.

Las formas de cambio que pueden presentarse depende mucho de la forma de cambio de las variables, de las restricciones así como del objetivo que se refleja en la función de optimización. Si existe un cambio dentro de una vecindad d entonces podría

ser posible la reutilización de la solución que se usó antes del cambio. Si está lejos de la vecindad entonces la solución no podría ser muy útil y por lo tanto es necesario buscar una nueva solución.

Otra forma de cambio que puede presentarse de una manera repetitiva o cíclica, por lo cual con una memoria se podrían almacenar las soluciones que se usen en cada estado del ciclo. Si los cambios son muy bruscos entonces el problema se vuelve muy complejo. Se tiene interés en aquellas funciones que tengan un cambio cíclico complejo e inclusive caótico para obtener mejores soluciones en cada momento y en un tiempo corto.

Los algoritmos evolutivos tienen la tendencia de converger al óptimo (o al menos a una solución muy cercana del óptimo), por lo cual todos los individuos de la población son muy semejantes entre sí. Cuando el problema cambia es difícil encontrar otra solución porque se converge en forma prematura y la solución puede ser de baja calidad. Esto es debido a la poca diversidad que existe en la población del algoritmo evolutivo, por lo cual se hace necesario preservar la diversidad del algoritmo. Se han sugerido diversos algoritmos evolutivos que mantienen una alta diversidad en su población. Para los diversos algoritmos que buscan preservar la diversidad, se ha sugerido la siguiente clasificación (Branke, 1999a, 2001a):

- Algoritmos evolutivos que evitan converger por medio de poblaciones dispersas; suelen usarse métodos de compartición (*sharing*) (Goldberg, 1989) y edad en los individuos.

- Algoritmos que detectan un cambio en el ambiente y explícitamente se toman acciones para incrementar la diversidad. Se usan técnicas de sembrado en donde se incluyen variaciones locales de soluciones, mutaciones, etc.

- Algoritmos que tienen una memoria. Hay dos tipos de memoria explícita e implícita. En la memoria explícita se almacenan algunos individuos para no perderlos en caso de que el problema cambie. En la memoria implícita se usan representaciones redundantes, diploides o poliploides.

Otra clasificación ha sido sugerido por Trojanowsky y Michalewitz en donde no solo se clasifican los algoritmos evolutivos sino también los tipos de ambientes dinámicos que pueden presentarse (Trojanowski y Michalewicz, 1999); según los autores existen en un ambiente problemas con funciones objetivo y restricciones estáticas y dinámicas. La combinación de funciones objetivo estáticas, dinámicas con las restricciones que pueden ser nulas, estáticas o dinámicas dan un total de seis posibles ambientes en que se puede encontrar un algoritmo evolutivo.

La clasificación de los algoritmos evolutivos para resolver problemas dinámicos que sugieren los autores son:

- Algoritmos que mantienen un nivel de diversidad en su población. Se pueden seleccionar técnicas de compartición (*sharing, crowding*), métodos basados en la temperatura y entropía, edad en los individuos, técnicas de sembrado de soluciones aleatorias y/o como resultado de algún método de búsqueda local.

- Algoritmos que usan mecanismos de adaptación y autoadaptación que implica un ajuste dinámico de los parámetros del algoritmo evolutivo ya sea en base a alguna regla heurística o estática. Cuando se modifican los parámetros en forma paralela con el algoritmo evolutivo se le llama autoadaptación. Las estrategias evolutivas usan frecuentemente las técnicas de autoadaptación.

- Algoritmos evolutivos que usan material redundante. Para adaptarse al cambio pueden usarse un razonamiento sobre la experiencia previa; por medio de una memoria se puede mantener y recolectar esta experiencia. Existen tres tipos de memoria, numérica, simbólica y exacta.

  1. La memoria numérica implica almacenar los parámetros del algoritmo evolutivo y reutilizarlos si lo amerita la situación actual.
  2. La memoria simbólica implica encontrar alguna relevancia o conocimiento que exista entre los esquemas de los individuos. El conocimiento es almacenado en forma de reglas que se usan para guiar la búsqueda.
  3. En la memoria exacta se incluyen genes adicionales a la representacion (diploides, poliploides) solamente unos genes permanecen activos dependiendo del problema y de acuerdo a algunas funciones de dominancia.

5.2 Aprendizaje en un medio ambiente dinámico

Existe una diversidad de métodos y arquitecturas de aprendizaje que tratan con información simbólica o se inspiran en la naturaleza. En las tareas de aprendizaje se asume un medio ambiente invariante porque en sí el apredizaje es un problema complejo, sin embargo en el momento de escalar a problemas reales muchos mecanismos empiezan a fallar. A pesar de lo anterior existen algunos mecanismos de aprendizaje que permiten ciertan dinamicidad en el problema como el aprendizaje por recompensa (Sutton, 1998; Barto, Bradtke, y Singh, 1993), las redes neuronales (Waugh, 1994), los sistemas adaptables de clasificadores (Zhou y Grefenstette, 1989) y algunos mecanismos basados en en aprendizaje simbólico.

Para plantear el problema de la dinamicidad en el medio ambiente es necesario establecer el mecanismo del aprendizaje en un forma genérica. Si establecemos el aprendizaje como el problema de encontrar una relación funcional para aproximar funciones entonces definimos a una relación funcional como R_F con componentes C, entradas x y salidas y, es decir, $y = R_F(x, C)$. Se considera que el valor deseado en la salida y_d permanece fijo en un intervalo largo de tiempo t, entonces establecemos una medida sobre el desempeño de la entidad de aprendizaje, $D = f_d(y_d, y) = f_d(y_d, R_F(x, C))$ donde f_d es una función que determina o evalúa el desempeño comparando la salida actual de la entidad de aprendizaje con la salida deseada. Por ejemplo, en problemas de control se aplica una evaluación basada en el error entre la salida del sistema controlado y con la salida deseada en el sistema y_d. El objetivo de un mecanismo de aprendizaje para la aproximacion de funciones es minimizar la función D modificando los parámetros C. El

aprendizaje en un medio ambiente dinámico implica que las salidas deseadas o los criterios de construcción de la función f_d cambien en el tiempo, es decir, $D = f_d(y_d(t), y)$. La función $y_d(t) = f_d(x, \eta, t)$ suele llamarse función de supervisión, donde x son las posibles entradas, η es un factor de ruido y t es el tiempo; en todo caso la función indica que las salidas deseadas dependen del tiempo.

Generalmente existen grandes intervalos de tiempo que permiten que las salidas deseadas y_d permanezcan constantes a cada valor x, siendo influidas por el factor ruido, sin embargo, en algún instante de tiempo las salidas deseadas cambian, aún siendo las mismas entradas y es donde se hace necesario de un mecanismo de aprendizaje que ajuste rápidamente sus componentes para mantener la función D en un valor mínimo. Tenemos $D = f_d(y_d(x, \eta, t), R_F(x, c))$, ahora consideremos lo siguiente, si $R_F(x, C_1)$ minimiza la función D en un instante de tiempo dado t_1, tenemos: $D = f_d(y_d(x, \eta, t_1), R_F(x, C_1)) \leq \delta$ donde δ es un parámetro de eficiencia e indica que si el valor D o la función f_d es menor que δ entonces la relación R_F o más bien sus componentes C_1 son los adecuados y han creado la relación funcional correcta.

En un momento dado, en un tiempo t_2 existe un cambio en la salida deseada entonces pueden suceder dos casos. Lo primero es que se siga dentro de la vecindad δ lo cual es una cualidad importante en los mecanismos de aprendizaje ya que al presentarse nuevos casos no vistos en el entrenamiento, la entidad generaliza adecuadamente o si existe un factor de ruido importante, la misma entidad tiene la suficiente robustez para mantenerse dentro del margen aceptable. En el segundo caso es necesario definir una vecindad δ_2 donde $\delta_2 > \delta$ y de alguna manera se determina como la magnitud de cambio que tiene la función y_d en el tiempo de t_1 a t_2. Como $|y_d(x, \eta, t_1) - y_d(x, \eta, t_2)| < \delta_2$, entonces existe una similitud entre $y_d(t_1)$ y $y_d(t_2)$, trasladando la relación funcional, $y_d(t_2)$ es casi igual a $R_F(x, C_1))$, tenemos $|y_d(x, \eta, t_2) - R_F(x, C_1)| < \delta_2$ esto indica que la relación funcional C_1 puede ser útil para encontrar un conjunto de componentes C_2 por la similitud que existe entre la salida deseada en el tiempo t_1 y el tiempo t_2 y entonces podría ser útil usar C_1 como referencia para encontrar C_2, es decir, el mecanismo de aprendizaje modificaría los componentes C_1 para llegar a los componentes C_2. En caso de estar fuera de la vecindad δ_2 implica aplicar un mecanismo de aprendizaje para encontrar C_2 de tal forma que se encuentre dentro de la vecindad δ. Cuando la función cambia de forma cíclica o repetitiva, sin permanecer dentro de la distancia δ_2 entonces es necesario recurrir a algún mecanismo de memoria o de retención para facilitar el retorno de los componentes que fueron útiles en el pasado y reutilizarlos; en otras palabras, se desean las propiedades de retención y reutilización en los mecanismos de aprendizaje que actúan en un medio ambiente dinámico.

5.3 Conclusiones

En el capítulo se plantearon dos tipos de problemas que existen cuando se interactúa con medios ambientes dinámicos. Cuando se trata de optimizar funciones dinámicas con algoritmos evolutivos se hace necesario mantener la diversidad de la población. El aprendizaje en un ambiente dinámico tambien requiere métodos para retener y reutilizar

relaciones funcionales que conlleva de alguna manera a un problema de mantenimiento de diversidad, siempre y cuando se use computación evolutiva en el mecanismo de aprendizaje. En los capítulos siguientes se buscará la manera de resolver problemas dinámicos tanto de optimización como de aprendizaje.

Capítulo 2

La computación evolutiva

2.1 Los procesos evolutivos

Muchos autores han propuesto que el origen de la vida en la Tierra ha surgido gracias a un proceso evolutivo. Es pausible que el proceso evolutivo aplicado en miles de años y a partir de la materia inerte, se haya logrado obtener formas vivas muy complejas. La teoría de la evolución natural ha logrado formar estructuras complejas. La emulación de los procesos evolutivos a permitido encontrar soluciones para resolver problemas complejos. Charles Darwin fue el primer autor que propuso la teoría de la evolución que se basa en la selección natural y en la diversidad. La selección natural la impone el ambiente en donde radique un ser vivo ya que determina su sobrevivencia y con ello la posibilidad de reproducirse. A la capacidad que tiene un individuo para sobrevivir y reproducirse se le llama *aptitud*. El medio ambiente siempre cambia, por lo cual, las especies que son aptas en un momento, ya no lo pueden ser después.

El fin último de todo ser vivo es sobrevivir ya sea como individuo o como especie. Todos los seres vivos buscan este fin a pesar de la presión que reciben del medio ambiente. En un proceso que continuamente afecta a todos los seres vivos, los que sobreviven se "adaptan" al medio ambiente y los que no lo logran, gradualmente van desapareciendo hasta extingirse por completo. Si una especie sobrevive, tiene la capacidad de reproducirse y transmitir sus características a sus descendientes, lo cual les ayudará a sobrevivir también. Los seres vivos que no sobrevivan no podrán reproducirse y desaparecerán. Un ser vivo que no sobrevive en un ambiente implica la no adaptación a él o que no se tienen las características necesarias para existir y con ello propagar sus características. Las propiedades o características de adaptabilidad de los seres vivos pueden ser simuladas en varios sistemas y algoritmos; sin embargo, es necesario conocer a detalle la teoría evolutiva. Primero daremos un repaso a la computación evolutiva y después revisaremos los algoritmos evolutivos más usados. La finalidad de este capítulo es tener un contexto general sobre los algoritmos evolutivos desde un punto de vista de proceso y de implantación.

2.2 Algoritmos evolutivos

La teoría de la evolución natural propone una explicación sobre el origen de las especies; también propone la existencia de procesos biológicos que permiten la supervivencia de una especie. Una consecuencia de los procesos evolutivos es la diversidad entre los individuos de una misma especie. El proceso de la selección natural permite que los individuos más aptos para sobrevivir, transmitan sus características a sus descendientes. Los individuos menos aptos tenderán a desaparecer y los más aptos permanecerán en el medio ambiente porque están mejor adaptadas a él (Darwin, 1997). Muchos autores han propuesto que la teoría evolutiva es universal ya que permite explicar la transmisión de las ideas (Dawkins, 1991), el origen de los sistemas nerviosos (Edelman, 1987), el sistema inmunológico, el lenguaje, etc. (Cziko, 1995; Plotkin, 1997). Si la teoría es convincente al explicar el origen de estructuras tan complejas como el cerebro, entonces podemos simular computacionalmente los procesos evolutivos para obtener estructuras o soluciones altamente eficientes.

Desde que se propuso la teoría evolutiva se han propuesto varios algoritmos o heurísticas que explican los procesos evolutivos. La primera descripción de un proceso evolutivo es la de Darwin, en donde establece que en todo proceso evolutivo se presenta la selección natural y además existen mecanismos que provocan una alta diversidad en la población. R. C. Lewontin simplifica el proceso evolutivo como una secuencia de eventos que incluye mecanismos para introducir variaciones ciegas, un proceso consistente de selección y finalmente hay mecanismos para la preservación y/o propagación de variantes (individuos con variaciones ciegas) seleccionadas (Lewontin, 1970). Donald T. Campbell propuso la heurística *genera, prueba, regenera*. La idea de Campbell es mostrar la naturaleza inventiva de los procesos evolutivos. Las palabras *genera, prueba, regenera* se refiere a tres fases continuas y consecutivas. La primera fase es la generación devariantes, la segunda fase considera a las pruebas realizadas a cada variante así como a su selección y la tercera fase se refiere a la regeneración de variantes, combinando las que fueron seleccionadas previamente (Campbell, 1960). G. C. Williams y Richard Dawkins proponen una forma diferente de ver a los procesos evolutivos. Ellos formularon la propuesta *replicador, interactuador, linaje*. Los replicadores son entidades que pueden hacer copias de sí mismos y se propagan en el espacio y se conservan o desaparecen en el tiempo. Todo replicador tiene contacto con el medio ambiente por medio de un interactuador. Al linaje pertenecen entidades que pueden cambiar indefinidamente a través del tiempo como resultado de la replicación y la interacción con el medio ambiente (Dawkins, 1991).

En las cuatro propuestas anteriores se pueden percibir varios procesos evolutivos. Primero tenemos el proceso evolutivo de variación o replicación que altera ciegamente las estructuras de formas diversas. En segundo lugar tenemos los procesos de selección que determina las estructuras más adecuadas para aplicarles una variación y finalmente tenemos los procesos de interacción o evaluación. El siguiente paso consiste en crear un algoritmo general evolutivo basándose en las ideas mostradas en esta sección.

2.3 Estructura general e implantación del algoritmo evolutivo

Para la implantación de los algoritmos evolutivos, según la sección anterior, son necesarios varios procedimientos. Es necesario un mecanismo de selección, un mecanismo de variación, un mecanismo de evaluación o de interacción con el medio ambiente y una forma de representación. Podemos separar cada procedimiento en funciones más específicas. Para trabajar con los algoritmos evolutivos es necesario establecer una analogía con los seres vivos (la teoría puede tratarse desde un punto de vista más universal; sin embargo, para propósitos de ilustración usaremos como referencia la evolución de las especies). Los seres vivos forman poblaciones en donde conviven e interactúan entre ellos y su medio ambiente; nacen, crecen, se reproducen y finalmente mueren. El proceso se repite por generaciones.

La analogía es que trabajamos con un conjunto de estructuras con funciones específicas; cada estructura representa a un individuo. El conjunto de estructuras es análogo a la población de individuos. Una estructura tienen relación con el código genético de cada individuo que es conocido como *genotipo*. Las funciones de cada estructura son análogas con el *fenotipo* de un individuo. La interacción con el medio ambiente permite evaluar la aptitud de cada individuo (se evalúa el fenotipo de un individuo). Los mecanismos de variación y replicación se relacionan con la reproducción. La reproducción se puede lograr por medio de la unión o el cruzamiento de dos individuos. La mutación de un individuo está relacionada con su variabilidad. La reproducción forma nuevos individuos con características similares a las de sus padres. El número de hijos por pareja tiene relación con la fertilidad. Es posible manejar varias poblaciones; por lo tanto, es necesario incluir funciones de desplazamiento entre poblaciones. Finalmente tenemos varios criterios para iniciar un algoritmo evolutivo así como para finalizarlo. Idealmente debería funcionar indefinidamente, pero por cuestiones prácticas su funcionamiento debe interrumpirse en un momento dado. En las siguiente secciones se revisarán tres algoritmos evolutivos que desde un punto de vista personal, son los más usados. La intención es demostrar que los procedimientos mencionados arriba se usan en los algoritmos evolutivos; bajo esta premisa, será posible aprovechar los algoritmos desarrollados en las siguientes secciones para tener algoritmos evolutivos con propiedades exaptivas.

2.4 Algoritmos genéticos

Los algoritmos genéticos (AGs) surgieron con los trabajos de Holland (1975) y se consolidaron como algoritmos de optimización y búsqueda por los trabajos realizados por algunos estudiantes de Holland como Goldberg y DeJong (Goldberg, 1983; DeJong, 1975). Los algoritmos genéticos son los algoritmos evolutivos más usados. La característica principal de estos algoritmos es la representación de cada individuo. Todos los individuos que en nuestro caso pueden ser posibles soluciones a un problema, son codificados mediante un alfabeto de baja cardinalidad. Dado que la representación

tiene analogía con los ácidos nucléicos que existen en los genes de los seres vivos, suele llamarse cromosoma a la palabra formada y que representa a una posible solución. La codificación determina el tamaño del cromosoma. La forma de representación más común es la binaria (una cardinalidad de tamaño dos) y suele usarse un vector de números binarios o bits para representar cada individuo. En la evaluación, se usa un procedimiento de decodificación y se le asigna a cada individuo un valor de desempeño o aptitud (Goldberg, 1989).

El seudocódigo de un algoritmo genético inicia con la generación de una población con individuos aleatorios. El proceso de evaluación requiere de una decodificación; es decir, es necesario decodificar las funciones o los parámetros del cromosoma ya que es lo que se va a evaluar. Todos los individuos son evaluados para obtener un valor de evaluación o aptitud. El valor de evaluación o aptitud se usa para seleccionar a los mejores individuos que serán los padres en la siguiente generación. La reproducción genera nuevos hijos a través de los padres. El mecanismo de cruce entre dos individuos genera hijos y el mecanismo de mutación altera con un grado pequeño de probabilidad el cromosoma de cada uno de los hijos. Finalmente se establece un criterio para terminar el algoritmo genético. Una forma de terminar consiste en ejecutar el algoritmo y esperar una solución aceptable aunque lo más común es ejecutar el algoritmo con un número fijo de evaluaciones.

1) $P \leftarrow$ Generación(N_{TI}, T_C)
2) $F_E \leftarrow$ Evaluación(P)
3) $P \leftarrow$ Selección(P, F_E)
4) $P \leftarrow$ Cruce(P)
5) $P \leftarrow$ Mutación(P)
6) Si es fin de algoritmo entonces terminar sino ir a (2)

Figura 2.1: Seudocódigo de un algoritmo genético.

El algoritmo genético se muestra en la figura 2.1. El primer proceso es la generación de una población de N_{TI} individuos; T_C es el tamaño del cromosoma que tendrá cada individuo. P es una matriz donde cada fila es un vector que representa a un individuo. El tamaño del vector es el tamaño del cromosoma T_C. El valor de evaluación de cada individuo es almacenado en el vector F_E. La selección de los individuos más aptos, usa como referencia el vector F_E y sustituye a los individuos menos aptos por copias de los individuos más aptos. Si toda la población es sustituida entonces tenemos un algoritmo genético generacional (todos los hijos sustituyen a sus padres), si sólo sustituimos a una parte de la población, entonces el algoritmo es llamado no generacional (algunos padres permanecen con sus hijos). Los mecanismos de cruce y de mutación son aplicados exclusivamente a los individuos ya seleccionados bajo un criterio probabilístico. El cruce es el intercambio de bits entre los vectores de dos individuos y la mutación consiste en alterar probabilísticamente cada bit del vector de un individuo.

El siguiente ejemplo es un demostración muy sencilla sobre el funcionamiento de un

algoritmo genético. Primero generamos una población de cuatro individuos ($N_{TI} = 4$) con cromosomas de $T_C = 4$ bits. Usaremos las vocales para representar cada gen, por lo cual, la cardinalidad es de tamaño 5. Supongamos que tenemos la siguiente población inicial:

$$\text{P} = \{\text{aiuo, eeio, ieie, ueou}\}$$

Supongamos que para la evaluación de los individuos de este ejemplo se toma cada letra y si es una "a" entonces se le asigna un valor de 5; por cada letra "i" se le asigna un 2. Para las demás vocales no se darán ningún valor. Al evaluar el primer individuo de la población, es decir, "aiuo" tenemos una evaluación igual a $5 + 2 + 0 + 0 = 7$. La evaluación de cada individuo se almacena en el vector F_E:

$$F_E = \{7, 2, 4, 0\}$$

Si sólo se seleccionan a los dos mejores individuos con la evaluación más alta entonces los padres son los siguientes:

$$\text{Padres} = \{\text{aiuo, ieie}\}$$

Para realizar el mecanismo de cruce es necesario establecer puntos de cruce o de intercambio de bits, en el cual los hijos tendrán cromosomas que son formados por las contribuciones que realicen los padres. En el ejemplo tenemos tres posibles puntos de cruce en el cromosoma. Si suponemos que el punto de cruce es la tercera posición (generalmente la posición se determina en forma aleatoria con una distribución uniforme), tenemos un intercambio de genes (bits) dando lugar a dos hijos:

$$\text{Padre 1} = \{\text{aiu - o}\}, \text{Padre 2} = \{\text{iei - e}\}$$
$$\text{Hijos} = \{\text{aiu - e, iei - o}\}$$

La mutación es un cambio fortuito en uno o más genes del cromosoma de un individuo. Si suponemos un cambio en el individuo "ieio" en la segunda posición por la letra "o", entonces obtenemos un nuevo individuo.

$$\text{Hijo} = \{\text{i e i o}\} \rightarrow \text{Hijo mutado} = \{\text{i o i o}\}$$

Los hijos sustituirán a los peores individuos, es decir, los que no fueron seleccionados. El algoritmo genético es no generacional. La población resultante es la siguiente:

$$\text{P} = \{\text{aiuo, ieie, aiue, ioio}\}$$

Estos pasos los podemos ejecutar varias veces. Después de varios ciclos podríamos obtener individuos con vocales "a" o "i" aunque todos tenderían al individuo "aaaa" que es el óptimo.

Del algoritmo genético se han derivado varios algoritmos evolutivos. Por ejemplo tenemos AGs de estado estable, AGs con búsqueda local o híbridos, AGs con especies

y nichos, AGs con islas e inyecciones, AGs con diploides y dominancias, AGs paralelos, AGs multiobjetivo, AGs messy (Bentley, 1999). Una mención aparte son los algoritmos genéticos que usan representaciones especializadas con operadores evolutivos específicos para manipularlas. Se busca lograr que las computadoras evolucionen programas usando representaciones de alto nivel, no precisamente vectores binarios sino listas que contienen operadores o instrucciones. Este tipo de algoritmos reciben el nombre de programación genética (Koza, 1993, 1994).

Existen algoritmos genéticos que no usan una representación específica. Este tipo de algoritmos que no usan codificación pero usan los mismos mecanismos de cruce y mutación (aunque en forma aritmética) son llamados algoritmos genéticos con genes reales (Michalewicz, 1999). La forma de implantar estos algoritmos es similar a los algoritmos genéticos. La diferencia es que no es necesario aplicar un procedimiento de decodificación porque se trabaja con los mismos parámetros representados como vectores de números reales.

2.5 Estrategias evolutivas

Las estrategias evolutivas fueron desarrolladas en Alemania por Rechenberg y Schwefel. Estos algoritmos usan poblaciones de individuos sin codificar; por lo tanto, pueden usarse directamente los valores de los parámetros en forma de vectores de números reales. Las estrategias evolutivas generan una población a partir de uno o varios padres. Tanto los hijos como los padres comparten un lugar en la población. Los hijos son evaluados y seleccionados para ser padres y sustituir a sus progenitores. Es una regla que se tengan más hijos que padres (Schwefel, 1981).

Los mecanismos de selección están basados en la aptitud de cada individuo. La generación de nuevos hijos a partir de los padres se basa en la suma de un valor aleatorio con distribución normal $N(r, \sigma)$ a cada padre. El parámetro r es la desviación estándar que tiene la función generadora de números aleatorios y σ es el tamaño de cada paso que puede tener el vector de parámetros respecto a un valor inicial. Usualmente se aplica la misma constante para el vector de parámetros aunque existen versiones del algoritmo en donde se usa una constante única σ para cada elemento del vector. En algoritmos más avanzados, la constante σ pueden ajustarse automáticamente; por lo cual, son llamados estrategias evolutivas con auto-adaptación.

Como ejemplo, tenemos una población de cuatro padres n_a, n_b, n_c, n_d. Podemos generar cuatro hijos por cada padre (relación padre: hijo de 1:4) por medio de una distribución normal $N(r, \sigma)$. Los hijos de a son:

$$\begin{aligned} h_{na1} &= n_a + N(r, \sigma) \\ h_{na2} &= n_a + N(r, \sigma) \\ h_{na3} &= n_a + N(r, \sigma) \\ h_{na4} &= n_a + N(r, \sigma) \end{aligned}$$

Lo mismo se aplica para generar los hijos de los padres n_b, n_c y n_d. El seudocódigo del algoritmo de estrategias evolutivas se muestra en la figura 2.2.

1) $(P_P, P_H) \leftarrow$ Inicio$(N_{TP}, N_{TH}, N_{TPr})$
2) $P_H \leftarrow$ Mutación(P_P)
3) $F_E \leftarrow$ Evaluación(P_P, P_H)
4) $P_P \leftarrow$ Selección(P_H)
5) Si es fin de algoritmo terminar, sino ir a (2)

Figura 2.2: Seudocódigo del algoritmo de estrategias evolutivas.

La constante N_{TP} indica el número de padres deseados, N_{TH} indica el número de hijos y N_{TPr} representa el número total de parámetros. P_P es la matriz que retiene a los padres. Es una matriz de N_{TP} filas por N_{TPr} columnas. P_H es una matriz que almacena a los hijos y tiene N_{TH} filas por N_{TPr} columnas. F_E es un vector para almacenar la evaluación de cada individuo y es de tamaño N_{TP}. El procedimiento de inicio genera padres aleatorios con sus respectivos hijos. Cada padre e hijo es un vector con varios parámetros generados aleatoriamente. Las matrices P_P y P_H retienen a los vectores aleatorios de los padres e hijos respectivamente. El procedimiento de mutación es la generación de hijos. De un solo padre se generan varios hijos. La regla es que deben ser más hijos que padres. En la evaluación se realiza la selección de padres e hijos para ser padres nuevamente. De la matriz P_P se eliminan los padres menos aptos y son sustituidos por padres o hijos más aptos. Finalmente se establece un criterio para terminar el algoritmo.

2.6 Programación evolutiva

La programación evolutiva es un concepto creado por Lawrence Fogel en donde por medio de mecanismos de mutación, se evolucionan máquinas de estados finitos (Fogel et al., 1966). David Fogel aplicó el mismo concepto desarrollado por su padre en vectores de parámetros (Fogel, 1991). El objetivo del algoritmo de la programación evolutiva consiste en aplicar mecanismos evolutivos de selección y mutación a varios individuos.

La población inicial es generada con vectores aleatorios. El mecanismo de selección consiste en enfrentar la aptitud de un individuo con la de los demás. Esto equivale a una competencia aleatoria en donde los mejores individuos adquieren más puntos. Por medio de la mutación se generan nuevos individuos. El mecanismo de mutación es muy similar al usado en el algoritmo de estrategias evolutivas. A cada parámetro se le añade un valor aleatorio generado con una distribución normal. El enfrentamiento entre individuos (incluyendo a los nuevos) dará como resultado la sustitución de los individuos más débiles por los más fuertes.

El siguiente ejemplo muestra el funcionamiento general del algoritmo para la programación evolutiva. Dados dos individuos n_a y n_b se generan dos hijos:

$$h_{na} = n_a + N(\mu, \sigma^2)$$

$$h_{nb} = n_b + N(\mu, \sigma^2)$$

donde $N(\mu, \sigma^2)$ es una variable aleatoria gaussiana, μ es la media y σ^2 es la varianza; luego evaluamos a los padres e hijos, suponemos que las evaluaciones de cada individuo son:

Individuo	Evaluación
n_a	65
n_b	25
h_{na}	43
h_{nb}	37

Ahora aplicamos el mecanismo de selección en todos los individuos. La selección consiste primero en establecer un número de enfrentamiento entre los individuos. El siguiente paso consiste en enfrentar a cada individuo con otros en forma aleatoria (se establece un ganador entre dos individuos al compararse el valor de evaluación). Cada individuo acumula sus victorias, así que los individuos con más enfrentamientos ganados son seleccionados para ser los padres en la siguiente generación. Siguiendo con el mismo ejemplo, si establecemos un número de enfrentamientos igual a dos, tenemos:

Individuo	Competencia	Selección
n_a	2	Seleccionado
n_b	0	Eliminado
h_{na}	1	Seleccionado
h_{nb}	0	Eliminado

El individuo n_a se enfrenta con dos individuos aleatorios, supongamos que son los individuos h_{na} y n_b. Como tiene una evaluación muy alta el individuo n_a gana ambos enfrentamientos. El individuo n_b no tiene ninguna oportunidad de ganar ya que tiene un valor de evaluación muy bajo. Si el individuo h_{na} se enfrenta con los individuos n_a y h_{nb} ganará sólo el enfrentamiento con h_{nb}. Finalmente el individuo h_{nb} puede tener la mala fortuna de enfrentarse a los individuos n_a y h_{na} por lo cual perderá ambas contiendas. Los individuos con más enfrentamientos ganados serán seleccionados para generar nuevos individuos. En el ejemplo corresponde a los individuos n_a y h_{na}.

La implantación del algoritmo de la programación evolutiva se muestra en la figura 2.3.

1) $P \leftarrow$Inicio(N_{TI}, N_{TPr})
2) $P \leftarrow$Mutación(P)
3) $F_E \leftarrow$Evaluación(P)
4) $P \leftarrow$Selección(P, F_E)
5) Si es fin de algoritmo terminar, sino ir a (2)

Figura 2.3: Seudocódigo del algoritmo de la programación evolutiva.

El algoritmo empieza considerando las constantes N_{TI} que se usa para representar el número total de individuos, y N_{TPr} que se usa para representar el número total de parámetros. En el procedimiento de inicio se realiza la generación de individuos que son almacenados en la matriz P. Cada fila de la matriz P, representa a un individuo que es un vector de parámetros aleatorios. También es generado el vector F_E para almacenar las aptitudes de cada individuo. El procedimiento de mutación, de evaluación y de selección tienen el mismo funcionamiento explicado anteriormente. Al final se aplica un criterio para terminar el algoritmo.

2.7 Conclusiones

La computación evolutiva surgió como una forma de representación y simulación de los procesos evolutivos. Al principio sólo se sugirieron heurísticas para describir a cada uno de los procesos evolutivos que al final de cuentas se simplifica a tres procesos básicos: El proceso de evaluación, el proceso de selección y el proceso de variación.

Tomando como base estos procesos se han desarrollado varios algoritmos evolutivos sugerido por varios autores y desde distintos lugares del mundo. En Europa se han inclinado más por el uso de las estrategias evolutivas que es un algoritmo evolutivo basado sobre todo en mutaciones especializadas sobre poblaciones en donde padres e hijos conviven y compiten entre sí. En los Estados Unidos la computación evolutiva empezó con los trabajos de Lawrence Fogel y su programación evolutiva en el cual su hijo David Fogel ha extendido su uso. Estos algoritmos se basan en competencias entre padres e hijos. Los algoritmos genéticos son una creación de Holland y son los algoritmos más usados (inclusive en todo el mundo.)

La principal diferencia que tienen los algoritmos genéticos respecto a la programación evolutiva y a las estrategias evolutivas es el mecanismo de codificación. La codificación usada en los algoritmos genéticos le ha dado ventaja porque permite representar casi cualquier problema y no sólo la representación de parámetros como lo hacen las estrategias evolutivas y la programación evolutiva. Lo que se quiere demostrar es que todos los algoritmos evolutivos se basan en los procesos evolutivos ya mencionados y cuando surge otro proceso evolutivo como la exaptación; entonces es posible implantarlo como se realizó con los procesos evolutivos de la selección, la evaluación y la variación ciega.

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

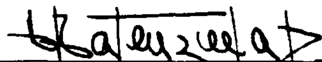
División de Electrónica, Computación, Información y Comunicaciones

Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

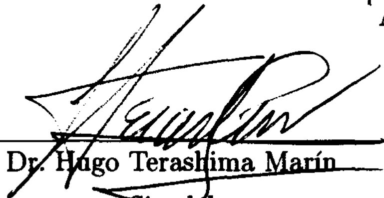
Los miembros del comité de tesis recomendamos que la presente tesis del M. C. Luis Martín Torres Treviño sea aceptada como requisito parcial para obtener el grado académico de **Doctor en Ciencias**, especialidad en:

Sistemas Inteligentes

Comité de tesis:



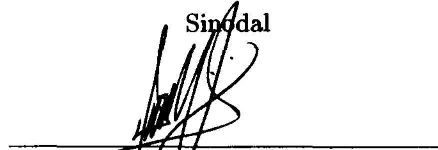
Dr. Manuel Valenzuela Rendón
Asesor de la tesis



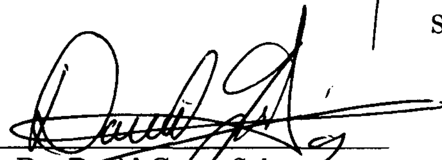
Dr. Hugo Terashima Marín
Sinodal

Dr. Carlos Coello Coello
Sinodal

Dr. Francisco Cantú
Sinodal



Dr. Horacio Martínez Alfaro
Sinodal



Dr. David Garza Salazar
Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

Mayo de 2004

Reconocimientos

Agradezco al CONACYT y al Centro de Sistemas Inteligentes del ITESM por el apoyo recibido durante el estudio. Agradezco también al Dr. Manuel Valenzuela Rendón por su conocimiento y apoyo durante el desarrollo de la tesis. Una mención especial a los Doctores Rogelio Soto, director del centro y al Dr. Hugo Terashima, coordinador del doctorado ya que permitieron culminar satisfactoriamente la tesis. También agradezco al Doctor Francisco Cantú por su apoyo en las etapas iniciales del estudio del doctorado y al Doctor Carlos Coello por sus oportunos consejos en el desarrollo de la tesis. Un agradecimiento al personal administrativo, en especial a la secretaria Dora Juarez. Agradezco a mis amigos Elizabeth Acosta, Ana Castillo, Dora Juarez, Araceli Velazquez, Adriana Cantú, Israel Martinez, Eduardo Novelo, José Galvez, José Mariscal, Javier Minero, Roberto Carrillo, Victor Uc, Roberto Gutierrez, Fernando Valles, Carlos Bautista, Fernando Von Borstel, Joaquín Gutiérrez y a todos los que con su aliento y sus consejos, permitieron culminar la tesis.

Finalmente agradezco profundamente a mi esposa Sandra Mariscal, a mi madre Elena Treviño y a mis Hermanos Luz Elena Torres, Daniel Torres y Diego Torres por creer en mi.

LUIS MARTÍN TORRES TREVIÑO

Instituto Tecnológico y de Estudios Superiores de Monterrey
Mayo 2004

SISTEMAS EXAPTIVOS: RETENCIÓN Y REUTILIZACIÓN DE CONOCIMIENTO EN ALGORITMOS EVOLUTIVOS

Luis Martín Torres Treviño, Dr.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2004

Asesor de la tesis: Dr. Manuel Valenzuela Rendón

El objetivo de la tesis es desarrollar un sistema con la capacidad de retener y reutilizar soluciones para resolver problemas dinámicos. Se considera que existen dos tipos de problemas dinámicos, los primeros se relacionan con la optimización de funciones dinámicas y los segundos se relacionan con el aprendizaje en un medio ambiente dinámico. La solución de los problemas dinámicos son importantes porque es el común de los problemas reales. La tesis se inspira en la exaptación que es el proceso evolutivo en el cual una estructura adaptada para una función particular se utiliza para lograr otra función generalmente diferente a la que se adaptó. Para implantar el proceso evolutivo de la exaptación se analizan algunos algoritmos evolutivos. También se analizan los mecanismos de aprendizaje que utilizan algoritmos evolutivos ya que se quiere involucrar a la exaptación en los mecanismos de aprendizaje. La exaptación tiene una relación con el aprendizaje por analogía; por lo cual también puede brindar una guía para la implantación de un sistema con capacidades exaptivas. La exaptación se divide en dos procedimientos, el primero es la reutilización de soluciones, el segundo es la retención. Para el primer procedimiento se analizan técnicas para modificar soluciones e insertarlas en la población inicial de un algoritmo genético u otro algoritmo evolutivo. Para el segundo procedimiento se analizan algunos algoritmos evolutivos que mantienen la diversidad en la población y se utilizan mecanismos de memoria junto con algunas técnicas de reutilización para formar sistemas evolutivos capaces de resolver problemas dinámicos. En la optimización de funciones dinámicas se probaron dos algoritmos con capacidades exaptivas y se comparó el desempeño con otro algoritmo especializado en resolver estos problemas. Se demuestra que los algoritmos propuestos son competitivos. En el aprendizaje en un medio ambiente dinámico, se utilizó una red neuronal sencilla con propiedades exaptivas que le permite aprender varias funciones en una sola estructura. Finalmente se propone el uso de un agente inteligente con capacidades exaptivas

en la reprogramación de tareas para demostrar que los procedimientos exaptivos propuestos le brindan alguna ventaja al agente comparado con otro agente que no tiene capacidades exaptivas.

A mi padre por sus enseñanzas y ejemplo para alcanzar las metas con tenacidad.

Índice de Figuras

2.1 Seudocódigo de un algoritmo genético. 9
2.2 Seudocódigo del algoritmo de estrategias evolutivas. 12
2.3 Seudocódigo del algoritmo de la programación evolutiva. 13

3.1 La relación funcional se forma estableciendo una relación entre los estados y las acciones con los componentes. 16
3.2 Algoritmo general de aprendizaje. 16
3.3 Una relación funcional representada como una ecuación matemática. . . 17
3.4 Una relación funcional representada como una ecuación matemática; mismos componentes, diferente estructura. 18
3.5 Algoritmo de aprendizaje evolutivo. 20
3.6 Algoritmo general del sistema adaptable de clasificadores. 21

4.1 Algoritmo general del razonamiento basado en casos. 29
4.2 Algoritmo general del sistema adaptable de clasificadores con memoria. 30

6.1 Algoritmo general para la exaptación. 41

7.1 Seudocódigo de un algoritmo genético simple con técnicas de sembrado. 47
7.2 Función a optimizar con $v_r = 0.5$. 50
7.3 Generación de vecindades. 51
7.4 Desempeño en la optimización de una función cambiante de tres AGS's que utiliza sembrado contra un AGS que no lo usa. 55
7.5 Desempeño en la programación de tareas con reutilización de la solución anterior de tres AGS's que utiliza sembrado contra un AGS que no lo usa. 55
7.6 Desempeño en la optimización del problema de la mochila cambiante de tres AGS's que utiliza sembrado contra un AGS que no lo usa. 56

8.1 Algoritmo PBIL 58
8.2 Algoritmo CGA 59
8.3 Algoritmo BOA 60
8.4 Algoritmo de mutación dirigida. 61
8.5 Algoritmo de mutación dirigida con reutilización de la solución I. . . . 63
8.6 Algoritmo de PBIL y CGA con reutilización de la solución I. 64
8.7 Desempeño del algoritmo genético simple comparado con los algoritmos de mutación dirigida, PBIL y CGA. 65

9.1 Estructura general de un sistema con capacidades exaptivas. 68
9.2 Algoritmo de reconocimiento por evaluación. 69
9.3 Un mecanismo de reconocimiento general. 70
9.4 Algoritmo de almacenamiento por similitud y mejor evaluación 71
9.5 Algoritmo de un sistema con características exaptivas. 72
9.6 Algoritmo genético exaptivo. 73
9.7 Algoritmo genético exaptivo con inyecciones. 73
9.8 Algoritmo genético simple. 74
9.9 Algoritmo genético simple con elitismo. 74
9.10 Algoritmo genético basado con dos poblaciones propuesto por Branke. . 75
9.11 Algoritmo genético con técnicas de sembrado. 76
9.12 Algoritmo genético con memoria y sembrado. 77
9.13 Desempeño de varios algoritmos con severidad $s = 0.0$ y factor de cambio $\lambda = 1$. 79
9.14 Desempeño de varios algoritmos con severidad $s = 0.5$ y factor de cambio $\lambda = 0.9$. 79

10.1 La red neuronal de funcionamiento por inhibición. 82
10.2 La función triangular tiene dos parámetros para controlar su comportamiento. 83
10.3 Las neuronas de activación excitan a su neurona de activación correspondiente e inhiben a las demás neuronas de activación. 83
10.4 Red neuronal de funcionamiento por inhibición de una entrada y una salida. 84
10.5 Procedimiento de reconocimiento. 85
10.6 Procedimiento de evaluación. 86
10.7 Algoritmo de búsqueda global. 86
10.8 Algoritmo de búsqueda local. 87
10.9 Procedimiento de almacenamiento. 87
10.10 Algoritmo del sistema seudoexaptivo neuronal. 88
10.11 Respuesta de las funciones K, K_2 y K_3. 89
10.12 Desempeño de la red neuronal basado en un algoritmo genético simple y en un sistema seudoexaptivo. 91

11.1 Estructura de un agente con características exaptivas. 94
11.2 Costos obtenidos en la programación de las tareas de cada agente en 100 días. 98
11.3 Desempeño promedio de ambos agentes en la reprogramación de tareas durante 100 días. 99

Índice General

Resumen iii

1 Introducción 1
1.1 Motivación 1
1.2 Estructura general de la investigación 3
1.3 Preguntas clave de la investigación 4
1.4 Contribuciones 4
1.5 Estructura de la tesis 5

2 La computación evolutiva 6
2.1 Los procesos evolutivos 6
2.2 Algoritmos evolutivos 7
2.3 Estructura general e implantación del algoritmo evolutivo 8
2.4 Algoritmos genéticos 8
2.5 Estrategias evolutivas 11
2.6 Programación evolutiva 12
2.7 Conclusiones 14

3 Mecanismos de aprendizaje 15
3.1 Introducción 15
3.1.1 Implantación de un algoritmo general de aprendizaje 15
3.1.2 Algunos ejemplos de relaciones funcionales 17
3.1.3 Aprendizaje supervisado y no supervisado 18
3.2 Algoritmos evolutivos en los mecanismos de aprendizaje 19
3.3 Sistemas adaptables de clasificadores 20
3.3.1 Componentes de un sistema adaptable de clasificadores 20
3.4 Redes Neuro evolutivas 21
3.5 Sistemas difusos evolutivos 22
3.6 Conclusiones 23

4 Aprendizaje por analogía 25
4.1 Introducción 25
4.2 Métodos del aprendizaje por analogía 26
4.2.1 Descripción y detección de similitudes 26

4.2.2 Modificación de soluciones 27
4.2.3 Almacenamiento 28
4.3 Razonamiento basado en casos 28
4.4 Sistema adaptable de clasificadores con memoria 29
4.5 Redes neuronales de aprendizaje incremental 30
4.6 Conclusiones 31

5 Solución de problemas dinámicos **33**
5.1 Optimización de funciones dinámicas 33
5.2 Aprendizaje en un medio ambiente dinámico 35
5.3 Conclusiones 36

6 La exaptación y los nuevos procesos evolutivos **38**
6.1 Introducción a la nueva teoría evolutiva 38
6.2 La adaptación y la exaptación 39
6.3 Procedimientos de la exaptación 40
6.4 Algoritmos para tener exaptación 41
6.5 Algoritmos evolutivos con exaptación 42
6.6 Conclusiones 42

7 Clasificación y evaluación de las técnicas de sembrado **43**
7.1 Introducción 43
7.2 Técnicas para la reutilización de soluciones en algoritmos evolutivos . . 45
7.3 Reutilización de soluciones en tres tipos de problemas 46
7.3.1 El problema de la mochila 47
7.3.2 La optimización de funciones cambiantes 48
7.3.3 La programación de tareas 50
7.4 Experimentación y resultados 53
7.5 Conclusiones 54

8 Reutilización del conocimiento en la mutación dirigida **57**
8.1 Algoritmos PBIL, CGA y BOA 57
8.2 La mutación dirigida 58
8.3 La reutilización de soluciones 62
8.4 Experimentación con la reutilización de soluciones 62
8.5 Conclusiones 63

9 Sistemas con capacidades exaptivas **66**
9.1 Introducción 66
9.2 Características y Componentes de un sistema seudoexaptivo 67
9.2.1 Mecanismos de reconocimiento 68
9.2.2 Mecanismo para la modificación 69
9.2.3 Almacenamiento 70
9.2.4 Mecanismos de búsqueda 71
9.3 Formas de implantación de sistemas con características exaptivas 71

9.4 Experimentación 74
9.4.1 Implantación de los algoritmos 74
9.4.2 Función de prueba 76
9.4.3 Descripción del experimento y resultados 77
9.5 Conclusiones 78

10 Redes neuronales y los algoritmos genéticos con exaptación 81
10.1 El sistema seudoexaptivo neuronal 81
10.2 La red neuronal de funcionamiento por inhibición 82
10.3 Características del sistema exaptivo neuronal 85
10.4 Experimentaciones sobre el comportamiento del sistema seudoexaptivo neuronal 87
10.5 Conclusiones 91

11 Agentes inteligentes con capacidades exaptivas 93
11.1 Características de un agente inteligente con exaptación 93
11.2 El agente seudoexaptivo en la reprogramación de tareas 94
11.3 Experimentación 97
11.4 Conclusiones 98

12 Conclusiones, contribuciones y trabajo futuro 100
12.1 Conclusiones 100
12.2 Contribuciones 102
12.3 Trabajo futuro 103

Bibliografía 105

SISTEMAS EXAPTIVOS: RETENCION Y REUTILIZACION DE CONOCIMIENTO EN ALGORITMOS EVOLUTIVOS

Por

M. C. LUIS MARTIN TORRES TREVIÑO

TESIS

PRESENTADA AL PROGRAMA DE GRADUADOS EN ELECTRONICA, COMPUTACION, INFORMACION Y COMUNICACIONES DEL INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY, CAMPUS MONTERREY,

COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADEMICO DE

DOCTOR EN CIENCIAS

TECNOLÓGICO DE MONTERREY.

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

MONTERREY, N. L. MAYO DE 2004

SISTEMAS EXAPTIVOS: RETENCION Y REUTILIZACION DE CONOCIMIENTO EN ALGORITMOS EVOLUTIVOS

Por

M. C. LUIS MARTIN TORRES TREVIÑO

TESIS

PRESENTADA AL PROGRAMA DE GRADUADOS EN ELECTRONICA, COMPUTACION, INFORMACION Y COMUNICACIONES DEL INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY CAMPUS MONTERREY.

COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADEMICO DE

DOCTOR EN CIENCIAS

TECNOLÓGICO DE MONTERREY.

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

MONTERREY, N. L. MAYO DE 2004

Capítulo 6

La exaptación y los nuevos procesos evolutivos

6.1 Introducción a la nueva teoría evolutiva

La teoría de la evolución implica muchos procesos además de la selección y la variación. Esta nueva tendencia propone nuevas ideas que enriquecen la teoría evolutiva, como las siguientes:

- Los equilibrios puntuados
- La mutación dirigida
- La exaptación
- La simbiosis
- La auto-organización

Todas las teorías que han surgido sobre la computación evolutiva se basan en la evolución de los seres vivos, es por eso que hablamos en términos de individuos; sin embargo, la analogía es sencilla al definir que un individuo equivale a una solución, ya sea un conjunto de parámetros o una relación funcional. Lo anterior nos da la posibilidad de extender cualquier proceso evolutivo a la computación evolutiva. Los nuevos procesos evolutivos mencionados arriba no son la excepción ya que pueden representarse en términos de algoritmos evolutivos; tal es el caso de la exaptación y la mutación dirigida que se presentan en esta tesis.

Los ***equilibrios puntuados*** es una teoría donde se establece que los mecanismos de selección no actúan en forma constante; suelen pasar periodos en donde los individuos casi no cambian y otros en donde los individuos cambian mucho (Eldredge y Gould, 1972). La ***mutación dirigida*** es una teoría propuesta por Carnes y su grupo de colaboradores (Cziko, 1995) que establece que algunos individuos de una población pueden aumentar y dirigir su grado de mutabilidad cuando las condiciones del ambiente cambian para lograr una adaptación más rápida. La ***simbiosis*** es la colaboración que puede

darse entre dos especies para recibir beneficios mutuos. La *exaptación* consiste en la reutilización de estructuras para darle una función distinta a la que originalmente fue adaptada, este es el tema central de la tesis, por lo tanto será analizada a detalle en las siguientes secciones. Cualquier sistema abierto (un sistema cerrado implica un sistema aislado, sin interacción con su entorno) puede tener interacciones entre sus componentes y en un momento dado puede surgir una organización entre sus componentes en tiempo o espacio, mantenerse estable o mostrar cambios de un estado a otro; el fenómeno recibe el nombre de *auto-organización* y se dice que la teoría evolutiva la facilita.

6.2 La adaptación y la exaptación

Podemos entender la exaptación si primero analizamos la adaptación. La adaptación es un proceso para construir estructuras con una utilidad basándose en los mecanismos de la selección y la variación. La adaptación permite formar estructuras con funciones bien definidas y que brindan una utilidad. Etimológicamente, el término adaptación viene de *ad* que quiere decir añadir y *aptación* que es una estructura que ofrece una utilidad (*aptus*). Se puede crear una función que relacione una estructura con una utilidad o un valor asignado por un proceso de evaluación. Si usamos la función entonces las estructuras bien adaptadas tendrán un valor de evaluación muy alto. Se ha demostrado que la adaptación también genera estructuras sin ninguna finalidad (*no aptaciones*) e inclusive se generan estructuras que son redundantes. Las estructuras sin ninguna utilidad pueden pasar mucho tiempo sin utilizarse; sin embargo, si existe un cambio en el ambiente, entonces puede existir en ellas alguna utilidad. Las estructuras pueden ser reutilizadas, a pesar de que surgieron como parte de estructuras adaptadas para otro fin.

La evolución de un ser vivo generalmente utiliza estructuras ya adaptadas, por lo cual las reutiliza. El mecanismo que brinda esta funcionalidad es la exaptación. Etimológicamente, el término exaptación viene de aptación excluida (ex - aptación) y se le llama así a las estructuras que dan una utilidad pero su origen fue por causas diferentes a las actuales. Como mencionamos anteriormente, existe una relación entre la funcionalidad que puede tener una estructura y su valor de evaluación. En un momento dado una estructura con un valor de evaluación que es un máximo global puede convertirse en un máximo local, por lo que es necesario desplazarse desde este punto hasta alcanzar el nuevo máximo global. El movimiento del punto máximo global es ocasionado por perturbaciones y cambios que suceden en el medio ambiente. La exaptación permite reutilizar lo que existe para adaptarlo a las nuevas circustancias impuestas por el medio ambiente. Finalmente tenemos estructuras adaptadas con funciones bien definidas con un alto valor de evaluación y estructuras no adaptadas sin ninguna utilidad o una función definida (tienen un valor de evaluación bajo) y surgen con las estructuras adaptadas. Un individuo puede tener ambos tipos de estructuras, aunque generalmente las estructuras útiles son tomadas en cuenta en el proceso evolutivo siempre y cuando el medio ambiente permanezca sin cambios notables (Gould y Vrba, 1982; Gould, 1991).

En resumen, los procesos evolutivos usan lo que existe en el momento para adap-

tarse a un medio ambiente específico. Las estructuras adaptadas funcionan en un ambiente determinado; si el ambiente cambia, entonces muchas estructuras que antes fueron útiles, ya no lo serán. En cambio, otras estructuras que antes eran inútiles, ahora pueden brindar una utilidad. La exaptación es la propiedad de aprovechar todas las estructuras que fueron útiles o no, que brinden una utilidad para el medio ambiente actual. Esto quiere decir que realiza la unión de estructuras útiles y de estructuras no útiles para adaptarse a las nuevas condiciones que el medio ambiente impone.

6.3 Procedimientos de la exaptación

La exaptación permite reutilizar estructuras, lo cual acelera la adaptación. La implantación de la exaptación nos permitirá reutilizar estructuras o soluciones, y además acelerar y mejorar los procedimientos de optimización o de aprendizaje. La exaptación tiene dos características relevantes; existe una unión de estructuras previamente adaptadas para un nuevo uso, también existe la unión de estructuras que surgieron con las estructuras adaptadas que no dan utilidad.

Por la sección anterior, podemos identificar varios elementos y funciones para tener exaptación. Se tienen estructuras que no tienen una función (no aptaciones). Se tienen estructuras con una utilidad bien definida (aptaciones). Existe implícitamente un mecanismo de unión, mezcla o fusión de estructuras. Los operadores son evolutivos, es decir, se basan en la selección natural y en la variación ciega. Existe una función inherente de reconocimiento o detección de similitudes. Existe una reutilización y una consideración a cambiar lo que existe; es como reutilizar lo que tenemos en un deshuesadero o sótano.

La exaptación es un procedimiento implícito en los procesos evolutivos; sin embargo, podemos hacerlo explícito. La exaptación tiene un proceso de reconocimiento basado en la utilidad con la finalidad de determinar qué estructura puede reutilizarse; se mide la utilidad de todas las estructuras y la de mayor utilidad se modifica. Este proceso también es aplicado con más de una estructura. La exaptación modifica estructuras usando procesos evolutivos y las retiene. Podemos decir que es una extensión de la adaptación. Con lo anterior, podemos identificar tres procedimientos de la exaptación:

1. Reconocimiento de estructuras útiles.

2. Modificación de estructuras.

3. Retención de estructuras modificadas.

Si analizamos los procedimientos de la exaptación, encontraremos una similitud con los procesos del aprendizaje por analogía. En la siguiente sección, se tomará ventaja de la similitud y realizaremos algoritmos para implantar sistemas con propiedades exaptivas.

6.4 Algoritmos para tener exaptación

Una estructura puede tener una alta utilidad en un ambiente, es decir, se adapta con las condiciones actuales del ambiente. Si el ambiente cambia y la estructura pierde utilidad, entonces es necesario determinar qué estructuras serán útiles ahora. Siempre se tienen estructuras redundantes e inútiles y cuando el ambiente cambia, la unión de una o más de estas estructuras pueden tener alta utilidad, mientras las que antes tenían una alta utilidad, pueden tener una baja utilidad en cualquier momento.

La exaptación permite que las estructuras de alta utilidad sean adaptadas para sobrevivir en el nuevo ambiente. La exaptación la podemos ver como un proceso del aprendizaje por analogía. Una estructura es equivalente a la solución (relación funcional o parámetros). El ambiente es equivalente a la descripción del problema o proceso. Un cambio en el ambiente es equivalente a un cambio en el problema o descripción. Una detección de una utilidad en una estructura es equivalente a una detección de una similitud entre el proceso actual y los que se resolvieron previamente. La adaptación equivale a una modificación o búsqueda. La retención es equivalente al almacenamiento.

Basándose en el aprendizaje por analogía, se puede describir un algoritmo general para obtener exaptación. Las condiciones son las siguientes. Es necesario una base de datos o memoria para guardar descripciones de problemas y sus soluciones. Cada problema tiene una descripción, las soluciones son políticas o parámetros. El problema consiste en encontrar una política (relación funcional) o un conjunto de parámetros para un proceso cambiante; ya sean políticas o parámetros, que son soluciones para satisfacer los requerimientos planteados en el proceso. El algoritmo general de la exaptación como un procedimiento de aprendizaje por analogía se muestra en la figura 6.1.

1) Lectura de descripción de un problema
2) Si existe similitud entonces modificar la solución, luego ir al punto (4).
3) Búsqueda de una solución
4) Almacenamiento de la descripción del problema con su solución
5) Si no es fin de algoritmo ir a (1), sino terminar.

Figura 6.1: Algoritmo general para la exaptación.

El objetivo de este trabajo es diseñar sistemas que reutilicen soluciones y las retengan, es decir, un sistema con capacidades exaptivas. Para verificar que un sistema es exaptivo, en necesario que cumpla con los siguientes requisitos. Se requiere de una entidad con estructuras útiles, inútiles y redundantes que detecte cuáles estructuras brindan utilidad en una nueva situación. La entidad debe adaptar estructuras para encontrar una solución a un nuevo problema. Debe almacenar o retiener las estructuras útiles; siendo más precisos, las estructuras sin utilidad también podrían almacenarse.

6.5 Algoritmos evolutivos con exaptación

El algoritmo evolutivo que utiliza poblaciones de soluciones puede desempeñar el papel de un sistema exaptivo. Al inicio el algoritmo evolutivo tiene poblaciones aleatorias de posibles soluciones. Cuando el algoritmo resuelve un problema en particular, maximiza una función meta que consiste en encontrar una solución con un alto valor de evaluación. Cuando el algoritmo termina, su población contiene soluciones que son muy similares entre sí. Existirán soluciones con alto valor de evaluación y otras soluciones con un valor no tan alto. Para el problema en particular, se considera solamente la solución con el valor de evaluación más alto. Generalmente las demás soluciones se desechan.

Si por alguna circunstancia el problema cambia ligeramente ya sea en parámetros, condiciones, modelos, etc., y la representación empleada es válida para el nuevo problema, entonces podemos mantener la población que se usó en el problema anterior (suponemos que el problema actual es similar al anterior) y ejecutar el algoritmo evolutivo. Lo que puede suceder son dos situaciones. Primero, las soluciones que antes tenían un alto valor de evaluación, es muy probable que su valor de evaluación disminuya con el nuevo problema. Segundo, las soluciones que no tenían un valor de evaluación alto, es posible que su valor disminuya más con el nuevo problema, o que su valor aumente, inclusive más que las soluciones que anteriormente tenían un alto valor.

La segunda situación muestra una característica muy importante de la exaptación que es la reutilización de estructuras o soluciones que resultaron de resolver problemas anteriores similares al actual. Es importante destacar que esta situación sólo es útil si el cambio en el problema no es muy alto; en todo caso será necesario incluir algún mecanismo de retención (una memoria) para no perder las soluciones de problemas que se resolvieron con anterioridad y que no tienen semejanza con el problema actual. En el capítulo 9 se revisarán algunos algoritmos que usarán esta forma de exaptación.

6.6 Conclusiones

Este capítulo revisa el proceso evolutivo de la exaptación y se trata de extraer y separar sus funcionalidades para representarlas en procedimientos identificables. Se encontró que en la exaptación existen procesos de reconocimiento, búsqueda o modificación, y almacenamiento. Los procedimientos de la exaptación tienen una similitud con el aprendizaje por analogía, sólo que se usan algoritmos evolutivos en lugar de las herramientas de búsqueda y modificación que son usados por los otros sistemas. Se propone también el desarrollo de un sistema exaptivo basado en algoritmos evolutivos que retienen su población para mantener de alguna forma las estructuras que les fueron útiles en el pasado para reutilizarlas en nuevos problemas. Los algoritmos para desarrollar sistemas exaptivos se mostrarán con detalle en el capítulo 8 y se aplicarán a partir del capítulo 9.

Capítulo 7

Clasificación y evaluación de las técnicas de sembrado

7.1 Introducción

Los algoritmos evolutivos son una emulación a los procesos propuestos por la teoría de la evolución, en donde existen procedimientos de evaluación, selección, y variación. Los algoritmos evolutivos usan poblaciones de individuos donde cada individuo de la población representa a una posible solución de un problema específico. Los algoritmos evolutivos tradicionalmente inician con soluciones que se generan en forma aleatoria; por lo tanto, el tiempo que tarda el algoritmo en encontrar una solución aceptable generalmente es alto y más aún cuando se trata de encontrar la solución óptima. Sin embargo, se puede disminuir el tiempo de búsqueda si de antemano se tiene una idea de la solución y se incluye en la población inicial.

El sembrado consiste en la inserción una o varias soluciones en la población de un algoritmo evolutivo. Es más común realizar el sembrado al principio; sin embargo, puede ejecutarse en cualquier momento, por ejemplo, cuando se pierde diversidad en la población durante la ejecución del algoritmo. Es importante aclarar que las técnicas empleadas aquí serán aplicadas a los algoritmos genéticos; sin embargo, no quiere decir que las técnicas no puedan ser aplicadas a otros algoritmos evolutivos como los algoritmos genéticos con genes reales, las estrategias evolutivas y la programación evolutiva.

Hay evidencia empírica que el sembrado es una técnica que funciona; sin embargo, no existe un análisis sobre su desempeño. El primer trabajo que se tiene referencia del uso de las técnicas de sembrado es de Grefenstette, en donde inserta soluciones que resultan al aplicar un método *gready* para el problema del TSP (*Travelling Salesman Problem*). Sustituye desde el 10% al 100% de la población inicial. La técnica le permite explorar espacios de búsqueda sugeridos por las soluciones sembradas. Al final el autor sugiere el uso de un algoritmo de búsqueda local para un ajuste fino de los resultados. Es importante destacar que Grefenstette menciona que el sembrado debe usarse con precaución para evitar una convergencia prematura en el algoritmo evolutivo (Grefenstette, 1987). De aquí en adelante se han sugerido infinidad de técnicas para sembrar

soluciones al inicio de un algoritmo genético y además, son muy similares entre sí; sin embargo, solo se revisarán las que a criterio personal son las más destacadas.

Darwen y Yu proponen insertar soluciones obtenidas por experiencia (Darwen y Yao, 1995; Yu y Bentley, 1998). Chan usa el sembrado de varias soluciones que se obtuvieron con métodos heurísticos (Chan y Hu, 2000). Potter ejecuta varios algoritmos genéticos para buscar soluciones a problemas básicos, luego se une cada población resultante en una más grande para buscar soluciones a problemas más complejos. Generalmente son problemas que pueden ser divididos en problemas más simples. Lo que se realiza es la siembra de poblaciones con individuos que representan soluciones básicas (Potter, De Jong, y Grefenstette, 1995).

Sushil es el autor que más ha empleado las técnicas de sembrado, en especial el sembrado de poblaciones de soluciones (Louis y Xu, 1996; Louis, 1997; Louis y Johnson, 1997; Louis y Li, 1997; Louis y Johnson, 1999). El sembrado se apoya en un sistema de razonamiento basado en casos ya que dependiendo de la similitud que exista en el problema actual respecto a otros resueltos en el pasado, establece las soluciones que se sembrarán en la población del algoritmo genético. Otra forma de sembrado consiste en insertar fórmulas o programas en la población inicial. Algunos trabajos sobre programación genética sugieren este tipo de sembrado y se han reportado que se obtienen mejores resultados que si se empiezan con poblaciones aleatorias (Koza, 1990, 1993, 1994).

Schoenaver y Xanthakus proponen una forma de reutilizar poblaciones para resolver problemas de optimización con restricciones. El proceso que proponen tienen dos fases. En la primera fase se usa un algoritmo genético con una población inicial aleatoria para evolucionar una población de posibles soluciones con una función de evaluación relacionada con la satisfacción de restricciones. En la segunda fase se toma la población resultante de la primera fase como la población inicial de un segundo algoritmo genético, en donde se incluye la función objetivo que se desea minimizar. Si existe alguna solución que no satisface las restricciones, entonces se le asigna un valor de evaluación de cero. Para mantener la diversidad se usan métodos de compartición de la evaluación (*sharing*) (Schoenauer y Xanthakis, 1993).

C. Henrik Westerberg y John Levine usan programación genética para crear un planeador, es decir, un sistema que fabrica un plan para un determinado problema. Los autores comparan el desempeño del algoritmo cuando inicia bajo diversos esquemas o estrategias de sembrado. Las estrategias son básicamente variaciones parciales de una solución pero son suficientes para superar a un algoritmo que inicia en forma aleatoria (Westerberg y Levine, 2001).

Ciesielski usa varias soluciones para sembrarlas en la población inicial de un algoritmo genético. Cada solución es útil en un momento determinado del problema (Vic Ciesielski, 1998). En el trabajo de Sharif se describe una técnica de sembrado en donde en la población inicial se insertan mutaciones de una solución. Las mutaciones sólo se aplican a una parte del cromosoma ya que se quiere reutilizar el resto de la información del mismo (Sharif y Barrett, 1998).

Enseguida se establecerá una clasificación de las técnicas de sembrado que se han aplicado en diferentes trabajos y se usará un seudocódigo para su representación.

Posteriormente se mostrarán tres formas de representación de problemas y la manera en que se puede reutilizar una solución cuando estos problemas cambian. Finalmente se muestra una comparación entre el algoritmo genético simple y el mismo algoritmo pero que usa cada una de las técnicas de sembrado.

7.2 Técnicas para la reutilización de soluciones en algoritmos evolutivos

Para sembrar necesitamos tener una o varias soluciones, ya sea por una aproximación de la solución óptima, el resultado de una regla heurística, el conocimiento de un experto o generalmente la solución que se obtiene de un problema similar al que se resuelve. Cualquiera que sea la forma, generamos una o más soluciones nuevas que podemos sembrar en el algoritmo evolutivo. No existe un estudio claro sobre la eficiencia que puede brindar a algún algoritmo evolutivo que usa una técnica de sembrado; tampoco existe una clasificación concreta de las técnicas de sembrado. Por las técnicas que se revisaron en la sección anterior, se propone la siguiente clasificación. Las formas de sembrado pueden ser el sembrado elitista, el sembrado variacional elitista y el sembrado poblacional.

El *sembrado elitista* consiste en insertar la mejor solución que se tenga en la población del algoritmo evolutivo. Por ejemplo, si tenemos una solución V y tenemos una población P de n individuos, podemos reemplazar el primer individuo por la solución, es decir, $P(1) = V$.

El *sembrado variacional elitista* consiste en alterar la solución de una o más formas para tener una población que contengan variaciones de la solución. Es llamado variación elitista porque usamos variaciones de una solución que se considera la mejor. Las variaciones de una solución las podemos realizar por medio de mutaciones de la solución en forma total o parcial. Esto significa que podemos alterar todo el cromosoma o sólo una fracción de él. Si consideramos que la solución se representa por medio de un vector V; entonces, una mutación total consiste en elegir cada uno de los elemento del vector y mutar probabilísticamente su valor. Una mutación parcial consiste en elegir determinadas posiciones del vector y mutar probabilísticamente su valor.

Una mutación cambia cada valor con una probabilidad predefinida. Si queremos aplicar una mutación al vector V, primero elegimos una posición, luego cambiamos el valor de esta posición, siempre y cuando la probabilidad para mutarlo $p_m >$ aleatorio$(0,1)$. La probabilidad de mutación p_m generalmente es alta. Si es muy alta en lugar de tener un sembrado variacional se estarían sembrando individuos opuestos a la solución y por lo tanto, tendríamos una población que podría converger muy rápido. Si es muy baja (casi cero), se puede tener un sembrado poblacional elitista (se siembra la misma solución varias veces). En forma experimental se ha determinado que la probabilidad de mutación no puede ser mayor que 0.5 ya que un valor más grande deja de tener sentido porque se obtienen valores opuestos a la solución.

Podemos tomar la probabilidad de mutación p_m como un valor constante para todos los individuos de la población; otra forma consiste en incrementar proporcional-

mente la probabilidad con cada individuo, así el primer individuo tendrá una probabilidad de mutación de 0, el segundo individuo tendrá una probabilidad de $2/(2\,N_{TI})$ y así sucesivamente hasta que el último individuo tenga una probabilidad de mutación igual a $N_{TI}/(2\,N_{TI})$ lo cual equivale a 0.5. Con las mutaciones totales o parciales de la solución, podemos generar a varios individuos que podemos sembrar en la población. De la relación entre los individuos sembrados y el tamaño de la población, obtenemos el porcentaje de sembrado. Por ejemplo, si tenemos una población de 20 individuos y sembramos 10 individuos (resultado de la variación elitista), entonces tendremos un sembrado del 50 por ciento. El resto de los individuos son aleatorios.

El *sembrado poblacional* consiste en insertar un conjunto de soluciones en la población. Las soluciones, como hemos mencionado, pueden ser el resultado de infinidad de métodos. Un método para generar soluciones a partir de una es crear vecindades a su alrededor. Generalmente la vecindad es pequeña ya que si es muy grande, sería una variación elitista. El procedimiento para la generación de vecindades depende mucho de la representación del problema usado, así que no es posible definir alguna en particular. Por ahora se puede decir que el vector binario V_V es vecindad de V si $\|(V - V_V\|) < d$ donde el parámetro d es la distancia de Hamming que existe entre ambos vectores y generalmente es un valor pequeño. Una alternativa es manejar el mismo concepto de distancia pero para parámetros decodificados a números reales. La distancia es ahora euclidiana. Se generan soluciones que se encuentren dentro de esta distancia y enseguida son codificados a vectores binarios para que puedan sembrarse en la población inicial del algoritmo genético (si se usan otros algoritmos evolutivos, la codificación no es necesaria.) Otra forma de reutilizar soluciones consiste en sembrar porciones de poblaciones que fueron el resultado de un algoritmo evolutivo que fue empleado para buscar soluciones a problemas similares al que tenemos en el momento. También se manejan porcentajes de sembrado; al igual que el sembrado variacional elitista. Por ejemplo, si tenemos 20 soluciones diferentes y al sembrarla en una población de 100 individuos, se tiene un porcentaje de sembrado al 20%. El seudocódigo de la figura 7.1 muestra genéricamente cómo se aplica alguna técnica de sembrado. El algoritmo inicia con una población aleatoria (procedimiento Generación) y posteriormente en el procedimiento de Sembrado se insertan nuevos individuos en la población inicial aplicando alguna de las tres técnicas de sembrado.

Puede notarse en el seudocódigo de la figura 7.1 que el sembrado sólo se aplica al inicio del algoritmo genético; sin embargo, como se mencionó puede aplicarse en cualquier momento de la ejecución.

7.3 Reutilización de soluciones en tres tipos de problemas

Para evaluar a cada técnica de sembrado se considerarán tres tipos de problemas (Michalewicz y Fogel, 1999). La primera representación es para los problemas de satisfactibilidad (SAT) donde tenemos n variables binarias. Las soluciones se representan por medio de vectores binarios de tamaño n. La segunda representación es para problemas

1) $(F_E, P) \leftarrow$ Generación$(Ciclos, N_{TG})$
2) $P \leftarrow$ Sembrado(P, t_p)
3) $P \leftarrow$ Evaluación(P, t_p)
4) $P \leftarrow$ Selección(P)
5) $P \leftarrow$ Cruce(P, p_c)
6) $P \leftarrow$ Mutación(P, p_m)
7) Si es fin de algoritmo terminar, sino ir a (3).
t_p Corresponde al tipo de sembrado.

Figura 7.1: Seudocódigo de un algoritmo genético simple con técnicas de sembrado.

cuyas soluciones relacionen permutaciones de números naturales $1 \ldots n$ donde n es el número de entidades del problema, por ejemplo, el número de ciudades para el problema del vendedor viajero. La tercera representación es para problemas de optimización donde la solución consiste en todos los números reales en n dimensiones. La solución se representa con un vector de números reales. Cuando se usa una representación binaria el espacio real es fraccionado en 2^b partes donde b es el tamaño del vector binario que se usará para representar una dimensión del espacio real. Enseguida se revisarán a detalle los tres problemas que se usarán en la experimentación.

7.3.1 El problema de la mochila

En el problema de la mochila se busca una combinación de objetos que maximicen un costo aprovechando lo mejor posible la capacidad de la mochila. Cada objeto tiene un peso y un costo, por lo cual podemos definir un vector binario V donde "1" indica la presencia del objeto en la mochila y con un "0" su ausencia. El peso total de la mochila se calcula como:

$$P_T = \sum_{i=1}^{N_{TO}} (P(i) \cdot V(i)) \quad (7.1)$$

donde N_{TO} corresponde al total de objetos y P es el peso que tiene cada objeto i. El costo total de todos los objetos que se están llevando en la mochila se calcula como:

$$C_m = \sum_{i=1}^{N_{TO}} (C(i) \cdot V(i)) \quad (7.2)$$

Se busca maximizar el peso P_T respetando la capacidad de la mochila y aproximandose lo mejor posible. Para evaluar cada solución V se definen las siguientes funciones de evaluación. La función de evaluación del peso total es:

$$f_{ep} = \begin{cases} 1 - ((C_m - P_T)/C_m) & P_T \leq C_m \\ 0 & P_T > C_m \end{cases} \quad (7.3)$$

donde C_m es la capacidad de la mochila. La función de evaluación de costo es:

$$f_{ec} = C_m / M_c \quad (7.4)$$

donde M_c es el máximo costo esperado. La evaluación neta es:

$$f_e = 0.5\, f_{ep} + 0.5\, f_{ec} \quad (7.5)$$

Al inicio se considerará una solución V tomando los pesos y los costos iniciales. Para modificar el problema podemos afectar la capacidad de la mochila, pero con un cambio en algunos pesos o en algunos costos es suficiente porque puede implicar buscar una nueva combinación de objetos. Se usará un cambio en el 30% de los objetos tanto en sus pesos como en sus costos que se generan en forma aleatoria entre 1 y 10 unidades de manera uniforme. Siempre se modifican los mismos objetos, asi que los demás objetos permanecen sin alteración.

El sembrado elitista se ejecuta insertando la solución V en una población inicial aleatoria. La solución V sustituye a cualquier individuo de la población. En el sembrado variacional se usan mutaciones del vector V que sustituyen al 50% de la población inicial aleatoria. Para generar las mutaciones simplemente se revisa la probabilidad de mutación de cada bit. Si se cumple la mutación entonces el bit se modifica por su complemento.

El sembrado poblacional es un poco diferente porque es necesario generar vecindades alrededor de la solución V porque sólo tenemos una. El generador de vecindades se basa en la distancia de Hamming d; se establece una distancia y posteriormente se toma el vector V para alterarlo en posiciones aleatorias en d cantidad de bits. Con el procedimiento aseguramos que se generarán vectores vecinos de V con una distancia máxima d. Pueden obtenerse vectores de una distancia menor porque existe la posibilidad de alterar el mismo bit.

7.3.2 La optimización de funciones cambiantes

Para el problema de optimización de funciones cambiantes se usará una función cuyo valor máximo depende directamente de un parámetro v_r, asi que cada vez que se altere este parámetro, también se alterará el valor máximo de la función. La función tiene dos argumentos que tienen un rango de 0 a 1, y se codificarán con ocho bits. El tamaño del vector binario que representará a las dos variables será de 16 bits. (8 bits para cada variable). La función se compone del producto de una función sinoidal con una envolvente exponencial decreciente y multiplica a una función que resulta del máximo de varias funciones gaussianas.

$$f_{gauss}(x, b, c) = \exp\left(-\left(\frac{x-c}{b}\right)^2\right) \quad (7.6)$$

$$f_1(x, v_r) = \exp^{-7x}\,\mathrm{sen}(v_r\, x) + 1 \quad (7.7)$$

$$f_{2a}(x, v_r) = \max(1.0\, f_{gauss}(x, 0.1, v_r), 0.82\, f_{gauss}(x, 0.2, 0.25)) \quad (7.8)$$

$$f_{2b}(x) = \max(0.52\, f_{gauss}(x, 0.2, 0.5), 0.92\, f_{gauss}(x, 0.2, 0.85)) \quad (7.9)$$

$$f_2(x)(x, v_r) = \max(f_{2a}(x, v_r), f_{2b}(x)) \quad (7.10)$$

$$f(x_1, x_2, v_r) = f_1(x_1, v_r)\, f_2(x_2, v_r) \quad (7.11)$$

El parámetro v_r modifica la forma de la función y su valor óptimo. Si el parámetro v_r toma el valor de 0.5, el valor óptimo se alcanza cuando $x_1 = 0.0118$ y $x_2 = 0.5$. El parámetro v_r cambiará cada vez que se use la función en un algoritmo y permanecerá sin cambio durante el funcionamiento del mismo. El cambio consiste en una alteración aleatoria de la variable v_r como $v_r \leftarrow 0.5 + 0.2$ (aleatorio(0,1) – aleatorio(0,1)); el valor constante de 0.2 corresponde al impacto de cambio que tendrá el parámetro v_r alrededor de 0.5. El diseño de la función permite generar varios picos cuya posición se ajusta por medio del parámetro v_r, se buscó una función relativamente sencilla de optimizar pero que tuviera un grado de cambio y sobre todo que sea evidente localizarlo. El objetivo es encontrar el valor de las variables que maximice la función cada vez que cambie, en donde será muy importante usar la información pasada para alcanzar el valor óptimo con el menor número de evaluaciones. Todos los algoritmos reutilizarán la solución $S = [0.0118, 0.5]$ que codificado corresponde al vector binario $I = [0111110100000011]$; de aquí en adelante se hará referencia a la solución reutilizada como S y a la solución codificada como vector I. La forma que tiene la función se muestra en la figura 7.2.

Considerando a la población P como una matriz de tamaño N_{TI} (total de individuos) y N_{TC} (Tamaño del cromosoma) tenemos, para el sembrado elitista insertaremos directamente el vector solución I en la población inicial P; sustituiremos el primer individuo de esta población por el individuo o vector I, es decir, $P(1, j) = I(j)$ donde: $j = 1 \ldots 16$ que corresponde a los bits del cromosoma.

El sembrado variacional consiste en un cambio aleatorio de la solución codificada I y su inserción en la población. El sembrado es del 50%, como la población es de 10 individuos, insertaremos 5 individuos. Los individuos pueden ser consecutivos o aleatorios. Se seleccionarán a los individuos en forma consecutiva. En el algoritmo de sembrado variacional es necesario definir el parámetro de mutabilidad por bit p_m ya que establecen la variación como constante o como proporcional; si es en forma constante los valores que se usan para p_m son 0.05, 0.1, 0.15, y 0.2. Si es en forma proporcional entonces se define el parámetro $p_m = i/10$ donde $i = 1 \ldots 5$. En ambos casos tenemos que el sembrado de mutaciones de I se obtiene como $I_m = Mutar(I, p_m)$ en el cual se insertan en la población como $P(i, j) = I_m(i, j)$ donde: $j = 1 \ldots 16$, $i = 1 \ldots 5$ para sembrar el 50%.

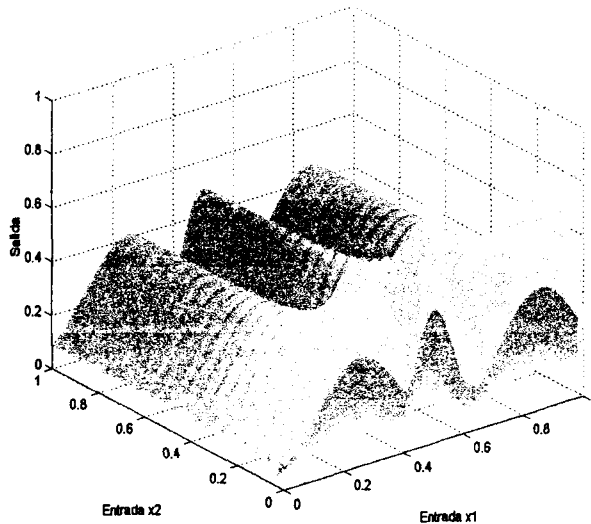


Figura 7.2: Función a optimizar con $v_r = 0.5$.

El procedimiento de mutación $Mutar(I, p_m)$ consiste en alterar un vector binario; cada elemento es alterado de acuerdo a una probabilidad de mutación p_m que puede permanecer constante o puede variar en forma proporcional como $p_m = k/2\,N_{TI}$ donde N_{TI} es el total de individuos a sembrar y $k = 1 \ldots N_{TI}$. El valor máximo deseado para la probabilidad de mutación es de 0.5 y será la probabilidad que tendrá el individuo N_{TI}.

El sembrado poblacional requiere de varias soluciones. Para generar varias soluciones similares a partir de una, usaremos un algoritmo generador de vecindades. Para el sembrado poblacional $P(i,j) = I_v(j)$ donde $j = 1 \ldots 16$, I_v es una variación de la solución I. El parámetro $i = 1 \ldots 5$ para sembrar el 50%. El procedimiento para generar vecindades consiste en tomar el vector solución I como entrada. Al decodificar el vector I obtenemos el valor real de las variables en un vector X. A cada variable se les añade una cantidad aleatoria con una distancia predeterminada d; es decir, $X_V = d \cdot aleatorio(-1, 1) + X$. Se usarán distancias $d = 0.01$, $d = 0.05$, $d = 0.1$, y $d = 0.15$. El vector X_V es una vecindad decodificada del vector X. El vector X_V se codifica a un vector binario I_v y es sembrado en la población inicial (figura 7.3).

7.3.3 La programación de tareas

La programación de tareas consiste en conjunto de máquinas y en una lista de tareas con tiempos de ejecución, de compromiso (en que se debe entregar la tarea ya realizada) y pesos o prioridades. Una tarea que tenga un peso alto debe ser entregada antes que una tarea de peso bajo si ambas tienen el mismo tiempo de entrega. Cada tarea se

1) $X \leftarrow$ Decodificación(I)
2) $X_V \leftarrow X + d(\text{aleatorio}(-1,1))$
3) $I_v \leftarrow$ codificación(X_V)
donde:
I es un vector binario de la solución codificada.
d es la distancia de la vecindad
X es el vector binario de las variables x_1 y x_2
X_V es una vecindad del vector binario X.
I_v es la vecindad X_V codificada a un vector binario.

Figura 7.3: Generación de vecindades.

ejecuta en una sola máquina y cualquier máquina puede tomar cualquier tarea. El problema consiste en determinar el orden óptimo en que se deben asignar las tareas a una máquina en particular. La máquina se asignará a cada tarea dependiendo de la disponibilidad para ejecutar la tarea, es decir, se escogerá a la máquina que pueda acabar la tarea en el menor tiempo considerando las asignaciones anteriores que tenga cada máquina.

Para determinar que un orden es adecuado, es necesario obtener una función de evaluación. Se considera que H_E es el tiempo compromiso de la entrega, P_S es el peso por tarea, y T_E es el tiempo en que se entregará la tarea terminada, entonces tenemos que dado un orden predefinido O para cada tarea (en términos logísticos es el pedido pd y N_p es el total de tareas o de pedidos), el tiempo de tardanza para cada pedido es:

$$L(O(pd)) = T_E(O(pd)) - H_E(O(pd)) \tag{7.12}$$

Si el valor de L es positivo, se tiene un retraso. Si es negativo es un indicativo de que se está terminando la tarea antes del tiempo compromiso. El tiempo de retraso total T se calcula como sigue:

$$T(O(pd)) = \max(L(O(pd)), 0) \tag{7.13}$$

donde $pd = 1 \ldots N_p$. Todo retraso es penalizado por un factor o peso, por lo cual, cada vez que exista un retraso es necesario acumularlo en una variable e. Al inicio e tiene un valor de cero, cada pedido o tarea se lee dependiendo del orden que se establezca en el vector O. El problema se reduce a encontrar una secuencia O de tareas o pedidos que minimizen el valor de e (Pinedo, 1995). La ecuación 7.14 permite evaluar cada secuencia sugerida en la lista O.

$$e = \sum_{pd=1}^{N_p} P_S(O(pd))\, T(O(pd)) \tag{7.14}$$

Para la programación de las tareas se considera el uso de un algoritmo genético con operadores especiales de cruce y mutación porque los mecanismos de cruce y mutación

que se usan para cromosomas binarios, pueden dar soluciones inválidas (Goldberg, 1989; Ordoñez, 1991). Se diseñó un mecanismo sencillo de cruce de secuencias llamado de intercambio. El cruce de intercambio consiste en elegir una posición aleatoria para ambas secuencias y considerar las tareas en estas posiciones. El siguente paso consiste en buscar las tareas elegidas en la secuencia contraria e intercambiar sus valores de esta posición a la posición aleatoria. Por ejemplo si tenemos dos secuencias:

[1 2 3 4 5 6 7]
[3 4 1 7 2 6 5]

y si suponemos que la posición aleatoria es dos, entonces para la primera secuencia corresponde a la tarea 2 y para la segunda secuencia corresponde a la tarea 4. Buscamos la tarea 2 de la primera secuencia en la segunda secuencia. La localizamos en la quinta posición. Se realiza un intercambio de tareas entre esta posición cinco y la posición aleatoria dos. El mismo procedimiento se realiza para la primera secuencia, es decir, localizamos la tarea 4 en la primera secuencia y después intercambiamos las tareas entre la posición en que se encuentra la tarea 4 (cuarta posición) hasta la tarea de la posición dos. Las secuencias que se generan con el cruce son:

[1 4 3 2 5 6 7]
[3 2 1 7 4 6 5]

El problema cambia cuando algunas características de las tareas o de las máquinas son modificadas. En nuestro caso se realiza un cambio en un 30% de las tareas, esto quiere decir que el tiempo compromiso, el tiempo de elaboración y los pesos de cada tarea son alterados sustituyéndolo por nuevos tiempos y pesos generados aleatoriamente de manera uniforme. El resto de las tareas permanecen sin cambio.

Para aplicar el sembrado elitista simplemente se elige la mejor solución encontrada (ya adaptada con las nuevas tareas) y se incluye en la población inicial sustituyéndolo por cualquier individuo. La población inicial siempre es aleatoria.

El sembrado variacional de una secuencia solución requiere de un procedimiento de variación. La variación de una solución consiste en elegir aleatoriamente dos posiciones e intercambiar sus valores, por ejemplo dada la secuencia:

[1 2 3 4 5 6 7]

si se eligen las posiciones 3 y 7 e intercambiamos sus valores, tenemos una nueva secuencia:

[1 2 7 4 5 6 3]

En número de veces en que se eligen posiciones aleatorias será el grado de variabilidad establecida; por ejemplo, si se elige un grado de variabilidad de tres entonces se toman tres veces dos posiciones de la secuencia para intercambiar sus valores.

Para el sembrado poblacional es necesario generar vecindades de una secuencia de tareas. Una forma de establecer una vecindad de una secuencia, consiste en localizar

una tarea k con la máquina m (llamese posición p_1) y posteriormente buscar la siguiente tarea l que tenga la misma máquina m (llámese posición p_2) (Pinedo, 1995). Enseguida se intercambian los índices de las tareas entre ambas posiciones para tener una nueva secuencia. Por ejemplo, dadas las secuencia de tareas T con las respectivas máquinas asignadas por tarea m:

$$T = [4\ 5\ 1\ 2\ 3\ 6\ 7]$$
$$M = [1\ 2\ 1\ 2\ 1\ 2\ 1]$$

Si suponemos que $p_1 = 2$ entonces la tarea $T(p_1) = 5$ y le correspone a la máquina $M(p_1) = 2$. La siguiente tarea que usa la máquina 2 es la número 2 y se localiza en la posición $p_2 = 4$ de la secuencia T. Al intercambiar las tareas tenemos una nueva secuencia entre las posiciones p_1 y p_2:

$$T' = [4\ 2\ 1\ 5\ 3\ 6\ 7].$$

7.4 Experimentación y resultados

En la experimentación se quiere comparar el desempeño de un algoritmo genético simple que inicia en forma aleatoria con el mismo algoritmo que reutiliza una solución. Se inicia con un problema generado en forma aleatoria y se resuelve con un AGS sin sembrado; la solución se almacena y se reutilizará por todos los algoritmos que usan una técnica de sembrado. El problema se cambia ligeramente y se busca una solución usando un AGS, un AGS con sembrado elitista, un AGS con sembrado variacional elitista y un AGS con sembrado poblacional. Los pasos de la experimentación se resumen enseguida:

1. Se establece el problema inicial.
2. Se busca una solución del problema usando un AGS sin sembrado.
3. El problema inicial cambia ligeramente.
4. Se busca una solución del problema usando un AGS sin sembrado
5. Se busca una solución del problema usando un AGS con cada una de las técnicas de sembrado.

Las características del algoritmo genético simple dependen del tipo de problema. La tabla siguiente resume las características de cada algoritmo en donde difieren ligeramente unos respecto de otros por la naturaleza del problema que están resolviendo. N_{TI} es el total de individuos, N_{TG} es el tamaño del cromosoma, T_{Gen} es el total de generaciones, p_c corresponde a la probabilidad de cruce, p_m es la probabilidad de mutación. Todos los algoritmo usan una selección de torneo de tamaño 2. Todas las figuras muestran el desempeño promedio de cada algoritmo durante 100 ejecuciones. La figura 7.4 muestra el desempeño de todos los algoritmos en la optimización de una función cambiante. Cada vez que se usa una técnica de sembrado, existe una ventaja respecto

Tabla 7.1: Características de cada algoritmo.

Problema	N_{TI}	N_{TG}	p_c	p_m	tipo de cruce	T_{Gen}
Prog. tareas	100	100	0.9	0.052	intercambio	50
Opt. funciones	10	100	0.8	0.052	uniforme un punto	50
Prob. mochila	100	100	0.8	0.052	uniforme un punto	250

al algoritmo genético simple. Por la gráfica podemos observar que el sembrado poblacional tiene una ligera ventaja respecto al resto de los algoritmos. La figura 7.5 muestra el desempeño de todos los algoritmos en la programación diaria de tareas. A pesar de la diversidad de problemas que pueden generarse, el uso de una técnica de sembrado es una ventaja sobre el algoritmo genético simple. Se resalta la ventaja de la técnica variacional elitista sobre el poblacional y el elitista. Finalmente la figura 7.6 muestra el desempeño de todos los algoritmos en el problema de la mochila con la evidente ventaja que le ofrece la técnica del sembrado comparado con el algoritmo genético simple.

7.5 Conclusiones

Cuando se usa un algoritmo genético sin ninguna información, su desempeño será menor que cuando lo iniciamos con algún conocimiento anterior sobre el problema que se resuelve. Se demostró que las técnicas de sembrado sí brindan una ventaja para encontrar soluciones en menos tiempo y con alta calidad que si usamos un algoritmo que empieza en forma aleatoria. Es importante aclarar que la variabilidad del problema es muy importante ya que si es muy alta, entonces es muy probable que se tenga convergencia prematura y la ventaja se pierde. Se supone que la variabilidad no es tan alta, por lo cual nos permite reutilizar las soluciones anteriores en cierta grado y con ello sacar alguna ventaja sobre el algoritmo que inicia la búsqueda en forma aleatoria. Es necesario investigar cómo afecta la variabilidad de un problema respecto a la técnica de sembrado utilizada, es decir, en algún momento el problema puede tener un cambio tan significativo que tal vez no sea conveniente reutilizar una solución, sino empezar en forma aleatoria y tratarlo como un problema completamente nuevo.

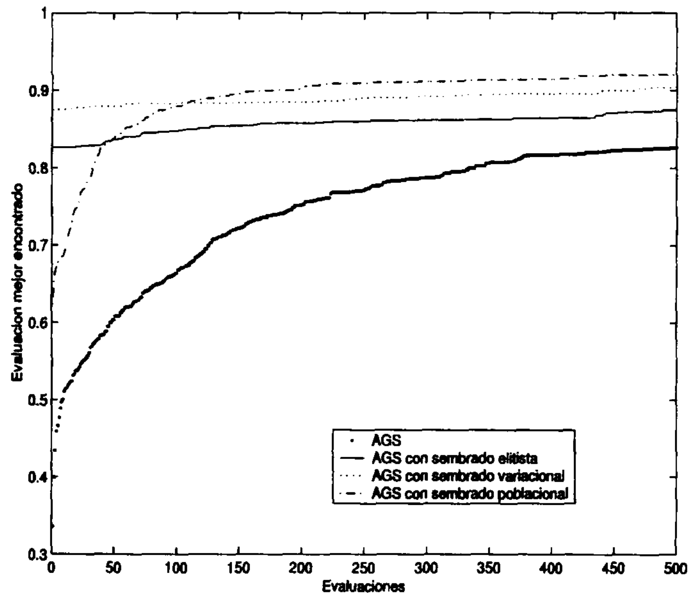


Figura 7.4: Desempeño en la optimización de una función cambiante de tres AGS's que utiliza sembrado contra un AGS que no lo usa.

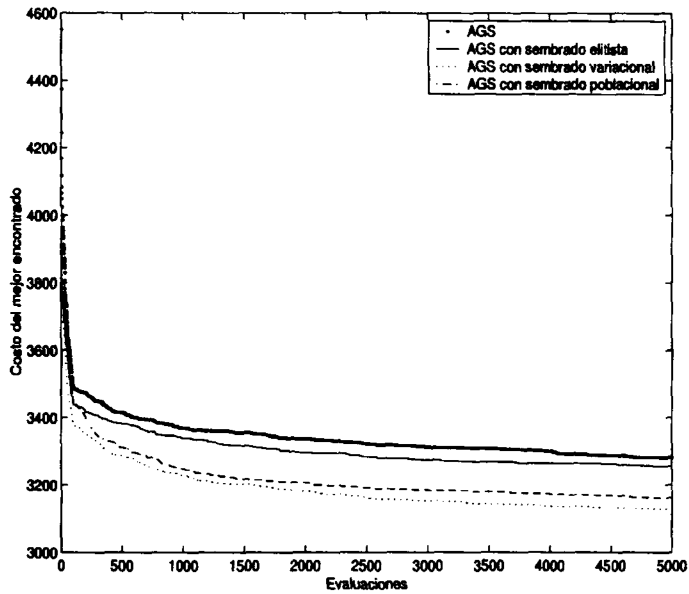


Figura 7.5: Desempeño en la programación de tareas con reutilización de la solución anterior de tres AGS's que utiliza sembrado contra un AGS que no lo usa.

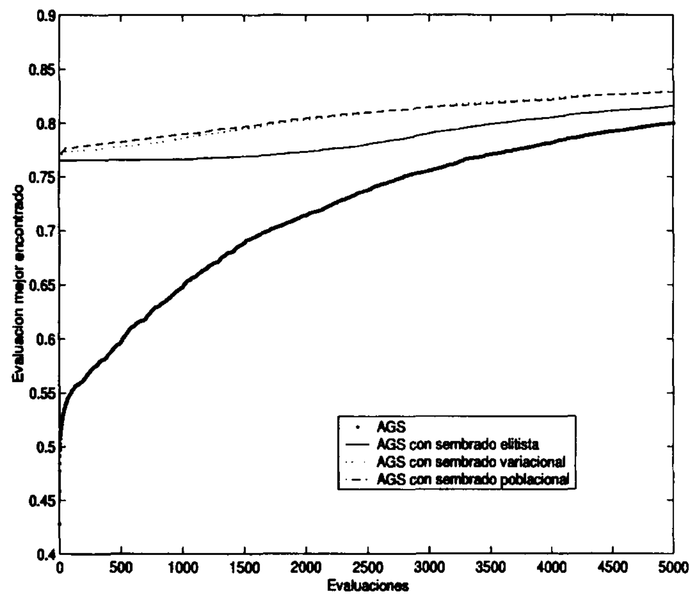


Figura 7.6: Desempeño en la optimización del problema de la mochila cambiante de tres AGS's que utiliza sembrado contra un AGS que no lo usa.

Capítulo 8

Reutilización del conocimiento en la mutación dirigida

Existen dos formas de generar soluciones en los algoritmos evolutivos. Una forma consiste en seleccionar individuos de una población y realizar procedimientos de cruce y mutación entre ellos (los individuos son representaciones de posibles soluciones). Otra forma consiste en extraer información de un conjunto de posibles soluciones con la finalidad de generar nuevas soluciones. La primera forma se usa, por ejemplo, por los algoritmos genéticos, las estrategias evolutivas y la programación genética. La segunda forma se usa por los algoritmos de PBIL (Population Based Incremental Learning), CGA (Compact Genetic Algorithm) y BOA (Bayesian Optimization Algorithm) que se revisarán en la primera sección. Se propone en este capítulo el algoritmo de la mutación dirigida (MD) y la forma en que puede reutilizar soluciones en estos algoritmos. La mutación dirigida tiene una estructura muy similar a las usadas por los algoritmos de PBIL y CGA; sin embargo, la forma de funcionamiento es diferente. Lo importante es resaltar cómo se pueden reutilizar soluciones en estos tipos de algoritmos, haciendo énfasis a la mutación dirigida.

8.1 Algoritmos PBIL, CGA y BOA

El algoritmo de PBIL es una propuesta de Shumeet Baluja en donde se representa a una población de individuos por medio de un vector de probabilidades. Cada elemento del vector representa la probabilidad para que un bit permanezca en un estado. El vector se usa para generar individuos; enseguida todos los individuos generados son evaluados y ordenados por el valor de la evaluación. Los individuos con las mejores evaluaciones se usan para calcular nuevas probabilidades para cada elemento del vector (Baluja, 1994). El algoritmo se muestra en la figura 8.1.

El algoritmo de CGA es muy similar al algoritmo de PBIL; también usa un vector de probabilidades para representar la tendencia probable promedio de una población. Por medio del vector de probabilidades, se generan dos individuos que compiten entre sí. El individuo ganador determina cómo debe modificarse el vector de probabilidades. El

1) Inicio del vector de probabilidad.
$V_P(j) \leftarrow 0.5$ donde $j = 1 \ldots \lambda$
2) Generación de n individuos I con el vector V_P.
$I(i,j)$ =generación(V_P) donde $i = 1 \ldots n$ y $j = 1 \ldots \lambda$
3) Evaluación de cada individuo.
$F_E(i) \leftarrow$evaluación($I(i,j)$) donde $i = 1 \ldots n$ y $j = 1 \ldots \lambda$
4) Ordenamiento de cada individuo de acuerdo a la aptitud F_E.
5) Actualización del vector de probabilidades.
$V_P(i) \leftarrow V_P(i) * (1 - l_r) + I(i,j) * l_r$
donde $i = 1 \ldots n_v$ y $j = 1 \ldots \lambda$
6) Si es fin del algoritmo entonces terminar sino ir a (2)
Parámetros:
n es el número de individuos generados por iteración.
λ es el tamaño del cromosoma o tamaño del vector de probabilidades.
n_v es el número de individuos usados en el paso de actualización.
l_r es la constante de aprendizaje.

Figura 8.1: Algoritmo PBIL

algoritmo se detiene cuando el vector de probabilidades converge; es decir, tiene valores de 0 ó 1. Se ha demostrado que el algoritmo CGA imita a un algoritmo genético simple con cruce uniforme y sin mutación (Harik, Lobo, y Goldberg, 1999). El algoritmo se muestra en la figura 8.2.

El algoritmo BOA consiste en la generación de una población aleatoria para seleccionar a los mejores individuos; enseguida se construye una red bayesiana con los individuos seleccionados para ajustar y generar nuevos individuos. Los nuevos individuos reemplazarán a una parte de la población inicial. El proceso de construcción de la red, generación y reemplazo se repite varias veces hasta cumplir con algún criterio para terminar el algoritmo. El algoritmo se muestra en la figura 8.3. A diferencia de los anteriores algoritmos, BOA sí brinda mejores soluciones que el algoritmo genético simple en determinados problemas ya que se tiene más información sobre la interacción que existe entre los bits (Pelikan, Goldberg, y Cantú-Paz, 1999).

8.2 La mutación dirigida

El proceso evolutivo de la mutación dirigida (MD) ha sido observado en determinados organismos unicelulares como las bacterias. El concepto de la mutación dirigida es una teoría que intenta explicar la adaptación que tienen ciertos organismos cuando existen cambios bruscos en el medio ambiente. Cairns y sus colaboradores definieron los siguiente pasos experimentales que permiten observar el comportamiento de un cultivo bacterial y con ello, demostrar la existencia de la mutación dirigida (Cairns y Miller, 1988):

1) Inicio del vector de probabilidad.
$V_P(j) \leftarrow 0.5$ donde $j = 1 \ldots \lambda$
2) Generación de dos individuos con el vector V_P.
$I(i,j) \leftarrow$Generación$(V_P(j))$ donde $i = 1 \ldots 2$ y $j = 1 \ldots \lambda$
3) Se determina un ganador y un perdedor entre los dos individuos de I.
$(I_{ganador}, I_{perdedor}) \leftarrow$ Competencia$(I(1,j), I(2,j))$, donde $j = 1 \ldots \lambda$
4) Actualización del vector de probabilidades con el ganador.
Si $I_{ganador} \neq I_{perdedor}$ entonces
Si $I_{ganador}(j) = 1$ entonces $V_P(j) \leftarrow V_P(j) + 1/n$
sino $V_P(j) \leftarrow V_P(j) - 1/n$
donde $j = 1 \ldots \lambda$
5) Se verifica convergencia en el vector de probabilidades.
Si $V_P(j) > 0$ y $V_P(j) < 1$ donde $j = 1 \ldots \lambda$ entonces ir a (2)
6) Fin del algoritmo. V_P representa la solución final.
Parámetros:
n es el tamaño de la población en simulación.
λ es el tamaño del cromosoma o tamaño del vector de probabilidades.

Figura 8.2: Algoritmo CGA

1. En un cultivo bacterial se aplica un cambio en el ambiente (generalmente un cambio en el alimento de las bacterias.)

2. Existe un aumento en la probabilidad de mutación (hipermutación) de las bacterias, es decir, aparecen más mutantes y la diversidad del cultivo aumenta.

3. Los genes que favorecen la aptitud, tendrán alta probabilidad para permanecer entre los individuos.

4. Los genes que no favorecen la aptitud tendrán alta probabilidad para cambiar a otros genes; por lo tanto, los genes que brinden baja aptitud perderán su presencia entre los individuos.

La mutación dirigida contradice un principio básico de la teoría evolutiva porque el organismo dirige la forma de su estructura y no el medio ambiente; por lo cual, se ha propuesto que la mutación dirigida es más bien el resultado de la hipermutación; es decir, existe un aumento de las mutaciones provocando que los individuos menos aptos desaparezcan y los más aptos permanezcan y dan la impresión de que los individuos "saben" qué genes deben cambiar o hacia donde dirigir la mutación.

La mutación dirigida está relacionada con los procesos evolutivos de la variación ciega ya que existe un alto grado de mutabilidad en todos los genes; además hay una retención de las características más favorables. Los genes que favorecen la aptitud del individuo son los que permanecen porque disminuyen su probabilidad de cambio a otros genes; es decir, hay retención de combinaciones de genes que dan alta aptitud.

1) Generación de una población aleatoria P.
2) Selección de individuos I.
3) Construcción de una red bayesiana B con los individuos I
y con una métrica y restricciones predefinidas.
4) Generación de nuevos individuos O de acuerdo a la distribución conjunta
que está codificada en la red bayesiana B.
5) Creación de una población P reemplazando algunos individuos I
por los nuevos individuos O.
6) Si no es fin del algoritmo entonces ir a (2)
sino terminar.

Figura 8.3: Algoritmo BOA

Los genes menos favorables tendrán una alta probabilidad para cambiar posiblemente a genes que brinden una mejor aptitud.

El algoritmo de la mutación dirigida

La mutación dirigida se basa en dos mecanismos, la retención de los mejores individuos y en la capacidad de cambio de cada uno de ellos. El mecanismo de retención se refleja por la probabilidad que tienen un gen de permanecer en un estado o en cambiar a otro más favorable. La representación de esta probabilidad se realiza por medio de un vector en donde cada posición representa la probabilidad de cambio de un gen. También es necesario representar al cromosoma de un solo individuo por medio de un vector binario.

El mecanismo de variación permite calcular las probabilidades de cada gen para cambiar de un estado a otro. El mecanismo de variación consiste en la elección de una posición aleatoria dentro del cromosoma y enseguida realizar una evaluación cuando el bit de esta posición toma el valor de uno y otra evaluación cuando toma el valor de cero. La evaluación más alta determinará la probabilidad del gen para permacener en el estado en que se encuentra o para cambiar a otro estado. El algoritmo de mutación dirigida se muestra en la figura 8.4. La codificación de las posibles soluciones se representan en un vector de bits de tamaño T_C. V_P es un vector que contiene la probabilidad de permanecer en un estado.

El primer paso consiste en definir un valor inicial para el vector del individuo V y el vector de probabilidad V_P. El vector de probabilidades tiene valores de 0.5 para todos los bits, esto quiere decir que cada elemento del vector tiene la misma probabilidad de cambiar ya sea para ser uno o cero. El vector binario V inicia con valores aleatorios. El siguiente paso consiste en elegir una posición aleatoria *pos* entre uno y el tamaño del vector T_C. Esta posición se determina por el vector de probabilidades V_P; por lo cual, las posiciones con valores de probabilidad igual o cercano a 0.5 tendrán alta probabilidad de ser elegidos. Los cercanos a cero o a uno tendrán baja probabilidad para

1) Inicio de un vector de probabilidades
y de un vector solución aleatorio
$V_P(j) \leftarrow 0.5$, $V(j) \leftarrow$aleatorio entre 0 ó 1
donde $j = \ldots N_{TG}$
2) Elección de una posición *pos* con el vector V_P.
$pos \leftarrow elegir(V_P)$
3) Evaluación del vector V en ambos estados.
$f_{e0} \leftarrow$evaluación$(V(pos) \leftarrow 0)$
$f_{e1} =$evaluación$(V(pos) \leftarrow 1)$
4) Cálculo de probabilidades.
$\delta \leftarrow |f_{e1} - f_{e0}|$
si $f_{e1} > f_{e0}$ entonces $V_P(pos) \leftarrow V_P(pos) + \alpha \cdot \delta$
si $f_{e1} < f_{e0}$ entonces $V_P(pos) \leftarrow V_P(pos) - \alpha \cdot \delta$
5) Si $V_P(pos) > 1$ entonces $V_P(pos) \leftarrow 1$
6) Si $V_P(pos) < 0$ entonces $V_P(pos) \leftarrow 0$
7) Si no se cumple un criterio para terminar el cálculo
de probabilidades entonces ir a (2)
8) Cambio de individuo.
Si $V_P(j) >$aleatorio$(0, 1)$ entonces $V(j) \leftarrow 1$
sino $V(j) \leftarrow 0$
donde $j = \ldots N_{TG}$
9) Si es fin del algoritmo ir a (2), sino terminar.
Parámetros:
N_{TG} es el número total de genes o bits.
α es una constante de velocidad del cálculo de probabilidades.

Figura 8.4: Algoritmo de mutación dirigida.

ser elegidos. Enseguida empieza un proceso de evaluación para verificar qué cambio es favorable en los dos estados posibles. Cuando se usa la evaluación de cada estado, se aumenta la probabilidad para que permanezca en un estado (0) o en otro (1) según sea el caso. El cálculo de la probabilidad implica la suma de la probabilidad anterior con la diferencia entre la evaluación en cero y la evaluación en uno y por medio de una constante α podemos acelerar su cálculo. Sólo se altera la probabilidad del bit elegido, los demás bits permanecerán sin cambio hasta que sean elegidos nuevamente.

Al terminar la ejecución del cálculo de probabilidades, podemos cambiar a un nuevo individuo. Para ello se utiliza el vector de probabilidades que es comparado con un valor aleatorio. Es de esperar que las posiciones que tienen probabilidades con valores cercanos a cero permanezcan así, al igual que los que tengan valores cercanos a uno. Al final existe un criterio para finalizar el algoritmo de mutación dirigida, que generalmente se determina por un número fijo de evaluaciones.

8.3 La reutilización de soluciones

La exaptación consiste en la reutilización de conocimiento en algoritmos evolutivos; por lo cual, podemos emplear el algoritmo de mutación dirigida para reutilizar soluciones. En caso de que se use el algoritmo de mutación dirigida y tenemos una solución con la certeza de que la podemos volver a utilizar en un problema, ya sea por similitud o por conocimiento previo, entonces se puede iniciar con esta solución sustituyéndola en el vector binario V. El vector de probabilidades también debe cambiar, sin embargo es necesario establecer en donde se tiene la certeza de que las posiciones permanecen sin cambio, es decir, con una probabilidad inicial de 0.5. Ejemplo: Si tenemos una solución codificada como: $S = (0110)$ entonces $V = S$ y $V_P = (0.2, 0.8, 0.5, 0.5)$ indicando que se tiene la certeza de que las posiciones uno y dos del vector S son un conocimiento válido y el resto de las posiciones pueden cambiar de valor. Se considera una probababilidad de 0.2 cuando el valor de la posición del vector S es cero y de 0.8 cuando sucede lo contrario. Se puede notar que el efecto es muy similar a una variación elitista con mutación parcial (capítulo 7). La reutilización provoca que el algoritmo de mutación dirigida se acerque o explore subespacios que tengan altas probabilidades de contener una buena solución. La finalidad es encontrar soluciones con alta aptitud en un tiempo inferior a que si se empieza sin ninguna información.

Para el algoritmo de PBIL, no se ha encontrado ninguna referencia en donde se explique alguna forma para sembrar soluciones en este algoritmo; sin embargo, se puede plantear una forma de hacerlo. El vector de probabilidades empieza con un valor de 0.5; si tenemos una solución entonces podemos modificar cada probabilidad por un valor cercano al estado de la solución. Si la solución codificada tiene un valor de uno, entonces podemos colocar un valor de 0.8 u otro valor cercano a uno; si es cero entonces se coloca el valor de 0.2. Tampoco se han encontrado referencias sobre la forma de reutilizar soluciones en el algoritmo CGA. Se puede suponer, al igual que en el algoritmo PBIL, que se pueden reutilizar soluciones modificando el vector de probabilidades acorde con la solución. Para el algoritmo del BOA las cosas son diferentes ya que Pelikan ha propuesto el uso de soluciones "prometedoras" para formar la red bayesiana. Si de antemano se conocen algunas soluciones, es posible su uso para formar la red bayesiana inicial. Al incluir soluciones en la población inicial se evitaría iniciar con individuos aleatorios.

En la siguiente sección se realizarán algunos experimentos con la reutilización de soluciones en el algoritmo de mutación dirigida y lo comparamos con los algoritmos de PBIL y CGA.

8.4 Experimentación con la reutilización de soluciones

Enseguida se mostrará el desempeño de la mutación dirigida, el algoritmo de PBIL y el CGA en la optimización de una función cambiante. La función es la que se usó en la sección 6.3.2. Todos los algoritmos reutilizarán la solución $S = [0.0118, 0.5]$ que

codificado corresponde al vector binario $I = [0111110100000011]$. La reutilización de soluciones en la mutación dirigida implica una modificación inicial del vector V y del vector de probabilidades V_P a los valores que indique la solución I. Se usan valores de 0.7 y 0.3 para las probabilidad alta y baja respectivamente. El algoritmo de la figura 8.5 muestra cómo implantar la reutilización del vector solución en la mutación dirigida. Los parámetros de la mutación dirigida son: 25 ciclos externos con 20 ciclos internos para el cálculo de probabilidades. El factor de velocidad α es igual a 1.

Inicio del algoritmo de mutación dirigida.
1) Recuperar la solución I que se reutilizará.
1)$V(i) = I(i)$, $V_P(i) = 0.5$ donde $i = 1 \ldots 16$
2) Se modifican las posiciones del vector de probabilidades en donde la solución es más factible:
si $V(i) = 1$ entonces $V_P(i) =$ Probabilidad alta
sino $V_P(i) =$ Probabilidad baja
3) Aplicación del algoritmo de mutación dirigida.
V es un vector de una posible solución codificada.
V_P es un vector de probabilidades.
I es el vector solución que se reutilizará.

Figura 8.5: Algoritmo de mutación dirigida con reutilización de la solución I.

Para los algoritmos PBIL y CGA se realiza el mismo procedimiento usado en la mutación dirigida. Como no se usa un vector V sólo se altera el vector de probabilidades V_P a los valores que indique la solución I. Se usan valores de 0.7 y 0.3 para las probabilidad alta y baja respectivamente. El algoritmo de la figura 8.6 muestra cómo implantar la reutilización del vector solución para ambos algoritmos. En los parámetros del algoritmo CGA se representan a una población n de 100 individuos durante 500 ciclos. Para los parámetros del algoritmo PBIL se generan 40 individuos y se utilizan 20 individuos en el paso de actualización del vector de probabilidades. Se aplican 25 ciclos y se usa una constante de aprendizaje de $l_r = 0.025$.

El desempeño del algoritmo genético junto con los algoritmos de mutación dirigida, PBIL y CGA se muestra en la gráfica 8.7. Se puede notar que los tres algoritmos superan ampliamente al AGS, sin embargo, entre sí no hay una ventaja significativa de alguno de ellos.

8.5 Conclusiones

En este capítulo se propuso la manera de reutilizar soluciones para un grupo de algoritmos que se basan en la información que se extrae de una población de individuos ya que es una manera diferente de resolver problemas en donde antes se usaban algoritmos genéticos. Lo primero que se destaca en su uso es la facilidad de implantación, y segundo es la representación ya que no requieren de mucha memoria (no existe explícitamente

Inicio del algoritmo de reutilización con PBIL y CGA.
1) Se modifican las posiciones del vector de probabilidades en donde la solución es más factible:
si $V(i) = 1$ entonces $V_P(i)$ = probabilidad alta
sino $V_P(i)$ = probabilidad baja
3) Aplicación del algoritmo de PBIL o el algoritmo CGA.
V es un vector de una posible solución codificada.
V_P es un vector de probabilidades.
I es el vector solución.

Figura 8.6: Algoritmo de PBIL y CGA con reutilización de la solución I.

una población de individuos) ni de métodos de cruce y mutación. Se propuso el algoritmo de mutación dirigida cuyo funcionamiento es similar al los algoritmos de PBIL, CGA y BOA aunque bajo principios diferentes.

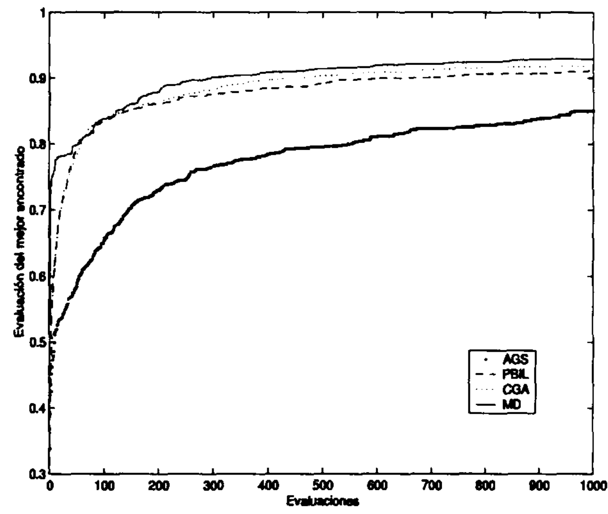


Figura 8.7: Desempeño del algoritmo genético simple comparado con los algoritmos de mutación dirigida, PBIL y CGA.

Capítulo 9

Sistemas con capacidades exaptivas

9.1 Introducción

El objetivo de este capítulo es demostrar la forma de crear sistemas con las mismas características exaptivas que se enunciaron en el capítulo 6. La exaptación se caracteriza por reutilizar y retener estructuras y como se mencionó, se necesitan procedimientos de reconocimiento, de modificación o búsqueda y de almacenamiento. El reconocimiento se aplica cuando se identifican una o más estructuras con una determinada utilidad en un medio ambiente cambiante. La modificación o búsqueda consiste en la adaptación de las estructuras para mantener y aumentar la utilidad en el medio ambiente. El almacenamiento o retención de las estructuras útiles es parte del mismo proceso evolutivo ya que las características que le permiten a un individuo ser útil, se retienen y luego se propagan entre sus descendientes. Se desea que los procesos se apliquen usando los procesos evolutivos, sin embargo, cada uno de estos procesos se pueden representar usando herramientas bien conocidas en la inteligencia computacional. Se demostrará la forma de crear sistemas con capacidades exaptivas usando herramientas evolutivas y otras de la inteligencia computacional.

Como se mencionó en capítulos anteriores, para la optimización de problemas dinámicos se han sugerido muchas propuestas (Branke, 1999a, 2001a), sin embargo solo se mencionarán los que tienen una relación con las ideas expuestas en el capítulo.

El CIGAR (Case Inicialized Genetic Algorithm) propuesto por Sushil Louis consiste en un sistema de razonamiento basado en casos con un algoritmo genético. Cada caso consiste en la mejor solución junto con el número de generación en que se almacenó, el valor de evaluación y la descripción del problema para detectar similitudes. El sistema de razonamiento por casos puede colocar casos de ejemplos para tener alguna referencia. El sistema inicia en forma aleatoria en donde se inyectan soluciones pertenecientes al sistema de razonamiento cuyo caso sea similar al nuevo problema. En cada determinado número de generaciones se inyectan soluciones del sistema de razonamiento a la población del algoritmo genético cuyo caso sea similar a uno ya resuelto; se busca al mejor individuos de la población y se almacena en la memoria del sistema de razonamiento, sustituyendo a la solución más similar pero con menor valor de evaluación. El proceso se repite en determinado número de generaciones; por

ejemplo, se pueden almacenar e inyectar soluciones en las generaciones 1, 25, 50 y 100. En CIGAR existen cuatro formas de inyectar soluciones, la primera forma consiste en inyectar soluciones que sean similares a la mejor solución encontrada; la segunda forma consiste en inyectar soluciones que sean similares a la peor solución encontrada. En la tercera forma se inyectan soluciones del sistema de razonamiento, en donde se escogen de acuerdo a un factor de probabilidad que es inversamente proporcional a la distancia de Hamming entre los casos del sistema de razonamiento y la mejor solución encontrada en la población. En la cuarta forma se inyectan soluciones del sistema de razonamiento, en donde se escogen de acuerdo a un factor de probabilidad que es directamente proporcional a la distancia de Hamming entre los casos del sistema de razonamiento y la solución de la población. En todos lo casos las soluciones inyectadas sustituyen a los peores individuos de la población. Últimamente el autor ha utilizado su sistema CIGAR sin el mecanismo para detectar similitudes por el costo o dificultad que implica determinar una métrica para un problema determinado. En su lugar ha utilizado el valor de evaluación (Louis, 1993a, 1993; Louis y Xu, 1996; Louis y Li, 1997; Louis y Johnson, 1999; Liu, 1996).

Otros autores que ha sugerido un sistema similar son Connie Ramsey y John J. Grefenstette en el cual forma un sistema basado en reglas para aprender varias secuencias. La idea es activar las reglas adecuadas según lo amerite la situación (Ramsey y Grefenstette, 1993). En el diseño evolutivo ha surgido un algoritmo que reutiliza soluciones llamado *algoritmo genético con el paradigma de "memoria"* en donde se almacenan en una memoria los genes que superan un umbral en el valor de evaluación. El umbral se establece dependiendo de la aplicación (Gero y Kazakov, 1998). El algoritmo se ha usado en el hardware evolutivo (Zebulum, Pacheco, y Vellasco, 2002).

El sistema que más se aproxima a lo propuesto en esta tesis es el algoritmo de J. Branke que consiste en un AGS con dos poblaciones (Branke, 1999b). La primera población es una memoria que almacena siempre las mejores soluciones encontradas y la segunda población busca nuevas soluciones iniciando en forma aleatoria al detectarse un cambio en la función objetivo. Branke ha propuesto otras arquitecturas por ejemplo las basadas en multipoblaciones (Branke, 2001b).

9.2 Características y Componentes de un sistema seudoexaptivo

La estructura general de un sistema seudoexaptivo así como sus elementos, se muestran en la figura 9.1. El reconocimiento identifica a un problema como conocido o no. Si es conocido, podemos traer de una memoria la solución que se usó, con la finalidad de modificarla y adaptarla al nuevo problema. Si el problema no es conocido entonces es necesario buscar una solución sin ninguna información adicional. Cuando se tiene una solución, la almacenamos junto con el problema (o una representación del mismo) para futuras referencias. El proceso se describió en el capítulo 6 y en este capítulo se mostrará la forma de implantar un sistema con características exaptivas usando métodos conocidos de reconocimiento, modificación, búsqueda y almacenamiento. En

el capítulo 3 y 4 se mencionaron varios mecanismos de reconocimientos que van desde las redes neuronales, hasta los sistemas de razonamiento basados en casos. También existen diversos mecanismos de búsqueda o modificación basados en procesos evolutivos (capítulo 2). Es posible crear sistemas con características exaptivas colocando por separado mecanismos de reconocimiento, almacenamiento y mecanismos de búsqueda o modificación. Los procedimientos se implantan en forma explícita, por lo cual tenemos sistemas seudo-exaptivos o con características exaptivas.

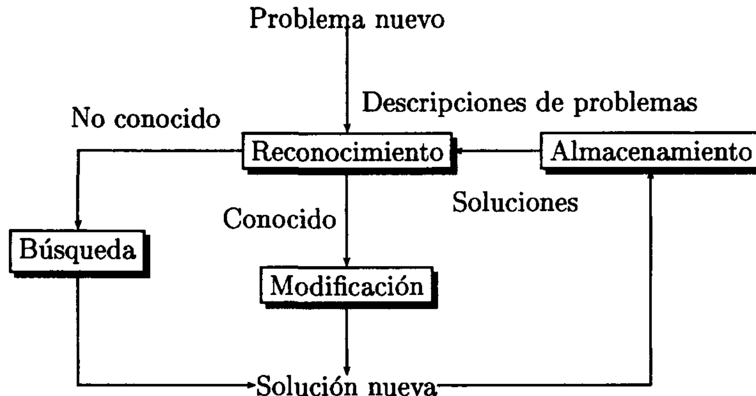


Figura 9.1: Estructura general de un sistema con capacidades exaptivas.

Es importante mencionar que la exaptación tiene una relación muy estrecha con el aprendizaje por analogía, y de hecho el sistema seudoexaptivo puede bien llamarse sistema de aprendizaje por analogía, con búsquedas y modificaciones basadas en la computación evolutiva.

9.2.1 Mecanismos de reconocimiento

El principal problema que tenemos es cómo reconocer la diferencia entre un problema y otro. La exaptación expresa una forma de reconocer la utilidad de una estructura. En un momento dado de un ser vivo, algunas de sus estructuras son útiles mientras que otras no tienen ninguna utilidad o son redundantes. Cuando las condiciones del ambiente cambian, las estructuras que eran útiles dejarán de serlo y las estructuras inútiles tendrán alguna utilidad, por lo cual, el proceso evolutivo las adaptará para el nuevo ambiente. Existe una analogía entre las estructuras y las soluciones; así que se puede usar un mecanismo de reconocimiento, en donde, se pueden tener en una memoria varias soluciones de problemas diferentes. Al presentarse un problema nuevo, cada solución de la memoria se prueba en el problema y se obtiene un valor de evaluación. La solución de la memoria con el valor de evaluación más alto será escogida para modificarla y adaptarla al nuevo problema (figura 9.2).

1) Evaluación de cada elemento de la memoria.
$F_E(i) \leftarrow$Evaluación$(M(i))$ donde $i = 1 \ldots N_{TE}$
2) Retención del elemento mejor evaluado.
$m \leftarrow 0$; $p \leftarrow 1$
Si $F_E(i) > m$ entonces $m \leftarrow F_E(i)$ y $p = i$
donde $i = 1 \ldots N_{TE}$
3) Si $m > g$ entonces es reconocido.
$r \leftarrow 1$
4) En caso contrario no es reconocido.
$r \leftarrow 0$, ir a (6)
5) La solución con la evaluación más alta está en $M(p)$.
6) El reconocimiento se indica en r.
Parámetros:
N_{TE} es el número total de elementos de la memoria
M es la memoria.
F_E es el valor de evaluación de cada elemento de la memoria.
g es un margen de evaluación preestablecido.
m es el valor de evaluación máximo.
p es la posición del individuo con mejor evaluación.

Figura 9.2: Algoritmo de reconocimiento por evaluación.

Otro método consiste en crear descripciones para cada problema, entonces se tiene una memoria de descripciones de problemas con sus respectivas soluciones. Al presentarse un problema nuevo, se obtiene una descripción del mismo, luego la descripción es comparada con las descripciones de la memoria. Las diferencias obtenidas entre la descripción del problema nuevo y las de la memoria, son usadas como una medida para determinar una similitud. La descripción de la memoria que tenga una diferencia mínima, será la más similar a la descripción del problema nuevo; por lo tanto, la solución que le corresponde será modificada para adaptarla al nuevo problema (figura 9.3). Para estimar una similitud cuantitativa que existe entre distintos problemas, generalmente suelen usarse la distancia de Hamming cuando los problemas se representan en forma binaria y la distancia euclidiana cuando se usan números reales.

9.2.2 Mecanismo para la modificación

La modificación de una solución debe implicar un menor esfuerzo que buscarla sin ninguna información porque la solución ayuda a restringir el espacio de búsqueda. Si existe una similitud entre dos problemas, entonces podemos suponer que existe una similitud entre sus soluciones; no es necesario realizar una búsqueda global, sólo es necesario buscar en las vecindades o localidades de la solución. Es por eso que el procedimiento de la modificación está relacionado con el procedimiento de la búsqueda

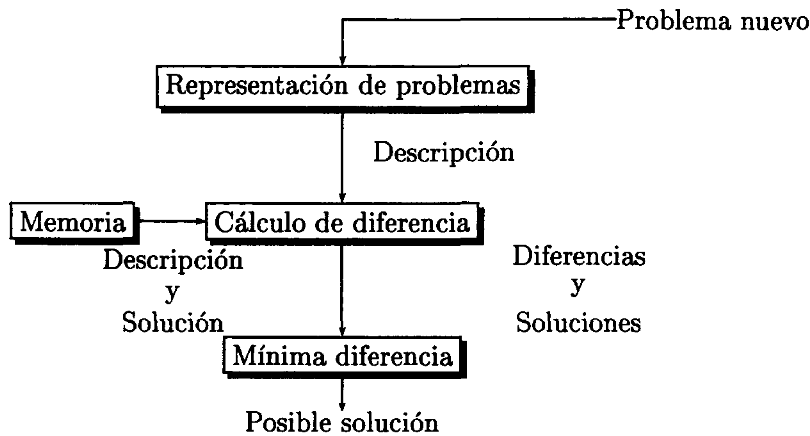


Figura 9.3: Un mecanismo de reconocimiento general.

local. Existen infinidad de métodos para aplicar búsqueda local. Sólo se mencionarán algunos de ellos.

La búsqueda local del alpinista es la referencia inmediata a un método sencillo y de fácil implantación. Consiste en realizar una modificación aleatoria a la solución y después evaluarla. Si se obtiene una alta calificación la solución se retiene, en caso contrario se busca otra. El proceso se repite hasta cumplir con algún criterio para terminarlo. Un método de búsqueda local puede ser implantado usando una técnica de sembrado de soluciones en un algoritmo evolutivo, este método es muy empleado por Sushil en sus trabajos (Louis y Johnson, 1997). Otro método de búsqueda local para permutaciones es el algoritmo de Lin-Kernighan que es considerado el más adecuado siempre y cuando se use como buscador local. Para optimización de funciones continuas tenemos algoritmos de divide y conquistarás, el método de Newton y el método del gradiente, con todas sus variantes. No suelen ser usados como buscadores globales porque generalmente quedan atrapados en mínimos (o máximos) locales (Michalewicz y Fogel, 1999).

9.2.3 Almacenamiento

El almacenamiento se liga con los procedimientos de reconocimiento. Usualmente las soluciones son guardadas directamente en una memoria junto con las descripciones de los problemas que les corresponden. Para evitar una acumulación excesiva de información se requieren mecanismos de mantenimiento que eliminen la información repetida o con poco uso.

Un método que se propone para el mantenimiento de unidades de memoria consiste en asignar un peso de uso para cada unidad. El peso tiene valores entre cero y uno. El valor de uno implica una unidad de mucho uso. Un valor de cero es todo lo contrario. Si la información de una unidad de memoria se usa, entonces el peso es actualizado a uno,

indicando que es de uso frecuente. Cuando no es usado, el peso disminuye a un valor constante (llamado factor de olvido). Si el peso de uso es muy pequeño quiere decir que no ha sido usado y por lo tanto corre el riesgo de ser reemplazado por información nueva. Un método más sencillo consiste en localizar en la memoria al elemento que fue reconocido y sustituirlo por la nueva información.

Otra forma de almacenamiento consiste en buscar y sustituir una solución almacenada que sea similar a la nueva solución. El método recuerda a los procedimientos de *crowding* usado en los algoritmos genéticos para mantener diversidad (Goldberg, 1989). Un método similar consiste en localizar en la memoria la solución más similar (con mínima distancia) a la que se quiere almacenar y sustituirla siempre y cuando la evaluación de la nueva solución es mayor que la existente en la memoria.

1) Establecer la nueva solución a almacenar I
2) Buscar una solución I_M similar a I en la memoria M
3) $f_{em} \leftarrow$ Evaluación(I_M); $fe \leftarrow$ Evaluación(I)
4) si $f_e > f_{em}$ entonces almacenar I en la memoria M sustituyendo la solución I_M
5) en caso contrario no hacer ningún cambio
6) Fin del algoritmo

Figura 9.4: Algoritmo de almacenamiento por similitud y mejor evaluación

9.2.4 Mecanismos de búsqueda

Es posible usar otros mecanismos de búsqueda global con inspiración evolutiva como las estrategias evolutivas y la programación genética porque generalmente evitan los mínimos locales. Otros mecanismos para encontrar soluciones como la búsqueda tabú, el recocido simulado, los procedimientos minimax, A*, etc., también pueden emplearse en estos sistemas. Para respetar el origen de la idea de exaptación, sólo emplearemos algoritmos genéticos.

9.3 Formas de implantación de sistemas con características exaptivas

El primer algoritmo que se puede desarrollar para implantar un sistema con características exaptivas consiste en la aplicación explícita de los mecanismos de reconocimiento, modificación, búsqueda y almacenamiento, del cual sus procedimientos se han explicado en las secciones anteriores. En la figura 9.5 se muestra el algoritmo general para tener sistemas con capacidades exaptivas.

El sistema está resolviendo problemas diversos, algunos son similares entre sí y otros no lo son. El sistema guarda cada una de las soluciones de los problemas que se

1) Reconocimiento por evaluación.
$(m, p) \leftarrow$Evaluación(M)
2)Si $m > g$ entonces $S \leftarrow$ Búsqueda Local($M(p)$); ir a (4)
3) $S \leftarrow$Búsqueda Global.
4) $M \leftarrow$Almacenamiento(S)
5) Fin del algoritmo.
Parámetros:
m es el valor de evaluación más alto de la memoria.
p es la posición de la memoria con la evaluación más alta.
g margen de reconocimiento.
M memoria de soluciones.
S es la solución.

Figura 9.5: Algoritmo de un sistema con características exaptivas.

han resuelto en el pasado. El primer procedimiento es el reconocimiento que se basa en la evaluación de las soluciones de la memoria con el problema actual para encontrar una con la evaluación más alta m, así como su posición en la memoria p. Si la solución de la memoria con más alto valor de evaluación supera un margen preestablecido g, entonces se considera como reconocido, es decir, el problema es similar a uno resuelto con anterioridad. La búsqueda local requiere de una referencia para modificarla y obtener una solución. Se usa como referencia la posición de la memoria con la más alta evaluación $M(p)$ y que su valor supera un margen preestablecido g. Si no existe reconocimiento, se realiza una búsqueda global para buscar una nueva solución. La solución modificada o encontrada S se almacena en la memoria M.

Existe otra forma de obtener un sistema exaptivo. El procedimiento consiste en una modificación del algoritmo genético simple para provocar la exaptación. Es una forma más natural, sin embargo tiene problemas porque no existe una memoria explícita; (sucede lo mismo en otros algoritmos evolutivos) sin embargo, bajo ciertas condiciones que se mencionarán enseguida, puede ser una buena alternativa para tener sistemas con características exaptivas. El algoritmo genético simple puede comportarse como un sistema exaptivo si se cumplen dos requisitos, el primero es que se tengan poblaciones grandes con alta diversidad, y el segundo es que la función objetivo no cambie de manera significativa. Para tener exaptación en un algoritmo genético es necesario iniciar con una población aleatoria sólo una vez. Cada vez que se ejecute el algoritmo se usa nuevamente la misma población sin iniciarla con individuos aleatorios. El algoritmo se describe en la figura 9.6.

Si la función realiza cambios muy bruscos entonces la población converge con rapidez a soluciones con baja evaluación. Para evitar la convergencia prematura, se inyectan soluciones aleatorias junto con variaciones y vecindades de la mejor solución que se encuentran en la población inicial. Es necesario modificar el algoritmo genético exaptivo incluyendo los mecanismos para inyectar soluciones bajo ciertas condiciones cambiantes

1) Se inicia con una población P previamente almacenada
(La primera vez P es aleatoria.)
2) $F_E \leftarrow$Evaluación(P)
3) $P \leftarrow$Selección(P, F_E)
4) $P \leftarrow$Cruce y mutación(P)
5) Si es fin del algoritmo entonces terminar, sino ir a (2)
6) Se almacena la población P.

Figura 9.6: Algoritmo genético exaptivo.

1) Se inicia con una población P previamente almacenada
(La primera vez P es aleatoria.)
2) Medida de la degradación de la población.
3) Inyección de soluciones aletorias y
variaciones del individuo con la más alta evaluación.
4) $F_E \leftarrow$Evaluación(P)
5) $P \leftarrow$Selección(P, F_E)
6) $P \leftarrow$Cruce y mutación(P)
7) Si es fin de algoritmo entonces terminar, sino ir a (2)
8) Se almacena la población P.

Figura 9.7: Algoritmo genético exaptivo con inyecciones.

en los valores de evaluación observadas en la población. El algoritmo genético exaptivo con inyecciones se muestra en la figura 9.7.

La degradación de la población existe cuando la evaluación del mejor individuo disminuye cuando la función cambia. Si la evaluación disminuye es un indicativo de que la función cambió y que es necesario inyectar en la población nuevas soluciones aleatorias o variaciones de la mejor solución encontrada. El número de individuos inyectados depende del nivel de degradación que se presente en la población. Si el nivel es muy alto es necesario inyectar individuos aleatorios, pero si no es tan bajo pueden inyectarse variaciones del mejor individuo o de la población. Para establecer el grado de degradación suele usarse un criterio empírico y depende mucho del problema que se resuelva. En la experimentación se analizarán algunas formas para detectar cambios en la función objetivo.

Lo ideal es tener algoritmos genéticos con poblaciones que tengan una alta diversidad (retención) lo cual permitiría considerar a cualquier individuo de la población cuando las condiciones cambien (reutilización), es decir, no sería necesario inyectar nuevos individuos. Si bien en este trabajo se buscó la manera de lograr retención y reutilización en algoritmos genéticos, es importante señalar que el problema de la retención está aún vigente. Una última arquitectura que se puede proponer es una

1) $P \leftarrow$ Inicio aleatorio
2) $F_E \leftarrow$ Evaluación(P)
3) Obtener al mejor individuo I y su valor de evaluación e de F_E
4) $P \leftarrow$ Selección(P, F_E)
5) $P \leftarrow$ Reproduction(P) (cruce y mutación)
6) Si no es fin de algoritmo entonces ir a (2) sino terminar

Figura 9.8: Algoritmo genético simple.

1) $P \leftarrow$ Inicio aleatorio
2) $F_E \leftarrow$ Evaluación(P)
3) Obtener al mejor individuo I y su valor de evaluación e de F_E
4) $P \leftarrow$ Selección(P, F_E)
5) $P \leftarrow$ Reproducción(P) (cruce y mutación)
6) Aplicar elitismo inyectando el individuo I en P
7) Si no es fin de algoritmo entonces ir a (2) sino terminar

Figura 9.9: Algoritmo genético simple con elitismo.

mezcla del algoritmo genético con inyecciones con una memoria. La arquitectura del tal arquitectura se muestra en la siguiente sección.

9.4 Experimentación

9.4.1 Implantación de los algoritmos

En la experimentación se implantaron tres algoritmos, el algoritmo genético simple con y sin elitismo junto con el algoritmo genético de dos poblaciones propuesto por Branke. Los algoritmos se usaron para comparar su desempeño contra dos algoritmos con características exaptivas; el primer algoritmo usa solamente técnicas de sembrado y el segundo algoritmo usa sembrado y mecanismos de memoria.

El algoritmo genético simple reutiliza la misma población que se inicializa en forma aleatoria solo al inicio de la ejecución. El algoritmo se muestra en la figura 9.8 donde P es la población, F_E retiene el valor de evaluación de cada individuo, I es una variable auxiliar para retener al mejor individuo encontrado por generación y su valor de evaluación se almacena en e.

El algoritmo genético simple con elitismo tiene las mismas características que el algoritmo anterior, sólo que se incluye un paso extra para mantener al mejor individuo encontrado por generación. El algoritmo se muestra en la figura 9.9.

El algoritmo genético simple con dos poblaciones propuesto por Branke (figura 9.10) tiene una población de memoria M y una población de búsqueda P. La mejor

1) $P \leftarrow$ Inicio aleatorio
2) $M \leftarrow$ Inicio aleatorio o memoria vacía
3) Obtener a la mejor evaluación del pasado e_a
4) $(F_E) \leftarrow$ Evaluación(P)
5) Obtener al mejor individuo I y su valor de evaluación e de F_E
6) $(F_{EM}) \leftarrow$ Evaluación(M)
7) Obtener al mejor individuo I_m y su valor de evaluación e_m de F_{EM}
8) Detectar al mejor individuo I_g entre I e I_m
9) Si $\max(e_m, e) < e_a$ entonces $P \leftarrow$Inicio aleatorio
10) Cada k generaciones el mejor individuo I_g se almacena en la memoria M
11) $(P) \leftarrow$ Selección(P, F_E)
12) $(P) \leftarrow$ Reproducción(P) (cruce y mutación)
13) Aplicar elitismo inyectando el individuo I en P
14) Si no es fin de algoritmo entonces ir a (3) sino terminar

Figura 9.10: Algoritmo genético basado con dos poblaciones propuesto por Branke.

solución se guarda en la memoria cada k generaciones, en donde se busca la mínima distancia (mayor similitud) entre la mejor solución encontrada I_g y las soluciones almacenadas en la memoria M. Para detectar un cambio en la memoria se compara el valor de evaluación de la mejor solución encontrada (el máximo entre e_m y e) contra el valor de evaluación de una generación anterior e_a (se asume que los cambios en la función objetivo se presentan después de varias generaciones, es decir, no se espera un cambio en la función sin antes evaluar a todos los individuos) El algoritmo reinicializa la población P cada vez que se detecta un cambio en la función objetivo.

El primer algoritmo propuesto es el algoritmo genético simple con sembrado. Se inspira en la exaptación porque reutiliza las soluciones actuales cuando detecta un cambio en la función de evaluación. Al detectarse el cambio se generan variaciones y vecindades de la mejor solución del momento y son inyectados en la población actual sustituyendo a un porcentaje de la población. Las soluciones vecinas pueden dar un buen punto de partida siempre y cuando la función no cambie mucho. Si la función cambia de una manera muy brusca es posible que las variaciones de la solución puedan dar alguna pista para obtener la solución de la nueva función. El algoritmo se muestra en la figura 9.11.

El segundo algoritmo es el AGS con memoria y sembrado; se inspira en la exaptación y se implanta desde el punto de vista del aprendizaje por analogía. El algoritmo tiene una memoria de soluciones útiles en el pasado, un procedimiento de almacenamiento, un mecanismo de búsqueda y un mecanismo de modificación basado en técnicas de sembrado. El mecanismo aplica básicamente dos procedimientos: Un mecanismo de reconocimiento basado en la evaluación de cada individuo de la memoria para compararlo con el valor de evaluación de la mejor solución encontrada y así detectar alguna

1) $P \leftarrow$ Inicio aleatorio
2) Obtener a la mejor evaluación del pasado e_a
3) $(F_E) \leftarrow$ Evaluación(P)
4) Obtener al mejor individuo I y su valor de evaluación e de F_E
5) Si $e < e_a$ entonces aplicar un sembrado del 50% de vecindades y variaciones de I en P
6) $(P) \leftarrow$ Selección (P, F_E)
7) $(P) \leftarrow$ Reproducción (P) (cruce y mutación)
8) Incluir elitismo inyectando I en P
9) Si no es fin de algoritmo entonces ir a (3) sino terminar

Figura 9.11: Algoritmo genético con técnicas de sembrado.

solución que puede ser útil en la nueva función. El procedimiento de almacenamiento guarda el mejor individuo encontrado. El algoritmo se muestra en la figura 9.12.

9.4.2 Función de prueba

Muchos autores han sugerido diversos problemas dinámicos para probar los algoritmos, sin embargo pueden ser o muy sencillos o muy complejos para usarse en la investigación. Branke ha sugerido un problema que consiste en la optimización de una función multidimensional con picos que cambian de posición, ancho y altura en el tiempo. Esta función se le conoce como MPB de *Moving Peak Benchmark*. La función depende de varios parámetros como la posición de los picos X, el vector de posición V, la altura de cada pico H y la anchura de cada pico W. La función tiene la forma siguiente:

$$f(x,t) = \max_{i=1,5} \frac{H(i)(t)}{1 + W(i)(t)\sum_{j=1}^{5}(x(j) - X(j)(t))^2} \tag{9.1}$$

En cierto número de generaciones, la altura H y la anchura W de cada pico es cambiado añadiendo un valor aleatorio con una distribución gaussiana. La localización de cada pico es movido por el vector V con una longitud fija s en una dirección aleatoria, por lo cual, el parámetro s permite controlar la severidad del cambio. Existe otro parámetro llamado λ que determina la tendencia del cambio que puede ser aleatoria ($\lambda = 0$) o depender de la dirección anterior ($\lambda > 0$). El cambio del vector V es como sigue:

$$V(i)(t) = \frac{s}{|r(i) + V(i)(t-1)|}((1-\lambda)r + \lambda V(i)(t-1)) \tag{9.2}$$

La variable r es un vector aleatorio normalizado con el valor de s. Los parámetros iniciales son:

1) $P \leftarrow$ inicio aleatorio.
2) $M \leftarrow$ Inicio aleatorio o memoria vacía
3) Obtener el mejor valor de evaluación anterior e_a
4) $(F_{EM}) \leftarrow$ Evaluation (M)
5) Obtener el mejor individuo I_m y su mejor valor de evaluación e_m de F_{EM}
6) Si $e_m > e_a$ entonces aplicar un sembrado al 50% vecindades y variaciones de I en P.
7) $(F_E) \leftarrow$ Evaluación(P)
8) Obtener el mejor individuo I y su valor de evaluación e de F_E
9) $(P) \leftarrow$ Selección(P, F_E)
10) $(P) \leftarrow$ Reproducción(P) (cruce y mutación)
11) Cada k generaciones guardar en la memoria M el mejor individuo I
12) Si no es fin de algoritmo entonces ir a (3) sino terminar

Figura 9.12: Algoritmo genético con memoria y sembrado.

$$P_c = \begin{bmatrix} 8 & 64 & 67 & 55 & 4 \\ 50 & 13 & 76 & 15 & 7 \\ 9 & 19 & 27 & 67 & 24 \\ 66 & 87 & 65 & 19 & 43 \\ 76 & 32 & 43 & 54 & 65 \end{bmatrix}$$

P_c representa a los picos iniciales y es una matriz para 5 variables y 5 picos por variable. Los vectores iniciales de la anchura y la altura son respectivamente $W =$[0.1 0.1 0.1 0.1 0.1] y $H =$ [50 50 50 50 50]. La matriz X corresponde a los 5 picos para las 5 variables y toma como valor inicial el contenido de la matriz P_c. El vector V toma valores aleatorios. La matriz V toma como valores iniciales el resultado de restar el valor de 0.5 a un número aleatorio con distribución gaussiana. La siguientes ecuaciones determinan el cambio de cada parámetro.

$$\sigma = N(0,1) \tag{9.3}$$
$$H(i)(t) = H(i)(t-1) + 7 \cdot \sigma \tag{9.4}$$
$$W(i)(t) = W(i)(t-1) + 0.01\,\sigma \tag{9.5}$$
$$X(t) = X(t-1) + V \tag{9.6}$$

Para la experimentación se generaron 100 parámetros con la finalidad de que cada algoritmo optimize la función con los mismos parámetros.

9.4.3 Descripción del experimento y resultados

Para la experimentación se usó en todos los algoritmos una probabilidad de cruce y de mutación de 0.8 y 0.025 respectivamente. La población es de 100 individuos

y para los algoritmos que usan una memoria se usaron espacios para 10 individuos y al inicio la memoria contiene datos aleatorios. El mecanismo de selección es de torneo de tamaño dos y con un mecanismo de cruce simple de un punto. La función utilizada requiere de 5 variables que son codificados en 8 bits cada una dando un total de 40 bits que es el tamaño del cromosoma. Por tener una función de optimización dinámica no es conveniente reportar la mejor solución encontrada sino que se reporta el comportamiento después de la ejecución que consiste en la evaluación promedio del mejor individuo encontrado en cada instante de tiempo T, es decir

$$e_p(T) = \frac{1}{T} \cdot \sum_{t=1}^{T} e(t) \tag{9.7}$$

donde $e(t)$ es la mejor evaluación en cada instante de tiempo t. El número de datos usados para obtener el promedio crece con el tiempo, así que la curva tiende a ser suave. La función de evaluación cambia cada 50 generaciones. Cada algoritmo funciona durante 5000 generaciones. Las gráficas que se obtienen son el resultado del desempeño promedio de 20 ejecuciones.

Existen dos casos de prueba, en el primero la localización de cada pico permanece sin cambio, sólo existen cambios en la altura y anchura de los picos. Para lograrlo se coloca en los parámetros del MPB los valores de $s = 0$ y $\lambda = 1$. La figura 9.13 muestra el comportamiento de cuatro algoritmos. El AGS con elitismo (AGSE) no se muestra en la figura ya que tiene un comportamiento muy similar al AGS. La figura sugiere un comportamiento muy similar entre el algoritmo de Branke (AGS2P) y el algoritmo con capacidades exaptivas basado en memoria y sembrado (AGSMS). Ambos superan al algoritmo genético simple y al algoritmo genético con sembrado (AGSS).

En el segundo caso los picos cambian y existe un movimiento más brusco en la altura y en la anchura. Esto hace a la función más difícil de optimizar. Para lograrlo se modifican los parámetros $s = 0.5$ y $\lambda = 0.9$ en MPB. La figura 9.14 muestra el comportamiento de cuatro algoritmos, sólo el AGS no es mostrado por tener un comportamiento muy similar al AGSE. El AGSSM supera ligeramente al AGSS y al AGS2P que tienen entre ellos un comportamiento muy similar.

Dado que existen infinidad de métodos de búsqueda local, búsqueda global, mecanismos de almacenamiento y de reconocimiento, las combinaciones que pueden existir para encontrar un sistema con capacidades exaptivas son muy altas. En el trabajo se propusieron dos algoritmos que hacen un uso intensivo de técnicas de sembrado, primero elitista para tener referencia de la solución anterior, segundo de variación elitista para generar posibles esquemas útiles, sobre todo si la función realiza un cambio muy brusco, y de vecindades si la función cambia ligeramente y es muy probable que una vecindad de la solución reutilizada pueda ser útil.

9.5 Conclusiones

En el capítulo se expuso la manera de obtener algoritmos con capacidades exaptiva de los cuales se propusieron dos de ello y se pusieron a prueba con otras alternativas

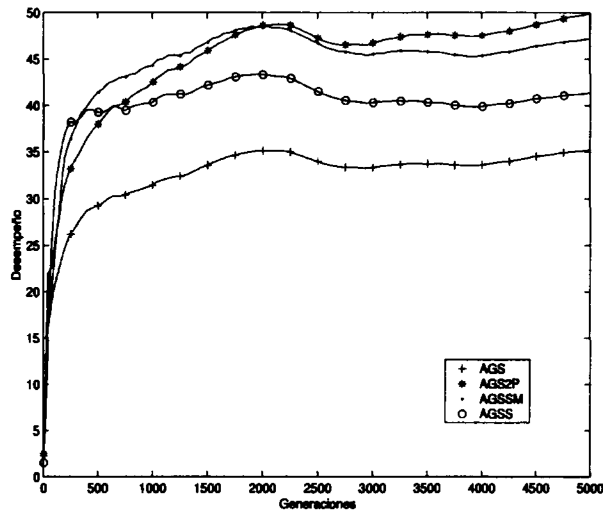


Figura 9.13: Desempeño de varios algoritmos con severidad $s = 0.0$ y factor de cambio $\lambda = 1$.

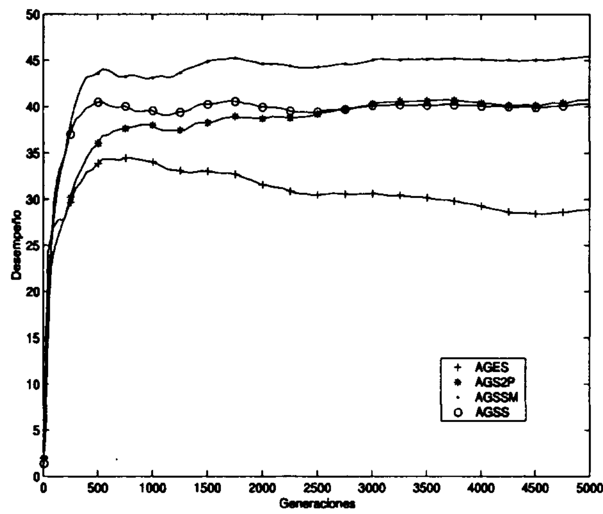


Figura 9.14: Desempeño de varios algoritmos con severidad $s = 0.5$ y factor de cambio $\lambda = 0.9$.

existentes en la literatura. Existen diversos mecanismos para reconocer, modificar, buscar y almacenar soluciones, sin embargo, se interesó en las propuestas evolutivas para encontrar mecanismos que le permitan a un algoritmo evolutivo retener y reutilizar soluciones. Los algoritmos propuestos hacen un uso intensivo de técnicas de sembrado de la mejor solución encontrada en el momento. En todos los algoritmos se usó la función representada en el MPB. sugerido por Branke. Se demostró que los algoritmos propuesto son competitivos para resolver problemas dinámicos, específicamente funciones dinámicas.

Los resultados nos indica que el sembrado de variaciones y vecindades permite reutilizar una solución cuando la función de optimización realiza un cambio ligero o brusco. Es necesario realizar más experimentos con otras funciones e inclusive con otros problemas, sin embargo, el espacio es reducido así como el tiempo, por lo cual es un tema para trabajos futuros.

Capítulo 10

Redes neuronales y los algoritmos genéticos con exaptación

10.1 El sistema seudoexaptivo neuronal

En el capítulo 3 se revisaron las formas de entrenamiento en las redes neuronales evolutivas haciendo énfasis al aprendizaje de políticas o relaciones funcionales. Se presentaron tres formas de realizar un entrenamiento evolutivo de las redes neuronales; una consistía en modificar los pesos de conexión que existen entre las neuronas, otra en modificar la estructura de la red neuronal y la última en modificar la regla de aprendizaje. Como se mencionó, una de las aplicaciones que pueden tener las redes neuronales es en la aproximación de funciones usando muestras de las entradas aplicadas a la función o proceso con sus respectivas salidas. Esta propiedad de las redes neuronales las hace muy valiosas para desarrollar sistemas de reconocimiento de señales, control, pronóstico e identificación de sistemas sobre todo cuando no es posible obtener una ecuación que indique el funcionamiento de algún proceso. Cuando se entrena una red neuronal se busca una relación funcional entre un conjunto de datos. Los datos pueden ser las entradas aplicadas a un proceso con las salidas obtenidas del mismo.

En algunas redes neuronales como la de retropropagación, se busca el aprendizaje de una relación a la vez, cuando por alguna circunstancia el proceso cambia, es necesario volver a entrenar la red neuronal porque la relación funcional que encontró ya no es útil. Generalmente cuando se realiza el entrenamiento, se inicia en forma aleatoria ya sea una red neuronal de retropropagación o una red neuronal evolutiva. Lo ideal es no empezar en forma aleatoria, sino aprovechar el conocimiento previo para reutilizarlo en circustancias en que es necesario aprender una nueva relación funcional y existe alguna semejanza entre lo hecho anteriormente y lo que se tiene ahora. En otras palabras, se desea desarrollar un sistema neuronal que pueda aprender varias funciones y que además, cuando tenga enfrente un proceso previamente aprendido, mejore su aprendizaje del mismo.

En este capítulo se expone un sistema exaptivo neuronal que tiene las propiedades mencionadas arriba, es decir, la reutilización de conocimiento previamente aprendido

y el incremento de su capacidad de aprendizaje al aprender nuevas funciones y mejorar las ya aprendidas. Para simplificar el sistema, se usará una red neuronal muy sencilla, que aproxima funciones usando tablas con pares de entradas y salidas. La descripción de la red neuronal se realiza en la segunda sección de este capítulo. Las características del sistema seudoexaptivo con redes neuronales se describe en la tercera sección. Los experimentos que se presentan en la sección 4 demuestran la eficiencia que tiene el sistema seudoexaptivo en el aprendizaje de varias funciones, en la reutilización del conocimiento previo y en el mejoramiento del conocimiento aprendido. Las conclusiones será un tema de la última sección.

10.2 La red neuronal de funcionamiento por inhibición

La red neuronal de funcionamiento por inhibición consiste en una estructura que simula el mecanismo presentado en algunos circuitos neuronales en donde al ser excitado un conjunto de neuronas, las neuronas que son opuestas a las mismas son inhibidas para permitir la acción total de la neurona excitada (Nabet y Pinter, 1991). Este tipo de circuitos son muy comunes en los mecanismos empleados para mantenerse en pie, al doblar un brazo y en el movimiento de los ojos. La red neuronal es muy sencilla y de fácil implantación, por lo cual se usará para demostrar que es posible reutilizar sus parámetros para aprender funciones cambiantes.

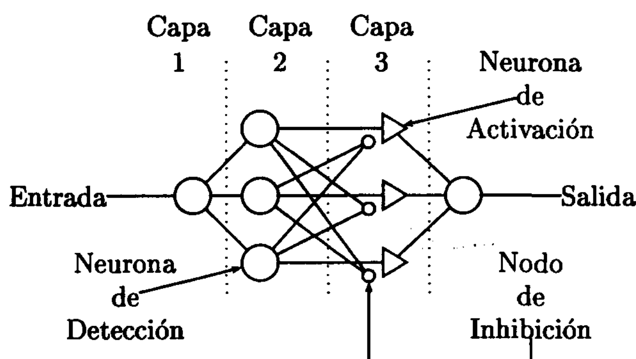


Figura 10.1: La red neuronal de funcionamiento por inhibición.

La red neuronal tiene cuatro capas 10.1. La primera capa distribuye la señal que proviene de la neurona de entrada. La segunda capa está compuesta por neuronas que detectan determinadas señales de entrada. Las neuronas usan una función de activación triangular simétrica; por lo cual, cuando la entrada es igual al centro de masa (cm), la función de activación tendrá el valor de uno a la salida (figura 10.2).

$$f_t(x, cm, b) = \max\left(\min\left(\left(\frac{x-b}{cm-b}\right), \left(\frac{b-x}{b-cm}\right)\right), 0\right) \quad (10.1)$$

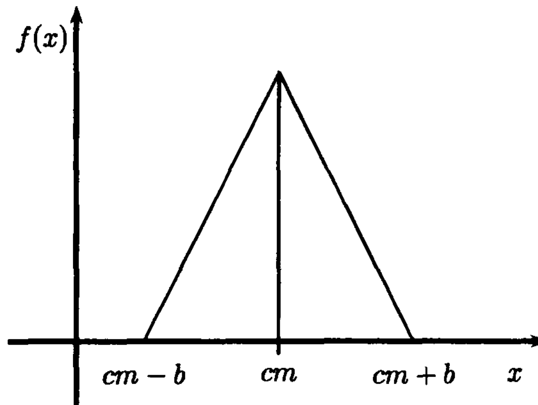


Figura 10.2: La función triangular tiene dos parámetros para controlar su comportamiento.

La tercera capa consiste en las neuronas de activación con entradas de inhibición y excitación. El funcionamiento de estas neuronas se relaciona con el concepto de que "el ganador se lleva todo"; por lo cual el funcionamiento se refleja aplicando inhibiciones entre las neuronas rivales. Cada neurona de detección se relaciona con una neurona de activación, que es excitada si recibe una señal de entrada cerca del centro de masa de la función de activación. Las neuronas de detección inhiben (por medio de un nodo de inhibición) a las demás neuronas de activación con la misma intensidad con la que excitan a sus correspondientes neuronas de activación (figura 10.3).

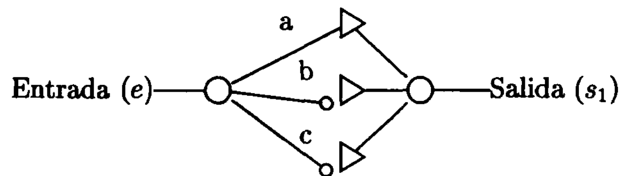


Figura 10.3: Las neuronas de activación excitan a su neurona de activación correspondiente e inhiben a las demás neuronas de activación.

Las neuronas de sensibilidad excitan a su neurona de activación correspondiente e inhiben a las demás neuronas de activación. Finalmente, en la última capa se realiza la integración por medio de la suma de todas las señales de las neuronas de activación. La señal integrada corresponde a la salida de la red neuronal. Si consideramos una red neuronal de tres neuronas de detección con su correspondiente neurona de activación, tendremos una red como se muestra en la figura 10.4.

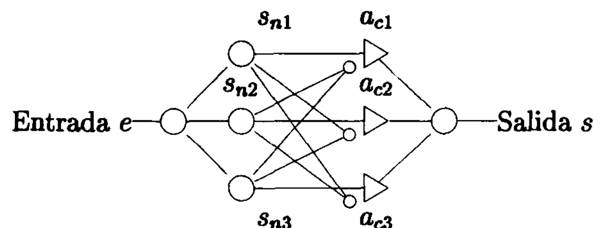


Figura 10.4: Red neuronal de funcionamiento por inhibición de una entrada y una salida.

En el siguiente se usará una red neuronal de una entrada y una salida (figura 10.4). Cada neurona de detección (por ejemplo s_{n1}) tiene una función triangular simétrica con dos parámetros (para la neurona s_{n1} son cm_1 y b_1). Las neuronas de activación (la neurona en forma triangular) funcionan como amplificadoras de las señales de las neuronas de detección. Por cada neurona de detección se tiene una neurona de activación. La neurona de activación tiene un factor de amplificación (a_{c1} por ejemplo); por lo tanto, tenemos otro parámetro a considerar. Dada una entrada e, la salida de cada neurona de detección es:

$$s_{n1}(e) = f_t(e, cm_1, b_1) \tag{10.2}$$

$$s_{n2}(e) = f_t(e, cm_2, b_2) \tag{10.3}$$

$$s_{n3}(e) = f_t(e, cm_3, b_3) \tag{10.4}$$

Las salidas de las neurona de activación son las siguientes:

$$s_1(e) = a_{c1}\,(s_{n1}(e)\,(1 - s_{n2}(e))\,(1 - s_{n3}(e))) \tag{10.5}$$

$$s_2(e) = a_{c2}\,(s_{n2}(e)\,(1 - s_{n1}(e))\,(1 - s_{n3}(e))) \tag{10.6}$$

$$s_3(e) = a_{c3}\,(s_{n3}(e)\,(1 - s_{n1}(e))(1 - s_{n2}(e))) \tag{10.7}$$

Finalmente la salida de la red neuronal corresponde a la suma de las salidas de las neuronas de activación:

$$S = s_1(e) + s_2(e) + s_3(e) \tag{10.8}$$

En resumen, tenemos tres parámetros por cada par compuesto de una neurona de activación y de una neurona de detección. Si tenemos tres pares de neuronas, entonces es necesario realizar el ajuste a 9 parámetros (tres parámetros por tres neuronas) para aproximar alguna función.

10.3 Características del sistema exaptivo neuronal

El sistema exaptivo neuronal tiene varias estructuras y procedimientos para su funcionamiento. El sistema debe ser capaz de aprender varias funciones. Existe una red neuronal por cada función aprendida, por lo cual el sistema necesita una memoria para albergar los parámetros de cada red neuronal. Para tener una memoria con una arquitectura uniforme, la estructura de cada red neuronal permanecerá fija en el sistema. En los experimentos se usará una red neuronal de 5 neuronas; como existen tres parámetros por neurona, tenemos 15 parámetros en total. Se usará una codificación de 8 bits por parámetro, por lo cual se tiene 8 por 15 = 120 bits. La memoria almacenará todos los parámetros de cada red neuronal y podrá almacenar máximo los parámetros de 10 redes. La memoria es una matriz de 10 filas por 120 columnas, ya que se tienen 15 parámetros codificados a 8 bits para cada una de las 10 redes neuronales.

Para entender a todo el sistema exaptivo, es necesario explicar cada procedimiento en forma individual. El primer procedimiento que se usará es el reconocimiento (figura 10.5) que busca una red neuronal contenida en la memoria M que mejor se aproxime a una función actual. La red neuronal se representa por el vector de parámetros de la función K. La función $Maxp$ localiza al más alto valor de evaluación del vector F_E para localizar la posición en la memoria de la red neuronal más apta (mejor aproxima a la función K). Si el valor v_x supera un margen preestablecido (en este caso el margen es de 0.85) entonces se reconoce la función y es posible usar la red neuronal colocada en la posición de la memoria p_x. La evaluación consiste en comparar las salidas de la red

$(P_R, r) \leftarrow$Reconocer(M, K)
1) $g \leftarrow 0.85$.
2) $F_E(i) \leftarrow$Evaluación$(M(i,j), K)$ para $j = 1 \ldots N_{TB}$, $i = 1 \ldots N_{TD}$
3) $(v_x, p_x) \leftarrow Maxp(F_E)$
4) si $v_x > g$ entonces $r = 1$
5) sino $r \leftarrow 0$
6) $P_R(j) \leftarrow M(p_x, j)$ para $j = 1 \ldots N_{TB}$
7) fin del algoritmo.

Figura 10.5: Procedimiento de reconocimiento.

neuronal con las salidas de la función con parámetros K (figura 10.6). Todos los datos del vector X se usan como entradas para la red neuronal y la función con parámetros K. La búsqueda global (figura 10.7) se ejecuta cuando la función K no puede ser

$F_E \leftarrow$ Evaluación(P_R, K)
1) $e \leftarrow 0$
Las entradas de prueba se definen en el vector X.
2) Respuesta de la red neuronal. $y_1 \leftarrow R_N(P_R, X(d))$
3) Respuesta de la función K. $y_2 \leftarrow$ función$(K, X(d))$
4) $e \leftarrow (y_1 - y_2)^2 + e$ para toda $d = 1 \ldots N_{TD}$
5) $F_E \leftarrow 1 - \sqrt{e}$
6) fin del algoritmo.
Parámetros:
N_{TD} es el total de datos de prueba contenidos en el vector X.
e corresponde al error cuadrático medio.

Figura 10.6: Procedimiento de evaluación.

aproximada por ninguna red neuronal de la memoria y consiste en buscar los parámetros adecuados de una red neuronal usando un algoritmo genético de $n = 100$ individuos, probabilidad de cruce y mutación de $p_c = 0.8$ y $p_m = 0.001$ respectivamente, selección de torneo de tamaño dos; después de 50 generaciones el mejor individuo encontrado I se almacena en la memoria.

$S \leftarrow$Búsqueda Global
1) Inicio de una población aleatoria P.
2) Algoritmo genético simple usando la función K.
3) El mejor individuo encontrado se almacena en el vector S.
4) Fin de algoritmo.

Figura 10.7: Algoritmo de búsqueda global.

La búsqueda local (figura 10.8) se aplica si existe un grado de reconocimiento de la función K, por lo cual, se aplica un sembrado poblacional usando los parámetros de la red neuronal que se coloca en la posición de memoria que mejor se aproxima a la función actual K. Las características son las mismas que las usadas en la búsqueda global (sólo que se usan menos ciclos, 10 en total, asumiendo que la reutilización implica menos esfuerzo en la búsqueda.)

El almacenamiento (figura 10.9) es muy sencillo y consiste en localizar a la posición del vector P_U con un peso de uso mínimo y sustituirlo por la nueva información (vector S). La función $Minp$ localiza de un vector el valor mínimo y su posición. Cada vez que se inserta nuevo conocimiento, el peso toma el valor de uno. Todos los pesos decrecen de forma constante a lo largo del algoritmo.

El algoritmo completo del sistema exaptivo neuronal se muestra en la figura 10.10; toma como argumentos la memoria que almacena los parámetros de las redes neuronales y los pesos de uso de cada red neuronal; otro argumento son los parámetros de la función

$S \leftarrow$ Búsqueda Local(P_r)
1) Inicio de una población aleatoria P.
2) Siembra de la solución P_r en la población P.
3) Algoritmo genético simple usando la función K.
4) El mejor individuo encontrado se almacena en el vector S.
5) Fin del algoritmo.

Figura 10.8: Algoritmo de búsqueda local.

$(P_U, M) \leftarrow$ Almacenamiento(S, P_U, M)
1) $(p_n, m_n) \leftarrow$ Minp(P_U)
2) $M(p_n, j) \leftarrow S(j)$ para toda $j = 1 \ldots N_{TB}$
3) $P_U(pn) \leftarrow 1$
4) fin de algoritmo
Parámetro: N_{TB} es el número total de bits del cromosoma S.

Figura 10.9: Procedimiento de almacenamiento.

de prueba (vector K). Se espera obtener a la salida nuevamente la memoria y los pesos de uso. El primer paso consiste en el mecanismo de reconocimiento de la función, es decir, se verifica si hay una red neuronal en la memoria M que aproxime dentro de un margen predefinido la nueva función con parámetros K. Si existe una red neuronal entonces se reutilizan sus parámetros para mejorar el aprendizaje de la función por medio de una búsqueda local, sino se inicia en forma aleatoria y se busca globalmente un conjunto de parámetros que aproxime la nueva función. Los parámetros modificados en la búsqueda local o global se almacenan en la memoria sustituyendo a parámetros de redes neuronales que han tenido poco uso.

10.4 Experimentaciones sobre el comportamiento del sistema seudoexaptivo neuronal

Los siguientes experimentos demuestran el desempeño de un sistema con capacidades exaptivas con redes neuronales y demuestran el funcionamiento de estos sistemas en el aprendizaje de varias funciones. El sistema seudoexaptivo tiene un mecanismo de memoria y un mecanismo de búsqueda local que le permite reutilizar soluciones que son reconocidas. Para el reconocimiento se usa una memoria que contiene soluciones de un problema anterior. Las soluciones son los parámetros de la red neuronal que permite aproximar una función particular. La búsqueda global se usa cuando no existe un reconocimiento, es decir, no hay en la memoria un conjunto de parámetros que permita reconocer la función nueva. El sistema tiene mecanismos de almacenamiento

1) $(M, P_U) \leftarrow$ SEN(M, P_U, K)
2) $(P_R, r) \leftarrow$ Reconocer(M, K)
3) $(P_U) \leftarrow$Actualiza(P_U, f_o)
4) si $r = 1$ entonces $S \leftarrow$Búsqueda local(P_R)
5) sino $S \leftarrow$ Búsqueda Global.
6) $(M, P_U) \leftarrow$ Almacenar(M, P_U, S)
7) fin del algoritmo.
Parámetros:
M es la memoria de parámetros.
P_U es el vector de pesos de uso.
K Parámetros de la función a aprender.
P_R Parámetros de la mejor red neuronal que aproxima a la función K.
r bandera de reconocimiento.
S Mejor vector solución encontrado.

Figura 10.10: Algoritmo del sistema seudoexaptivo neuronal.

que permiten almacenar los parámetros encontrados ya sea por una modificación de parámetros extraídos de la memoria o encontrados por el algoritmo de búsqueda global. La primera característica de un sistema exaptivo es su capacidad de aprender varias funciones. Se generó una función que depende de tres parámetros que determinan una forma particular. La función se muestra enseguida:

$$f(x, K) = K(1) + K(2)\text{sen}(10 - x) + K(3)\,\exp(4 - x^2) \tag{10.9}$$

donde K representa a un vector de tres parámetros. Probaremos cinco funciones que tienen ciertas características. La función inicial que se llamará también original, tiene los siguientes parámetros: $K = [0.1550\ 0.1120\ 0.0093]$. Los parámetros de cada función son: $K_1 = [0.0250\ 0.1120\ 0.0093]$, $K_2 = [0.1550\ 0.1120\ 0.0500]$, $K_3 = [0.0250\ -0.1500\ 0.0110]$, $K_4 = [0.1650\ 0.1430\ 0.0089]$.

Para las siguientes gráficas de funciones, la función original siempre aparecerá con asteriscos. Las demás funciones aparecerán en forma continua. La función original (la función con los parámetros K) se usará como referencia respecto a las similitudes y diferencias que existe en esta función respecto a las demás. Las relaciones que existen entre las funciones son: K_1 es más alta que K, K_2 tiene una similitud con K en una parte del dominio (al final), K_3 es diferente a K. La función K_4 es muy similar a la función K en casi todo el dominio. La figura 10.11 muestra la forma de algunas de las funciones.

Procedimientos de la experimentación

El sistema exaptivo empieza con una memoria M de 10 elementos (filas) en donde se codifican 15 parámetros de 8 bits, lo cual da lugar a 120 bits (columnas). Cada

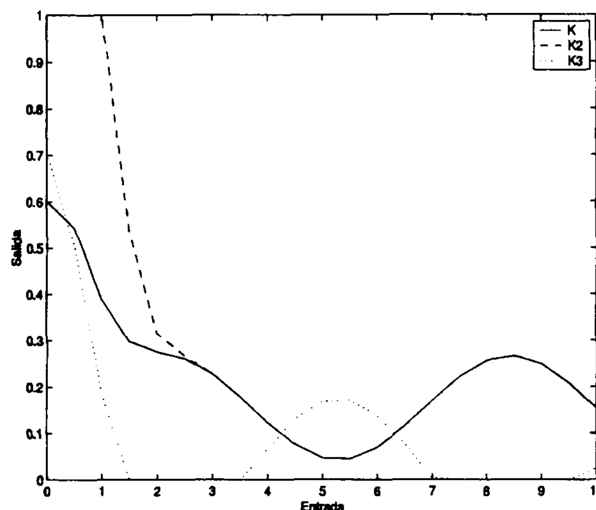


Figura 10.11: Respuesta de las funciones K, K_2 y K_3.

elemento de la memoria tiene a su vez, un peso de uso que está representado en un vector P_U. I_x representa al mejor individuo encontrado durante la búsqueda local o global. El procedimiento toma como entrada las estructuras M, P_U y K y a la salida se obtienen nuevamente las estructuras actualizadas de la memoria y de los pesos de uso.

El algoritmo genético simple se usa para los mecanismos de búsqueda o de modificación. El algoritmo tiene 100 individuos y se aplica una probabilidad de cruce de 0.85 y de mutación de 0.052. Se usa una selección por torneo de tamaño dos. Para la modificación se usa un sembrado poblacional elitista en el 30% de la población en donde se insertan vecindades de la solución reutilizada con una distancia de 0.05. En el capítulo 7 se revisaron las técnicas de sembrado.

El primer experimento consiste en la observación del desempeño del sistema cuando aprende nuevas funciones. Se usarán los parámetros K, K_1, K_2, K_3 y K_4 porque cada uno representa a una función distinta. La memoria empieza con valores aleatorios y con pesos de uso igual a 1. El margen en donde se considera que se está reconociendo una función es de 0.85; esto quiere decir que cualquier valor de evaluación de un conjunto de parámetros de la memoria que supere el margen se considerará como reconocido (ver procedimiento de evaluación de la figura 10.6.) El primer paso consiste en usar los parámetros del vector K. El resultado que se obtiene es que no se reconoció e I_x se almacenó en la primera posición de la memoria. En el segundo paso se usan los parámetros del vector K_1. Se obtiene el mismo resultado ya que tampoco se reconoció la nueva función, por lo cual la aprende sin reutilizar alguna información de la memoria. El mejor individuo encontrado se almacena en la segunda posición de la memoria. Para las funciones con los parámetros K_2 y K_3 se obtiene el mismo resultado. Los mejores parámetros encontrados se almacenan en las posiciones 3 y 4 respectivamente. La

función con los parámetros K_4, el sistema lo reconoce; es decir, en la memoria existen parámetros que aproximan la función con los parámetros K_4. La similitud existe en la primera posición de la memoria. La función K_4 es similar a la función K. La memoria siempre almacena cada nuevo conocimiento en la memoria M en forma secuencial, por lo tanto, los parámetros encontrados se almacenan en la quinta posición de la memoria.

El segundo experimento consiste en probar que el sistema reconoce el conocimiento previo, es decir, que no olvida las funciones previamente aprendidas. En el sistema nuevamente se dan las cinco funciones desde K hasta K_4. Para la primera función K se obtiene una alta evaluación de 0.9590 y se reconoce en la quinta posición de la memoria ya que existe una similitud de la función con parámetros K con la función de parámetros K_4. El sistema realiza una búsqueda local a partir de los parámetros de la quinta posición de la memoria y se obtiene un vector de parámetros que se almacena en la sexta posición (se está realizando un almacenamiento secuencial). Se empieza a introducir cada función con los parámetros K_1 al K_4. La función K_1 si se reconoce y se almacena el resultado del sistema en la séptima posición. La función K_2 también se reconoce ya que es similar a los parámetros que están colocados en la tercera posición de la memoria. La solución del algoritmo se almacena en la octava posición de la memoria. Para la función K_3 su evaluación es inferior al margen preestablecido, así que no se reconoce la función; por lo tanto, se buscan nuevos parámetros que son almacenados en la novena posición. La función K_4 se reconoce en la quinta posición de la memoria ya que en el primer experimento se grabó en esa posición. La solución se almacena en la décima posición de la memoria. El espacio de la memoria se ha terminado. En casi todas las funciones se usa la búsqueda local. Es un indicativo de que se reconocen las funciones y no es necesario empezar con valores aleatorios. Cada red neuronal que reconoce a una función en particular se representa en la memoria.

En el tercer experimento se quiere verificar cómo se perfecciona el aprendizaje de una función en particular cuando su presencia se repite. Usaremos a la función K_3 para verificar que realmente mejora el aprendizaje de la función. Lo que se realiza es que se presenta la misma función K_3 al sistema en cuatro ocasiones. Cada ocasión es un paso del experimento. En el primer paso del experimento se reconoce en la cuarta posición de la memoria y se almacena en la primera posición. En el segundo paso del experimento la función se reconoce en la primera posición y se almacena en la segunda posición. En el tercer paso del experimento se reconoce en la segunda posición 2 y se almacena en la tercera posición. Finalmente en el cuarto paso del experimento se reconoce en la tercera posición y se almacena en la cuarta. El desempeño aumenta ya que desde la primera ejecución la evaluación era de 0.89 y subió hasta 0.95 (ver procedimiento de evaluación de la figura 10.6.)

En el último experimento se quiere evaluar el desempeño promedio de una red neuronal que usa un algoritmo genético que inicia en forma aleatoria para compararlo con un sistema seudoexaptivo neuronal. El procedimiento consiste en presentarle una a una las cinco funciones K, presentando primero la función K, luego K_1, K_2, K_3 y K_4. Al presentarse una función se ejecuta el algoritmo genético simple con un reinicio aleatorio, también se ejecuta el sistema exaptivo para aprender la misma función. El desempeño de cada algoritmo se guarda. El proceso se repite veinte veces con la

finalidad de presentarle por cuatro veces la misma función a cada algoritmo, siempre en el mismo orden. El desempeño promedio después de la ejecución consiste en la evaluación promedio del mejor individuo encontrado en cada instante de tiempo T, es decir $(e_p(T) = \frac{1}{T} \cdot \sum_{t=1}^{T} e(t))$, donde $e(t)$ es la mejor evaluación en cada instante de tiempo t. El número de datos usados para obtener el promedio crece con el tiempo. La figura 10.12 muestra el desempeño de ambos sistemas. Al principio ambos algoritmos tienen el mismo desempeño ya que el sistema seudoexaptivo neuronal (SEN) no tiene ningún conocimiento almacenado. Conforme pasa el tiempo (la función cambia cada 25 generaciones) el sistema seudoexaptivo incrementa su eficiencia mientras que el algoritmo genético sigue comportandose igual. Es de esperar que la curva de desempeño de sistema seudoexaptivo siga incrementando.

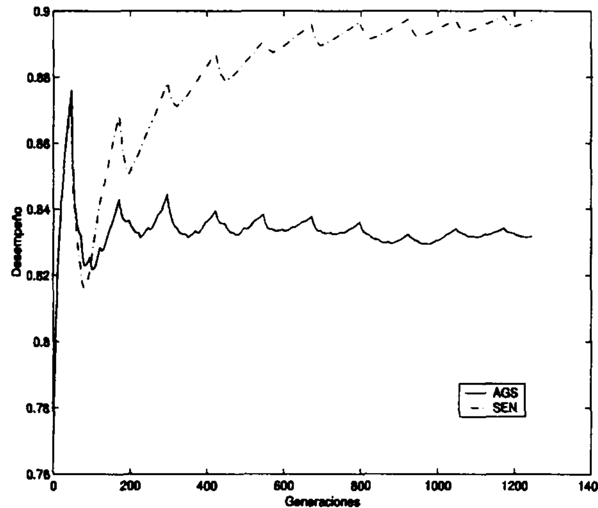


Figura 10.12: Desempeño de la red neuronal basado en un algoritmo genético simple y en un sistema seudoexaptivo.

10.5 Conclusiones

Se ha sustituido el proceso de aprendizaje por una sistema seudoexaptivo. Las ventajas de estos sistemas son evidentes. La memoria de parámetros junto con el mecanismo de reconocimiento, permite el aprendizaje de una función cambiante o el aprendizaje de múltiples funciones.

Otra ventaja es el tiempo de búsqueda que se reduce considerablemente tomando en cuenta que el sistema ya tiene en su memoria soluciones similares. El tiempo en que se alcanza la solución óptima usualmente es alto, aunque es menor cuando existe reutilización de soluciones.

Una característica que se observa en estos sistemas es su capacidad de especialización. El sistema aprende cada vez mejor el conocimiento más reciente (se puede decir

que el más útil), sin embargo va olvidando o desechando el conocimiento que no le es útil. Se puede decir que el sistema tiende a especializarse cada vez más en una función en particular, es decir, tiene una alta plasticidad en su estructura que le permite al sistema incrementar su capacidad.

Capítulo 12

Conclusiones, contribuciones y trabajo futuro

12.1 Conclusiones

La exaptación es un mecanismo económico de la naturaleza para permitir la adaptación de cualquier ser vivo en un tiempo corto y en la forma más adecuada posible. Se considera que la adaptabilidad de un ser vivo tiene relación con la solución a un problema. Existen estructuras en un ser vivo que le permiten sobrevivir a pesar de que la estructura tuvo otro fin u otro origen. Así, una solución puede ser útil en un problema que tenga relación a otro resuelto con anterioridad o que no lo tenga.

Cuando se trabajan con algoritmos evolutivo se asume un medio ambiente estático o al menos que sus cambio no son muy bruscos. Lo que se logran son soluciones muy cercanas al óptimo. Se sabe que el medio ambiente no es estático y que la mayor parte de los problemas interesantes involucran un medio ambiente dinámico. Si el medio ambiente permanece estático mientras se busca una solución adecuada entonces se puede usar cualquier algoritmo evolutivo sin que sea necesario recurrir a la información de pasado, es decir, se puede empezar en forma aleatoria; sin embargo, si el medio ambiente cambia de una forma imprevesible inclusive en el momento de la ejecución de un algoritmo evolutivo, entonces se tendría que detener la búsqueda implicando tener soluciones de baja calidad. Si al detectar un cambio en el medio ambiente, reiniciamos la población de forma aleatoria perderemos mucha información que podría ser útil. Es en donde se hace necesario recurrir a algoritmos que reutilizan y retengan información que fue útil en el pasado para tener siempre y en todo momento la mejor solución posible.

Los algoritmos que utilizan exaptación son una alternativa para tener rápidamente una solución en un tiempo muy corto. Por medio de las técnicas de sembrado se puede reutilizar una solución con más eficiencia. Con los mecanismos de memoria se pueden retener soluciones útiles en el pasado y con mecanismos de reconocimiento que se basan en detectar cambios en la utilidad que brinda una solución actual, se puede tener sistemas seudoexaptivo capaces de optimizar funciones dinámicas o de crear sistemas

de aprendizaje que interactúen en un medio ambiente dinámico.

A pesar de la eficiencia que puede existir, también existen riesgos. Si el ambiente cambia de manera muy brusca existirá un tiempo largo para que algún algoritmo con exaptación brinde una solución adecuada porque cada vez que se repitan condiciones ambientales, existirá una solución que se mejorará un poco cada vez, con la condición de que no sean muchos los cambios y que exista alguna similitud entre otros sucedidos en el pasado. En caso de existir un cambio relativamente suave o que se repita (cíclico) entonces podemos esperar que el sistema llegue poco a poco a un conocimiento total del sistema y logre brinda soluciones casi de inmediato. La red neuronal del capítulo 10 es un ejemplo en donde se muestra que el desempeño de la red aumenta poco a poco en el tiempo.

Si colocamos mecanismos de memoria basados en la similitud que existe entre dos problemas, se debe tener precaución en la vecindad, es decir, en que momento el problema es nuevo o es uno ya conocido cuya solución puede reutilizarse. Si la distancia es muy pequeña, entonces la capacidad de la memoria puede terminarse ya que tendría mucha información muy parecida entre sí. Si la distancia es muy grande, entonces se puede llegar al problema de generalizar demasiado y dar siempre la misma solución para un mismo tipo de problemas y en el momento de reutilizarla, podría no ser una buena solución.

Cuando se usan mecanismos de reconocimiento basados en la evaluación (en nuestro caso es el mecanismo utilizado por estar inspirado en la exaptación), se tiene el mismo problema, ya que si el valor de evaluación de una solución en particular es alto, entonces debería utilizarse, pero la pregunta es: ¿Que tal alto debe ser el valor de evaluación para que pueda reutilizarse? En los algoritmos propuestos para la optimización de funciones dinámicas, cada vez que existía un cambio en la función de optimización, se generaba una población aleatoria en donde se sembraban variaciones y vecindades del individuo con el valor de evaluación más alto en la nueva función. Para el sistema basado en redes neuronales se usó también el valor de evaluación, en donde si superaba un margen preestablecido, entonces se consideraba como útil para ser reutilizado. En el problema del agente inteligente seudoexaptivo, el reconocimiento por evaluación se utilizó de manera diferente a los dos anteriores ya que primero se generaban soluciones aleatorias y soluciones generadas por heurísticas. Todas las soluciones se evaluaban y se elegía la solución con alto valor de evaluación para sembrarla en una población de soluciones aleatorias en forma de vecindades y variaciones. Con lo anterior se puede notar que no existe un método bien definido para establecer un reconocimiento de que un problema es similar a otro resuelto en el pasado.

El mecanismo de memoria resuelve muchos problemas relacionados con la retención, sin embargo atrae otros ya que es necesario un mecanismo que administre la memoria. Branke sugiere almacenar cada determinado número de generaciones la mejor solución encontrada (capítulo 8), Sushil Louis sugiere almacenar un conjunto de soluciones de baja calidad para tener puntos de retorno en caso de que la función cambie bruscamente y con ello dejar algunos esquemas intactos (capítulo 8). A pesar de las variantes que pueden realizarse, el uso de la memoria hace de un sistema evolutivo una entidad artificial, no inspirada en la naturaleza sino de heurísticas, en otras palabras

no es un sistema exaptivo puro.

Para lograr tener sistemas exaptivos puros, es necesario revisar a detalle el problema de la diversidad, los nichos e islas ya que se podrían tener mecanismos implícitos de memoria sin aditamentos o procesos que la administren. Otra forma es la representación del cromosoma, ya que se usan formas muy simples y como se sabe, en los cromosomas reales existen muchas redundancias, genes repetidos o inclusive genes que dan deformaciones o apéndices que no dan una utilidad o ventaja. Considero que el camino para encontrar un sistema exaptivo puro se dirige hacia algoritmos que usen una representación diferente, mantengan la diversidad o se basen en nichos, subpoblaciones o islas.

12.2 Contribuciones

Las contribuciones más importantes de la tesis son dos, la primera es una estabilización en las técnicas de sembrado, para su clasificación y descripción. La segunda contribución es un sistema alternativo para resolver problema dinámicos ya sea optimización de funciones dinámicas o aprendizaje en un medio ambiente dinámico. El sistema se basa en la teoría de la exaptación y como se implantó usando métodos explícitos, se le llama sistema seudoexaptivo.

Las técnicas de sembrado son la forma más natural para reutilizar soluciones en un algoritmo genético. El estudio de las diversas técnicas ha permitido establecer una clasificación de las técnicas. Se proponen tres técnicas, la elitista, la variacional elitista y la poblacional. La técnica de sembrado elitista es la forma más fácil de reutilizar una solución ya que sólo es necesario contar con una solución y se inserta directamente en la población. El sembrado variacional tambien requiere de una solución, sin embargo a partir de esta solución se generan variaciones o mutaciones (usando el argot evolutivo) que son insertados en la población. La técnica exige un proceso de variación que depende de parámetros y además es necesario establecer el impacto que se quiere hacer al cromosoma. La forma más común es aplicar el mismo proceso de mutación con alta probabilidad, sin embargo se pueden aplicar diversos criterios que se analizaron en el capítulo 7. El sembrado variacional brinda una ventaja sobre un inicio aleatorio sobre todo si la función dinámica que se optimiza realiza cambios muy bruscos. Finalmente las técnicas de sembrado poblacional requiere de muchas soluciones que son insertadas en la población. Cuando se tiene una solución, entonces es necesario generar vecindades de la misma. La técnica al parecer es la más eficiente cuando existen cambios suaves en la función de optimización, sin embargo, existe un costo en la programación de los métodos para generar vecindades ya que dependen mucho del problema que se este resolviendo tal cual se analizó en el capítulo 7.

El sistema seudoexaptivo es un algoritmo evolutivo con capacidades de retención y reutilización de soluciones. Un sistema exaptivo tiene procesos implícitos de reconocimiento, búsqueda, modificación y almacenamiento que se hacen explícitos al implantarse con técnicas de sistemas inteligentes.

Para tener exaptación es necesario incluir mecanismos que detecten cambios en

la función objetivo, reutilicen la mejor solución que se disponga generando varias soluciones por medio de variaciones y vecindades y se retengan usando algún medio de memoria o por medio de algún mecanismo que mantenga una diversidad

El sistema seudoexaptivo permite tener soluciones aceptables en un periodo de tiempo muy corto. El sistema va mejorando su desempeño constantemente al interactuar con su medio, por lo cual la calidad de una solución aumenta en todo momento. Si la función cambia constantemente, entonces será más difícil que la calidad de la solución aumente sobre todo si no existe repetibilidad ni semejanza en la función objetivo.

La arquitectura mostrada en el capítulo 6 puede usarse otros métodos que no presisamente se basen en la computación evolutiva, es decir, se pueden usar otros mecanismos de reconocimiento, mecanismos de almacenamiento, de búsqueda global y búsqueda local. Algunos mecanismos son señalados en el capítulo 8.

12.3 Trabajo futuro

Es necesario seguir analizando las técnicas de sembrado ya que existen técnicas como el generador de vecindades en donde no se probó a detalle el método de variación parcial ya que con saber que genes se deben cambiar y que genes no, se podría mantener ciertos esquemas que podrían ser útiles y alterar a otros esquemas que no le dan una utilidad y así obtener soluciones de mejor calidad en menos tiempo.

El sistema seudoexaptivo usa mecanismos de reutilización y retención explícitos. Un objetivo claro a futuro es la creación de un sistema evolutivo con exaptación implícita. Como se mencionó arriba esto quiere decir que se requieren mecanismos que permitan mantener una diversidad entre los individuos de la población , otra es el estudio de representaciones que faciliten la exaptación, es decir, con genes redundantes, o sin utilidad aparente que pueden ser activados en algún momento específico. Una tercera posibilidad es usar algoritmos evolutivos que generen nichos, trabajen en islas o generen subpoblaciones, cada conjunto de nicho, subpoblación o isla dedicado a un conjunto de problemas similares. Si hay un nuevo problema, pues se activa un nuevo conjunto. Tal vez si mezclamos los conceptos de nuevas representaciones y los conjuntos de islas, nichos o subpoblaciones se lograría tener un sistema exaptivo puro.

Existen problemas de aprendizaje dinámico en donde podría ser de mucha utilidad un sistema seudoexaptivo. Por ejemplo, el control de inventarios en donde existen variables dinámicas como la demanda que depende de muchos factores como la edad del consumidor, la moda, la temporada, la zona, etc. Una persona con experiencia podría determinar la política adecuada para determinado momento o condición del proceso ya sea para aumentar el inventario o para disminuirlo. Al colocar un sistema seudoexaptivo que apoye las desiciones de la persona que controla el proceso, le brindaría políticas que poco a poco serían de igual valor e inclusive superiores a los de la misma persona. Si la persona deja el proceso por cualquier circuntancia, entonces la nueva persona podría tener la seguridad de dar la política acertada al usar el sistema de apoyo y así acelerar el aprendizaje para controlar el proceso. De otra manera las políticas que tome la nueva persona estarían cargadas de incertidumbre y aunque después de un tiempo llegaría a

tener la eficiencia de la persona que sustituyó, ese tiempo representarían pérdidas para la empresa de cuyo proceso dependen.

Finalmente se tiene un interés muy particular por los mecanismos de aprendizaje basados en redes neuronales; se dice que el cerebro es un sistema exaptivo por excelencia, entonces se puede investigar la forma de crear sistemas exaptivos basados en redes neuronales en donde puedan aprender secuencias, patrones, clasificaciones y relaciones funcionales en una misma estructura. Una estructura así podría ser muy útil porque se llegaría a la creación de sistemas multifuncionales ya que actualmente todavía tenemos métodos muy concretos para una familia muy reducida de problemas. Otra tendencia es lograr una unificación de todas las teorías evolutivas para desarrollar un algoritmo evolutivo único. El objetivo es ambicioso, pero como todo, se puede lograr con un proceso evolutivo de investigación.

Bibliografía

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Barto, A. G., Bradtke, S. J., y Singh, S. P. (1993). *Learning to act using real-time dynamic programming* (Tech. Rep. No. UM-CS-1993-002).

Bentley, P. J. (1999). *Evolutionary design by computer.* San Francisco, CA: Morgan Kaufmann.

Branke, J. (1999a). Evolutionary approaches to dynamic optimization problems; a survey. In *Gecco workshop on evolutionary algorithms for dynamic optimization problems* (pags. 134–37). Orlando, FL.

Branke, J. (1999b). Memory-enhanced evolutionary algorithms for dynamic optimization problems. In *Congress on evolutionary computation (CEC'99)* (pags. 1875–1882). IEEE.

Branke, J. (2001a). Evolutionary approaches to dynamic optimization problems - updated survey. In J. Branke y T. Bäck (Eds.), *Evolutionary algorithms for dynamic optimization problems* (pags. 27–30). San Francisco, California, USA.

Branke, J. (2001b). Multi-population approach to dynamic optimization problems. In I. Parmee (Ed.), *Adaptive computing in design and manufacture (acdm 2000)* (pags. 299–308). Springer.

Cairns, O., y Miller. (1988). The origin of mutants. *Nature, 335*, 142–145.

Campbell, D. T. (1960). Blind variation and selective retention in creative thought as in other knowledge processes. *Psychological Review, 67*, 380–400.

Campbell, D. T. (1974). Evolutionary epistemology. In P. Schilpp (Ed.), *The philosophy of karl popper, vol I* (pags. 413–63). La Salle, Illinois: Open Court.

Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In J. C. R.S. Michalsky y T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pags. 371–392).

Carpenter, G., Grossberg, S., y Reynolds, J. (1991). ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks, 4*, 565–588.

Carpenter, G., Grossberg, S., y Rosen, D. (1991). *Fuzzy ART: An adaptive resonance algorithm for rapid, stable classification of analog patterns* (Tech. Rep. No. No. CAS/CNS-TR- 91-006). Boston, MA: Boston University.

Carpenter, G. A., y Grossberg, S. (1987a). A massively parallel architecture for a

self-organizing neural pattern recognition machine. *Computer vision, graphics, and image processing, 37*, 54–115.

Carpenter, G. A., y Grossberg, S. (1987b). ART 2: Self organization of stable category recognition codes for analog input patterns. *Applied Optics: Special Issue on Neural Networks, 26*, 4919–4930.

Carpenter, G. A., y Grossberg, S. (1990). ART 3: Hierarchical search using chemical transmitters in self- organizing pattern recognition architectures. *Neural Networks, 3*, 129–152.

Chan, W. T., y Hu, H. (2000). Precast production scheduling with genetic algorithms. In *Proceedings of the 2000 congress on evolutionary computation, vol. 2* (pags. 1087–1094). La Jolla, CA.

Christopher K. Riesbeck, R. C. S. (1989). *Inside case-based reasoning.* NJ: Morgan Kaufmann.

Cziko, G. (1995). *Without miracles: Universal selection theory and the second darwinian revolution.* Cambridge, MA: MIT Press.

Darwen, P. J., y Yao, X. (1995). On evolving robust strategies for iterated prisoner's dilemma. In X. Yao (Ed.), *Lecture notes in computer science, vol. 956* (pags. 276–292). Heidelberg, Germany: Springer-Verlag.

Darwin, C. (1997). *El origen de las especies.* México: UNAM.

Dawkins, R. (1991). *The selfish gene.* New York: Oxford University Press.

DeJong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. *Dissertation Abstracts International, 36(19)*, 5140B. (University Microfilms No. 76-9381)

Edelman, G. M. (1987). *Neural darwinism.* New York: Basic Books.

Eldredge, N., y Gould, S. J. (1972). Punctuated equilibria: An alternative to phyletic gradualism. In T. Schopf (Ed.), *Models in paleobiology* (pags. 82–115). San Francisco, CA: Freeman, Cooper and Company.

Fogel, D. B. (1991). *System identification through simulated evolution: A machine learning approach to modeling.* Needham Heights, MA: Ginn Press.

Fogel, L., Owens, A., y Walsh, M. (1966). *Artificial intelligence through simulated evolution.* New York: John Wiley and Sons.

Freeman, J. A., y Skapura, D. (1992). *Neural networks: Algorithms, applications, and programming techniques.* Reading, MA: Addison Wesley.

Gero, J. S., y Kazakov, V. A. (1998). Evolving design genes in space layout planning problems. *Artificial Intelligence in Engineering, 12(3)*, 163–176.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning(doctoral dissertation, university of michigan). *Dissertation Abstracts International, 44(10)*, 3174B. (University Microfilms No. 8402282)

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley Pub. Co.

Gould, S. J. (1991). A crucial tool for an evolutionary psychology. *Journal of Social Issues, 47*, 43–65.

Gould, S. J., y Vrba, S. (1982). Exaptation: A missing term in the science of form. *Paleobiology, 8*, 4–15.

Graubard, S. R. (1993). *El nuevo debate sobre la inteligencia artificial. Sistemas simbólicos y redes neuronales.* México: Editorial Gedisa.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. *In Genetic algorithms and simulated annealing, L. D. Davis(Ed) London:Pitman.*

Harik, G. R., Lobo, F. G., y Goldberg, D. E. (1999). The compact genetic algorithm. *Evolutionary computation, IEEE, 3(4).*

Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2 ed.). NJ: Prentice Hall.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Ann Arbor: University of Michigan Press.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, y T. M. M. (Eds.) (Eds.), *Machine learning II* (pags. 593–623). Los altos, CA: Morgan Kaufmann.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., y Thagard, P. R. (1986). *Induction: Processes of inference, learning, and discovery.* Cambridge, MA: MIT Press.

Kawakami, Y., T.; Kahazu. (1996). A study on evolutionary synthesis of classifier system architectures. *Proceedings of IEEE International Conference on Evolutionary Computation.*, 155–160.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, *4*, 461–476.

Kolodner, J. (1993). *Case-based reasoning.* CA: Morgan Kaufmann.

Koza, J. R. (1990). Genetically breeding populations of computer programs to solve problems in artificial intelligence. In *Proceedings of the second international conference on tools for AI, herndon, virginia* (pags. 819–827). IEEE Computer Society Press, Los Alamitos, CA, USA.

Koza, J. R. (1993). *Genetic programming: on the programming of computers by means of natural selection.* Cambridge, MA: MIT Press.

Koza, J. R. (1994). *Genetic programming II: Automatic discovery of reusable programs.* Cambridge, MA: MIT Press.

Leake, D. B. (1996). *Case-based reasoning: Experiences, lessons & future directions.* Cambridge: MIT Press.

Lewontin, R. C. (1970). The units of selection. *Annual Review of Ecology and Systematics*, *1*, 1–18.

Liu, X. (1996). *Combining genetic algorithm and casebased reasoning for structure design.* Tesis de Maestría no publicada, University of Nevada, Department of Computer Science., Reno, Nevada.

Louis, S. (1997). Working from blueprints: Evolutionary learning for design. *Artificial Intelligence in Engineering*, *3*(11), 335–341.

Louis, S., y Li, G. (1997). Augmenting genetic algorithms with memory to solve traveling salesman problems. In P. Wang (Ed.), *Proceedings of the joint conference on information sciences* (pags. 108–111). Duke, CA: Duke University Press.

Louis, S. J., y Johnson, J. (1997). Solving similar problems using genetic algorithms and case-based memory. In T. Back (Ed.), *Proceedings of the seventh international conference on genetic algorithms (icga97)* (pags. 283–290). San Francisco, CA: Morgan Kaufmann.

Louis, S. J., y Johnson, J. (1999). Robustness of case-initialized genetic algorithms. In P. Wang (Ed.), *FLAIRS-99*. Orlando, FL: AAAI Press.

Louis, S. J., y Xu, Z. (1996). Genetic algorithms for open shop scheduling and rescheduling. In M. E. Cohen y D. L. Hudson (Eds.), *ISCA 11th Int. Conf. on computers and their applications* (pags. 99–102). , Raleigh, NC: . International Society for Computers and Their Applications.

Louis, S. J. (1993). *Genetic algorithms as a computational tool for design.* Dissertación Doctoral no publicada, Department of Computer Science, Indiana University.

Mendel, J. M. (1995). Fuzzy logic systems for engineering. *Proceedings of IEEE, March, 83*, 345–377.

Michalewicz, Z. (1999). *Genetic algorithms + data structures = evolution programs.* New York: Springer-Verlag.

Michalewicz, Z., y Fogel, D. B. (1999). *How to solve it: Modern heuristics.* New York: Springer.

Mondada, F., y Floreano, D. (1995). Evolution of neural control structures: Some experiments on mobile robots. *Robotics and Autonomous System, 16*, 183–195.

Montana, D. J., y Davis, L. D. (1989). Training feedforward networks using genetic algorithms. In E. E. Daniel (Ed.), *In proceeding of the international joint conference on artificial intelligence* (pags. 762–767). Detroit, MI.

Nabet, B., y Pinter, R. B. (1991). *Sensory neural networks: Lateral inhibition.* Boca Raton: CRC press.

Nguyen, H. T., y Walker, E. A. (1997). *A first course in fuzzy logic.* Boca Raton: CRC Press.

Ordoñez, I. (1991). *Soluciones al problema del vendedor viajero generalizado mediante algoritmos genéticos con operadores especiales de cruce.* Tesis de Maestría no publicada, Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Monterrey., Nuevo León, México.

Pelikan, M., Goldberg, D. E., y Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, y R. E. Smith (Eds.), *Proceedings of the genetic and evolutionary computation conference GECCO-99* (pags. 525–532). Orlando, FL: Morgan Kaufmann Publishers, San Francisco, CA.

Piaget, J. (1980). *Adaptation and intelligence: Organic selección and phenocopy.* Chicago: University of Chicago Press.

Pinedo, M. (1995). *Scheduling: Theory, algorithms, and systems.* Englewood Cliffs, NJ: Prentice Hall.

Plotkin, H. (1997). *Darwin machines.* Cambridge, MA: Harvard University Press.

Potter, M. A., De Jong, K. A., y Grefenstette, J. J. (1995). A coevolutionary approach to learning sequential decision rules. In L. Eshelman (Ed.), *Proceedings of the*

sixth international conference on genetic algorithms (pags. 83–92). San Francisco, CA: Morgan Kaufmann.

Ramsey, C. L., y Grefenstette, J. J. (1993). Case-based initialization of genetic algorithms. In S. Forrest (Ed.), *Proc. of the Fifth Int. Conf. on Genetic Algorithms* (pags. 84–91). San Mateo, CA: Morgan Kaufmann.

Rumelhart, D. E., McClelland, J. L., y PDP Research Group the. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition, vol 1 and 2.* MA: MIT Press.

Russell, S., y Norvik, P. (1995). *Artificial intelligence: A modern approach.* Upper saddle river, NJ: Prentice-Hall.

Saravanan, N., y Fogel, D. B. (1995). Using evolutionary programming to create neural networks. *IEEE Expert, 10(3)*, 23–27.

Schoenauer, M., y Xanthakis, S. (1993). Constrained GA optimization. In S. Forrest (Ed.), *Proc. of the Fifth Int. Conf. on Genetic Algorithms* (pags. 573–580). San Mateo, CA: Morgan Kaufmann.

Schwefel, H. P. (1981). *Numerical optimization of computer models.* Chichester, UK: John Wiley.

Sharif, A. M., y Barrett, A.Ñ. (1998). Utilising knowledge for optimum mesh design. *IEE Colloquium on Knowledge Discovery and Data Mining, 173*(Digest No. 1998/310), 4/1-4/5.

Smith, R. E., y Cribbs, H. B. (1994). Is a learning classifier system a type of neural network? *Evolutionary Computation, 2*(1), 19–36.

Sutton, R. S. (1998). Reinforcement learning: Past, present and future. In *SEAL* (pag. 195-197).

Thrun, S. (1995). *Lifelong learning: A case of study* (Tech. Rep. No. CMU-CS-95-208). New York: Carnegie Mellon University.

Thrun, S., y O'Sullivan, J. (1995). *Clustering learning tasks and the selective cross-task transfer of knowledge* (Tech. Rep. No. CMU-CS-95-209). New York: Carnegie Mellon University.

Torres-Treviño, L. M. (1998). *Controladores dinámicos con la red neuronal por máxima sensibilidad.* Tesis de Maestría no publicada, Universidad Autónoma de San Luis Potosí, México, SLP, México.

Trojanowski, K., y Michalewicz, Z. (1999). Evolutionary algorithms for non-stationary environments. In *Proc. of 8th workshop: Intelligent information systems* (pags. 229–240). ICS PAS Press.

Valenzuela-Rendón, M. (1991). The fuzzy classifier system: A classifier system for continuously varying variables. In *Proceedings of the 4th International Conference on Genetic Algorithms* (pags. 346–353). San Diego, CA: Morgan Kaufmann.

Vic Ciesielski, P. S. (1998). Real time genetic scheduling of aircraft landing times. *The 1998 IEEE International Conference on Evolutionary Computation Proceedings, 2*, 360–364.

Watson, I. (1997). *Applying case-based reasoning: Techniques for enterprise systems.* CA: Morgan Kaufmann.

Waugh, S. (1994). *Dynamic learning algorithms.*

Westerberg, C. H., y Levine, J. (2001). Investigation of different seeding strategies in a genetic planner. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, y H. Tijink (Eds.), *Applications of evolutionary computing, evoworkshops 2001: Evocop, evoflight, evoiasp, evolearn, and evostim* (pags. 505–514). Como, Italy: Springer.

Yu, T., y Bentley, P. (1998). Methods to evolve legal phenotypes. In A. E. Eiben, T. Back, M. Schoenauer, y H.-P. Schwefel (Eds.), *Fifth international conference on parallel problem solving from nature* (Vol. 1498, pags. 280–291). Amsterdam: Springer-Verlag.

Zebulum, R. S., Pacheco, M. A. C., y Vellasco, M. M. B. (2002). *Evolution electronics: Automatic design of electronic circuits and systems by genetic algorithms.* USA: The CRC Press International Series on Computational Intelligence.

Zhou, H. H. (1985). Classifier system with long-term memory in machine learning. In *Proceeding of the first international conference on genetic algorithms and their applications* (pags. 178–182). Pittsburgh, PA.

Zhou, H. H., y Grefenstette, J. J. (1989). Learning by analogy in genetic classifier systems. In J. D. Schaffer (Ed.), *Proceedings of the 3rd international conference on genetic algorithms* (pags. 291–297). Fairfax, Virginia: Morgan Kaufmann.

Capítulo 11

Agentes inteligentes con capacidades exaptivas

11.1 Características de un agente inteligente con exaptación

Un agente lo podemos definir como una entidad capaz de interactuar con su entorno. El agente tiene sensores para percibir cambios en el ambiente y actuadores para modificarlo. El agente tiene en su interior una política a seguir o relación funcional, y un objetivo o meta. Es posible establecer una clasificación de agentes como la que se propone enseguida (Russell y Norvik, 1995):

- Un agente reactivo sólo usa la relación funcional establecida. El agente puede ser más completo si se incluye un modelo del entorno (un estado interno) y así pueda elegir la mejor acción acorde con la situación actual.
- Un agente basado en metas buscará la mejor acción que le permita cumplir con el objetivo establecido. Generalmente para este tipo de agentes, se incluyen mecanismos de búsqueda y planeación para llevar a cabo la meta.
- Un agente basado en utilidades prácticamente es un agente con metas, sólo que en lugar de buscar metas, se busca maximizar una utilidad. Podemos decir que la meta sigue siendo la misma, sólo que ahora podemos ejecutarla maximizando o minimizando ciertos atributos del agente o del entorno.

En este capítulo estamos proponiendo un agente con capacidades seudoexaptivo y de acuerdo a la clasificación anterior, se clasifica como un agente basado en utilidades; sin embargo, tiene algunas características que lo hacen diferente ya que tiene la habilidad de aumentar su capacidad de aprendizaje; puede aprender distintas políticas y reutilizarlas cuando existen condiciones similares a las del pasado.

Los componentes del agente seudoexaptivo son los mismos al del sistema propuesto en el capítulo 5. Usa mecanismos de reconocimiento, de almacenamiento y de

modificación o búsqueda. La figura 11.1 muestra la estructura de un agente con características exaptivas. El comportamiento del agente seudoexaptivo es el siguiente. Generalmente al inicio el agente no tiene conocimiento del proceso. El objetivo consiste en maximizar o minimizar un parámetro propio o de su entorno. Para cumplir con el objetivo, el agente debe construir (llámese aprender) una política o una relación funcional adecuada.

En el agente se representan las relaciones funcionales por medio de reglas, ecuaciones, pesos de conexión de las redes neuronales, etc. Un agente seudoexaptivo usa algoritmos evolutivos para modificar una política o relación funcional con la finalidad de maximizar el objetivo asignado. La característica fundamental es la reutilización de las políticas siempre y cuando exista una similitud entre los procesos en que se desenvuelve el agente y lo que tiene almacenado en su memoria.

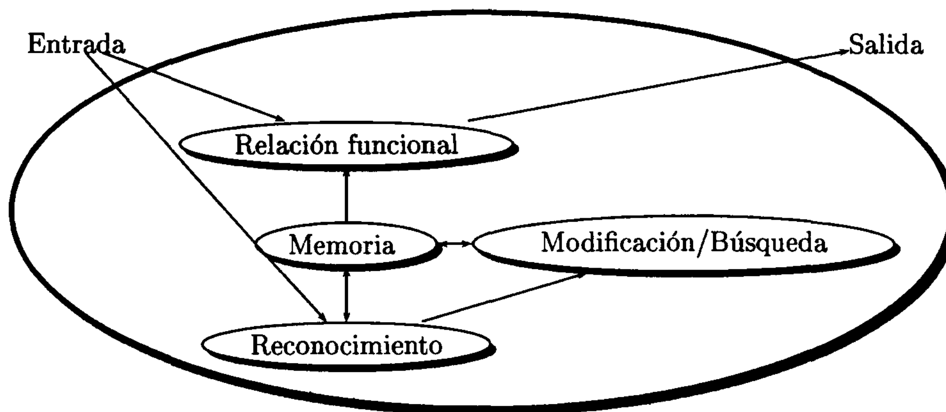


Figura 11.1: Estructura de un agente con características exaptivas.

11.2 El agente seudoexaptivo en la reprogramación de tareas

Se mostrará el desempeño de un agente seudoexaptivo en la reprogramación de tareas en varias máquinas. El objetivo del experimento es demostrar la forma en que puede implantarse un agente en problemas en que las condiciones cambian. Se tiene un conjunto de máquinas y a su vez, se cuenta con una lista de tareas con tiempos de ejecución, de compromiso (en que se debe entregar la tarea ya realizada) y pesos o prioridades. Una tarea que tenga un alto peso debe ser entregada antes que una tarea

de bajo peso considerando que ambas tienen el mismo tiempo de entrega. Cada tarea se ejecuta en una sola máquina y cualquier máquina puede tomar cualquier tarea.

La lista de tareas tiene un orden predefinido, generalmente es por el tiempo en que cada tarea se requiere. El problema consiste en determinar el orden óptimo en que se deben dar las tareas a las máquinas para su elaboración. La máquina se asignará a cada tarea dependiendo de la disponibilidad para ejecutar la tarea, es decir, se escogerá a la máquina que pueda acabar la tarea en el menor tiempo considerando las asignaciones anteriores que tenga cada máquina.

Para determinar que un orden es adecuado, es necesario obtener una función de evaluación. H_E es el tiempo compromiso de la entrega, P_S es el peso por tarea, y T_E es el tiempo en que se entregará la tarea terminada, entonces tenemos que dado un orden predefinido O para cada tarea (en términos logísticos es el pedido pd y N_T es el total de tareas o de pedidos), el tiempo de retraso es:

$$L(O(pd)) = T_E(O(pd)) - H_E(O(pd)) \tag{11.1}$$

Si el valor de L es positivo, se tiene un retardo. Si es negativo es un indicativo de que se está terminando la tarea antes del tiempo compromiso. El tiempo de retardo T se calcula como sigue:

$$T(O(pd)) = \max(L(O(pd)), 0) \tag{11.2}$$

donde $pd = 1 \dots N_T$. Todo retardo es penalizado por un factor o peso, por lo cual, cada vez que exista un retardo es necesario acumularlo en una variable e. Al inicio e tiene un valor de cero, cada pedido o tarea se lee dependiendo del orden que se establezca en el vector O. El problema se reduce a encontrar una secuencia O de tareas o pedidos que minimicen e (Pinedo, 1995). La ecuación 11.3 permite evaluar cada secuencia sugerida en la lista O.

$$e = \sum_{pd=1}^{N_T} (P_S(O(pd))\, T(O(pd))) \tag{11.3}$$

La reprogramación de tareas es un problema muy común en líneas de producción ya que a pesar de tener planes diarios de producción, siempre vienen pedidos a cada momento que son urgentes ya sea por clientes muy especiales, falta de materia prima o por tener mermas. En este momento se debe romper el orden que se estableció y es necesario reconsiderarlo tomando en cuenta las tareas nuevas y las tareas que no se han ejecutado. Las tareas que se han ejecutado y las que estén en proceso no deben tomarse en cuenta (a menos que las tareas consistan en varias subtareas que no es nuestro caso).

Cuando se tienen nuevas tareas es necesario buscar otro orden o secuencia O_N porque las tareas nuevas y las que ya se programaron (o están en proceso) cambian las características del problema. En muchos casos es prohibitivo buscar un nueva secuencia porque la programación de las tareas puede tomar mucho tiempo y se debe tomar una decisión inmediata. La secuencia O puede ser útil añadiéndole las nuevas tareas y eliminando las que ya se elaboraron y las que están en proceso, es decir, se reutiliza de alguna manera el programa o secuencia anterior. El problema se reduce a determinar

la forma en que se añadirán las nuevas tareas a la secuencia O. Una forma consiste en respetar el orden previo y añadirlas al final de la secuencia, por ejemplo, la secuencia anterior es: [1 2 3 4 5 6], las nuevas tareas son: [7 8 9]. Al insertar las nuevas tareas en la secuencia anterior tenemos: [1 2 3 4 5 6 7 8 9]. Otro criterio consiste en añadir las tareas dependiendo del peso que tenga cada una respecto a los pesos de las tareas existentes. Por ejemplo; la secuencia anterior es [1 2 3 4 5 6], los pesos de la secuencia son: [5 4 5 3 1 1] respectivamente. Al insertar en la secuencia anterior las nuevas tareas [7 8 9] con sus pesos respectivos: [1 3 5], tenemos como secuencia nueva [1 9 2 3 8 4 5 7 6] con los pesos [5 5 4 5 3 3 1 1 1] respectivamente. Para la programación de las tareas se considera el uso de un algoritmo genético que reutiliza secuencias, por lo cual, como los mecanismos de cruce y mutación pueden dar soluciones inválidas, se usan operadores especiales de cruce y mutación (Goldberg, 1989; Ordoñez, 1991). Para el funcionamiento del agente seudoexaptivo, es necesario describir la estructura que tiene. El agente seudoexaptivo toma como entrada la hora compromiso, el tiempo de elaboración y el peso o prioridad de cada tarea. El agente también requiere de la secuencia de programación anterior de las tareas y el tiempo de disponibilidad de cada máquina. Como salida el agente brinda la secuencia de programación, el tiempo de entrega y la máquina asignada para cada tarea.

La estructura del agente consiste en una memoria para retener la secuencia anterior incluyendo las máquinas a las cuales se asignaron, tiempos de disponibilidad de cada máquina y la hora de entrega de cada tarea. Tambien tiene un mecanismo para insertar las nuevas tareas en la secuencia de la memoria, un mecanismo que genera una nueva secuencia insertando al final las nuevas tareas y un mecanismo que genera una nueva secuencia insertando en forma ordenada por los pesos cada nueva tarea.

Un procedimiento importante del agente es para la toma de decisiones. El procedimiento primero generan 25 individuos aleatorios que se usarán como población inicial. El siguiente paso consiste en generar 25 individuos resultado de la variación de la secuencia con tareas insertadas al final. Finalmente se generan 25 individuos resultado de la variación de la secuencia con tareas insertadas en forma ordenada por sus pesos. Todos los individuos son evaluados y si resulta mejor un individuo aleatorio, entonces se toman los 25 individuos aleatorios como población inicial del algoritmo genético. Si resulta mejor un individuo de los 25 que fueron generados a partir de la secuencia con tareas insertadas al final, entonces se siembra en forma variacional 8 individuos en la población inicial (corresponde aproximadamente a un sembrado del 30%.) Si resulta mejor un individuo de los 25 que fueron generados a partir de la secuencia con tareas insertadas por orden de sus pesos, entonces se siembra en forma variacional 8 individuos en la poblacion inicial. Finalmente el agente tiene un algoritmo genético con un tamaño de población de 25 individuos y se ejecuta durante 50 generaciones. Se usa un cruce de PMX con una probabilidad de cruce de 0.9 y una probabilidad de mutación de 0.052. Se usa una selección de torneo de tamaño 2.

El agente inteligente tiene las mismas entradas y las mismas salidas con la diferencia de que tiene otra estructura que consiste en un algoritmo genético que inicia con una población aleatoria para encontrar una secuencia que tenga el tamaño de las tareas que quedaron pendientes más las tareas nuevas. Las características del algoritmo

genético son las mismas que las del agente seudoexaptivo.

11.3 Experimentación

En el experimento se quiere probar la eficiencia que tiene un agente que reutiliza secuencias de tareas, respecto a otro que no lo hace. Suponemos que al inicio del día se programan 100 tareas, posteriormente existe un tiempo de corte donde se incluyen nuevas tareas que deben ser programadas bajo las condiciones de disponibilidad que se tengan en las máquinas. El escenario indica que siempre se reprogramarán cada día 10 tareas nuevas considerando las 100 iniciales. Se probará el desempeño de ambos agentes durante 100 días y se graficará el promedio del desempeño de cada agente en términos de costos. El experimento tiene varios pasos.

El primer paso consiste en generar aleatoriamente 100 tareas con diferentes pesos, tiempos de elaboración y horas compromiso. El margen de cada peso es de 1 a 5, las horas compromiso son de 5 a 300 unidades y el tiempo de elaboración es de 4 a 20 unidades de tiempo. Todos los valores son generados aleatoriamente de manera uniforme. Ambos agentes programan todas las tareas en 10 máquinas usando un algoritmo genético simple con cruce PMX durante 50 generaciones con 100 individuos. Se utiliza una selección de torneo de tamaño 2. Las probabilidades de cruce y mutación es de 0.8 y 0.052 respectivamente. El proceso de mutación consiste en un intercambio de tareas entre dos posiciones aleatorias de una secuencia. El cruce de un punto puede ocasionar secuencias inválidas, es por eso que se opta por usar el cruce de PMX que es un mecanismo de cruce para obtener secuencias válidas a partír de dos secuencias (Goldberg, 1989). Al ejecutarse el algoritmo genético al final de las 50 generaciones se obtiene una secuencia solución que se aplica al instante asignando cada tarea en el orden definido en la solución.

En el siguiente paso se establecer el tiempo de corte, es decir, el momento en que llegan 10 tareas nuevas generadas aleatoriamente pero con un tiempo de entrega posterior al tiempo de corte. Se determinar las tareas que están en proceso así como las que se terminaron para eliminarlas de la secuencia solución. Se calcular el tiempo de disponibilidad de cada máquina en el momento del corte, es decir, del inicio de la reprogramación. No se deben considerar las tareas ya concluidas o en proceso y por lo tanto son eliminadas de la secuencia solución. Para el siguiente paso se determinar el total de tareas a programar considerando las tareas que quedaron pendientes en la secuencia solución y las nuevas que llegaron en el tiempo de corte. Se utiliza el agente inteligente para encontrar una nueva programación por medio de un algoritmo genético, primero con poblaciones aleatorias. También se utilizar el agente seudoexaptivo reutilizando la secuencia solución sin que contenga las tareas que estaban en proceso o terminadas. El agente modificará la secuencia solución insertando las nuevas tareas al final de la secuencia o de acuerdo a los pesos de las tareas.

En el último paso se almacena el desempeño de ambos agentes en la reprogramación de las tareas nuevas y las pendientes considerando el costo de la mejor secuencia encontrada de cada uno de los agentes. La programación del día termina.

Todos los pasos se ejecutan 100 veces para obtener el desempeño de ambos agentes por 100 días. Al final se obtiene el desempeño promedio de cada agente en esos días.

El desempeño de ambos agentes para 100 días se muestra en la figura 11.3. La figura es una muestra clara que la exaptación si le brinda ventaja al agente exaptivo porque puede decidir entre empezar igual que el agente inteligente, es decir, en forma aleatoria o reutilizar la secuencia anterior sembrándola en la población inicial.

El costo de la programación diaria se muestra en la figura 11.2. La diferencia es pequeña,sin embargo existe un ahorro en el desempeño del agente seudoexaptivo en un promedio de 60 unidades monetarias diarias lo cual significa un ahorro promedio de 6000 unidades monetarias.

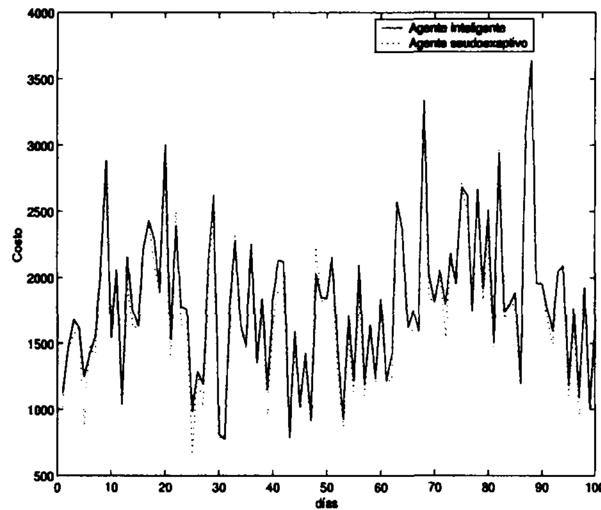


Figura 11.2: Costos obtenidos en la programación de las tareas de cada agente en 100 días.

11.4 Conclusiones

Los agentes inteligentes pueden ser enriquecidos en su comportamiento cuando tienen la capacidad de reutilizar y de retener conocimiento que le es útil. El agente seudoexaptivo que reprograma tareas sólo tiene la capacidad de retener la solución anterior pero tiene la capacidad de decisión para reutilizarla o empezar con otra solución dependiendo de las características que tengan las nuevas tareas. La decisión parece ser muy simple, sin embargo es suficiente para ahorrar un costo significativo en cuanto al tiempo de respuesta y a la calidad de la solución si lo comparamos con un agente inteligente sin exaptación que siempre decide empezar sin información y en forma aleatoria ya que cada problema lo considera como uno nuevo.

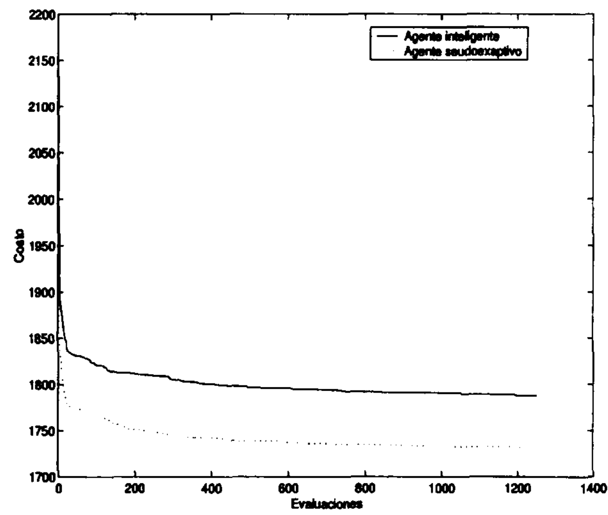


Figura 11.3: Desempeño promedio de ambos agentes en la reprogramación de tareas durante 100 días.