UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

U. A. C. P. y P.

"STA: Algoritmo Genético Estadístico basado en un Modelo Probabilístico de la Población"

T E S I S

QUE, PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACION,

P R E S E N T A :

ENRIQUE DIAZ INFANTE RENDON

ASESOR: DR. ANGEL KURI MORALES

México, D. F., **Febrero 2001**

Con mi eterno agradecimiento...

A mi familia: Papá, Mamá, Ale y Rossy
por toda su paciencia, amor y comprensión.

A Cynthia, a quien siempre llevaré en mi corazón,
por su apoyo y motivación.

A mi tutor: Dr. Angel Kuri M.
por el entusiasmo mostrado, su tiempo y dedicación.

A mis sinodales
por sus valiosos comentarios.

Tabla de Contenido

Tabla de Figuras

Resumen

En esta tesis se presenta un algoritmo de optimización que consta de dos fases. La primer fase (genética) obtiene una solución mediante un proceso evolutivo, en el cual se modelan las soluciones prometedoras con un vector de distribución de probabilidad y se utiliza dicho vector para generar nuevas propuestas y guiar la exploración del espacio de búsqueda. La segunda fase (escalador) modifica iteradamente en un bit la mejor solución obtenida hasta el momento. El proceso termina cuando no sea posible mejorar la solución al alterar un bit.

Este algoritmo genético simula de manera estadística el comportamiento de los operadores genéticos tradicionales: selección y cruza, evitando su uso y por ende, la definición heurística de sus parámetros. A pesar de que teóricamente no es necesario, se incluye en el sistema una versión adaptada del operador de mutación debido a los resultados experimentales.

El algoritmo estadístico es probado en un conjunto de funciones clásicas (de DeJong), dando resultados similares a los del Algoritmo Genético Simple con cruza uniforme.

El Algoritmo Genético Estadístico tiene como parámetros la probabilidad de mutación, el tamaño de la población y el tamaño de la élite. Mediante experimentos, se muestra la función que desarrolla cada uno de estos parámetros dentro del proceso evolutivo.

Una desventaja de los algoritmos genéticos tradicionales es el gran consumo de memoria que requieren en problemas con tamaños de población grande. Debido a que el vector de probabilidad es un modelo de la población, un cálculo anticipado de éste evita que el Algoritmo Genético Estadístico tenga que almacenar toda la población, lo que resulta en ahorros considerables de memoria.

Finalmente se señala que la manera de escalar los valores de evaluación, el método de actualización del vector de probabilidad y el modelo de probabilidad utilizado son direcciones en las cuales es posible profundizar la investigación en un futuro.

1 Introducción

Uno de los temas de mayor interés en la actualidad, es la *optimización*. La optimización se entiende como el proceso de búsqueda de la "mejor" solución de un problema específico, de acuerdo con ciertos criterios definidos previamente. Los resultados de este procedimiento, permiten la reducción del tiempo y costo de los procesos en general.

Entre los métodos de optimización que han tenido mayor auge en los últimos años se encuentran los Algoritmos Evolutivos (AE)[1], destacando los Algoritmos Genéticos (AG) en una gran variedad de aplicaciones. A pesar de su versatilidad, no pueden considerarse como sistemas automáticos de aplicación general como sería deseable, ya que requieren que la persona que los utilice tenga cierto conocimiento previo del problema.

Los algoritmos evolutivos fueron inspirados por la teoría de la evolución de Darwin y por el conocimiento sobre Genética. Debido a ello, realizan procesos por medio de operadores que emulan a aquellos encontrados en la naturaleza, para generar de manera heurística nuevas propuestas de solución al problema especificado. Los operadores, como se conocen tradicionalmente, definen ciertos parámetros que controlan su actuación dentro de la búsqueda. Los valores de los parámetros determinan en gran medida si al final del proceso se encontrará una solución adecuada y si ésta se encontrará eficientemente.

La definición correcta de los parámetros de operación del algoritmo no es una tarea sencilla. Por ello, se han creado diversas formas de establecer esos parámetros. Algunos autores argumentan que existen valores para algunos parámetros que hacen que el algoritmo sea robusto siempre y cuando se cumplan ciertas características (véase [HARI99a], [GOLD93]). Otros autores defienden la idea de que es inapropiado utilizar los mismos valores durante toda la búsqueda y por lo tanto, definen métodos que controlan los parámetros conforme avanza el algoritmo (p.ej. [BÄCK92] [HINT97a]). Algunas clasificaciones de los métodos de definición y control de los parámetros pueden encontrarse en [EIBE99], [ANGE95] y [HINT97b].

El paradigma de la genética ha provisto a los científicos de conocimientos muy valiosos que han sido determinantes para el buen funcionamiento de los algoritmos diseñados. Sin embargo, como todo paradigma define una visión determinada y oculta otras posibilidades.

Debido a mi formación de Ingeniería, tengo una visión pragmática respecto al diseño de los Algoritmos Evolutivos. No creo necesario apegarse rigurosamente a las ideas inspiradoras de los algoritmos. Por ello, pienso que es posible lograr el comportamiento evolutivo sin necesidad de utilizar los operadores genéticos. Y en consecuencia, reducir la necesidad de definir o controlar los parámetros relacionados con estos operadores.

[1] Este término es utilizado de manera general para referirse a los algoritmos que simulan evolución, ya sea en escala genética o en otros niveles de abstracción, a saber, Algoritmos Genéticos, Programación Evolutiva y Estrategias Evolutivas.

1.1 Motivación

A pesar de haberse presentado por primera vez hace más de veinte años, el área de los Algoritmos Evolutivos sigue siendo un campo fértil de investigación. De manera general, se distinguen cuatro rubros bajo los cuales pueden catalogarse la mayoría de los artículos publicados respecto al tema.

El primer rubro es el de las *aplicaciones* donde se muestran múltiples e ingeniosas formas de utilizar los Algoritmos Evolutivos (p.ej. [DEIT95], [PINT99]). Es de llamar la atención que estas aplicaciones conciernen no solamente a sectores industriales sino también a otras áreas de investigación científica.

La segunda categoría agrupa a las publicaciones que presentan métodos para definir o *controlar* los valores de los diversos *parámetros* involucrados en el proceso evolutivo (p.ej. [GREF86], [SCHA89]). Ésta es un área de gran interés, ya que los algoritmos tradicionales dependen en gran medida de la correcta definición o control en el tiempo de dichas variables para la obtención de buenos resultados.

En tercer término encontramos a los *desarrollos teóricos* del área. En ellos, se procura describir las bases formales que fundamentan el comportamiento y resultados de los algoritmos en problemas complejos[2] (p.ej. [MÜHL92], [VOSE96], [RUDO97]). A pesar de los grandes esfuerzos realizados, todavía no se conoce la naturaleza exacta del comportamiento de los algoritmos.

La cuarta categoría, en la cual se inscribe la presente tesis, se refiere a aquellos trabajos donde se muestran estudios que permiten *diseñar nuevos algoritmos* evolutivos que mejoren las características de los algoritmos anteriores. Debido a la dificultad asociada con el diseño, son pocos los artículos que podemos encontrar, en comparación con los relacionados con los demás rubros.

Dos sistemas basados en paradigmas distintos, sirvieron como inspiración a esta tesis: los Algoritmos Genéticos [GOLD89] y los Sistemas de Hormigas [DORI96] (véase Apéndice A). La comparación entre el proceso de aprendizaje de ambos sistemas hace creer que es posible diseñar un sistema evolutivo que resuelva rápida y eficientemente problemas de optimización, que además reduzca la cantidad necesaria de parámetros heurísticos.

1.2 Objetivo

Estudiar nuevos métodos de evolución mediante la creación de un algoritmo eficiente y efectivo que sirva como base para futuras investigaciones.

[2] La complejidad de un problema suele estar asociada con un espacio de búsqueda de grandes dimensiones y un espacio de soluciones reducido.

1.3 Organización del Documento

Hasta el momento se ha dicho que la optimización es una tarea de gran importancia, ya que consiste en buscar la mejor solución a un problema dado bajo ciertos criterios, lo cual permite la reducción del tiempo y costo de los procesos en general.

También se ha dicho que a pesar del gran auge que han tenido los algoritmos genéticos como sistemas de optimización, aun no pueden considerarse sistemas automáticos de aplicación general, ya que utilizan operadores que requieren parámetros definidos heurísticamente por el usuario.

Como antecedentes del diseño del algoritmo de estudio de esta tesis, se describirá el funcionamiento del Algoritmo Genético Simple. Mediante un ejemplo simple se explicará cada uno de los procesos que componen al algoritmo para finalizar la sección con el pseudo-código correspondiente.

En el siguiente capítulo se describe el Algoritmo Genético Estadístico, tanto en su fase evolutiva como en su fase del escalador. Con él se muestra una nueva técnica evolutiva basada en el modelo de la población a través de un vector de distribución de probabilidad.

En el capítulo 3 se muestran los experimentos realizados sobre un conjunto de problemas matemáticos bien conocidos en el área. A manera de análisis de la función que desempeñan los parámetros en el Algoritmo Genético Estadístico, en el capítulo 4 se muestran los resultados obtenidos al variar cada uno de los parámetros en un ejercicio determinado.

Dado que en la Inteligencia Artificial comúnmente se trata con problemas cuyo espacio de búsqueda rebasa las capacidades de almacenamiento de los equipos actuales, en el capítulo 5 se describen las modificaciones necesarias de la fase genética del algoritmo estadístico para reducir el espacio de memoria requerido en ejercicios con tamaños de población grande.

La definición del modelo probabilístico, como está planteado en el Algoritmo Genético Estadístico, impone algunas dificultades y limitantes. En el capítulo 6 se describen los problemas encontrados y se dan algunas guías para futuras investigaciones.

En el último capítulo se establecen las conclusiones resultantes del estudio de esta tesis.

1.4 Algoritmo Genético Simple

En 1975, John Holland [HOLL75] se inspiró en la genética y la selección natural para proponer un método computacional de búsqueda, conocido ahora como Algoritmo Genético Simple (SGA[3]).

La idea de Holland fue simular parcialmente el proceso evolutivo de las especies haciendo énfasis en aquellas funciones de la genética que se considera que tienen mayor influencia en la naturaleza; éstas son: la selección natural de los mejores individuos, el apareo o cruza de los individuos para la procreación y la mutación como mecanismo de adaptación de las especies.

Los Algoritmos Genéticos (AGs) han sido utilizados con éxito en distintos sectores industriales y de investigación para atacar problemas de optimización, aprendizaje de máquina y programación automática, entre otros. En esta tesis, nos enfocaremos a la faceta relacionada con la optimización y específicamente con la optimización de funciones numéricas.

[3] Debido a sus siglas en inglés: *Simple Genetic Algorithm.*

La propuesta original de Holland ha sufrido muchas modificaciones. En el libro "A Comprehensive Approach to Genetic Algorithms in Optimization and Learning" [KURI99a] se presenta una taxonomía en la cual se reconocen 2,624,400 formas de definir un AG. Además de que se menciona que esta taxonomía no es exhaustiva[4]. En múltiples ocasiones se ha demostrado que el SGA es comúnmente inferior que sus variantes (véase p.ej. [DAVI91]). A pesar de esto, se sigue tomando el algoritmo original como base y por ello es conocido también como Algoritmo Genético Canónico.

A continuación, veremos los pasos que se siguen al utilizar el SGA y analizaremos el proceso mediante un ejemplo ilustrativo.

Supongamos que se desea conocer el valor máximo de la siguiente función:

$$f(x,y) = x + 2y$$

dados los siguientes límites[5]:

$$-16 < x < 16$$
$$-16 < y < 16$$

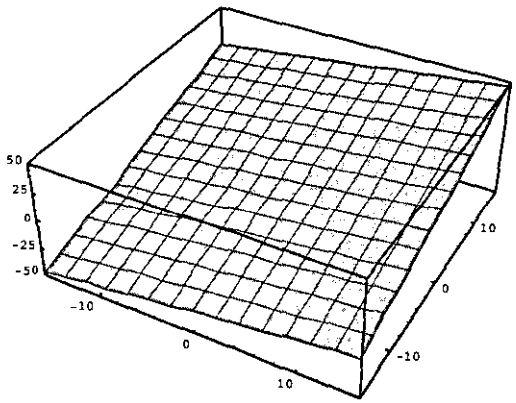


Figura 1: Gráfica de la Función-Ejemplo: f(x,y)=x+2y

A simple vista podemos observar que el valor máximo es cercano a 48 y se obtiene cuando ambas variables tienen valor cercano a 16.

El hecho de conocer la forma de la función y el resultado de antemano nos ayudará a concentrarnos en el proceso del Algoritmo Genético y no en el problema.

[4] No se incluyen ciertos operadores como la inversión [GOLD89], ni se consideran variantes del estilo propuesto en esta tesis.

[5] Estos límites fueron seleccionados para simplificar la representación que presentaré posteriormente.

1.4.1 Representación

El primer paso para la solución de un problema en cómputo consiste en encontrar una *representación* adecuada del mismo. Con ello, se transforma el dominio del problema en el de búsqueda.

Los AGs representan las soluciones factibles con una cadena de caracteres de un alfabeto determinado. En los problemas de optimización, el valor de cada variable se representa con una cadena. Estas cadenas se concatenan para formar una propuesta de solución. Comúnmente se utiliza el *alfabeto binario* (i.e. unos y ceros).

En el caso de querer representar valores numéricos, se puede optar entre diferentes códigos para interpretar la cadena. Los *códigos* más usados son el binario ponderado y el Gray (véase apéndice A). En este ejemplo utilizaremos el *binario ponderado*, ya que es el más simple y refleja las características del código que tradicionalmente conocemos: decimal ponderado.

El ejemplo actual necesita que podamos representar números reales, lo cual nos lleva a una nueva decisión: la notación. Podemos optar por notación de punto fijo o punto flotante. Debido al funcionamiento de los AGs, suele preferirse la *notación de punto fijo*. Todos los experimentos de esta tesis utilizan esta notación.

Dadas estas decisiones (alfabeto binario, código binario ponderado y notación de punto fijo), el siguiente paso consiste en determinar el número de caracteres o bits que se necesitarán para representar la variable, tanto en su parte entera como fraccionaria.

La cantidad de bits para la parte entera está determinada por el dominio de la función a optimizar. Debe poder representar todos los valores enteros del dominio, ajustándose al menor tamaño posible[6]. Observamos que el dominio de la función, en este ejemplo, está dado por el intervalo (-16,16) para ambas variables. Por ello, seleccionamos cuatro bits enteros para denotar valores entre cero y quince. Anexamos un bit extra para representar el signo; cero, para números positivos y uno, para números negativos[7].

La cantidad de bits fraccionarios dependerá de la cantidad de dígitos decimales de precisión que queramos en el resultado final. Para simplificar el ejercicio, supongamos que solamente nos interesa un dígito decimal. Por ello, seleccionamos cuatro bits fraccionarios para su representación.

Con estos datos, podemos formar una cadena de nueve caracteres de longitud que representa el valor de una variable. Por ejemplo, la cadena:

signo	parte entera				parte fraccionaria			
1	1	1	0	0	1	0	1	1

equivale al número decimal –12.6875, calculado de la siguiente manera:

$$-1^{1}*[(1*2^{3})+(1*2^{2})+(0*2^{1})+(0*2^{0})+(1*2^{-1})+(0*2^{-2})+(1*2^{-3})+(1*2^{-4})].$$

[6] Los algoritmos pueden funcionar aun cuando no se utilice el menor tamaño posible, sin embargo, esto puede provocar fallas en la eficiencia y efectividad del algoritmo.

[7] Nótese que esta representación nos permite tener un cero positivo y un cero negativo. Para fines prácticos, haremos caso omiso de este detalle.

Al unir dos cadenas de este tipo obtendremos una propuesta para los valores de las variables del problema. En términos de los AGs, esta cadena es un cromosoma que determina a un individuo.

1.4.2 Población Inicial

La primer etapa del AG consiste en crear un conjunto de individuos con la representación descrita anteriormente. A este conjunto de individuos se le conoce como población. Debido a que normalmente no contamos con mayor conocimiento que la descripción del problema, se elige el valor de cada bit al azar[6], teniendo la misma probabilidad de seleccionar un cero o un uno.

La decisión a tomar en este momento es el tamaño que se utilizará para este conjunto. El tamaño de la población es un factor determinante en la calidad del resultado final. Existen múltiples estudios respecto al tamaño ideal de la población (p.ej. [GOLD91], [DEBK92], [HARI97]). Sin embargo, no existe todavía una fórmula general. Es aún, tema de investigación. En este ejemplo se eligió arbitrariamente un tamaño de 10 individuos. En la siguiente tabla se muestra a los individuos generados.

	x (binario)								y (binario)									
no.	sgn	parte entera				parte fraccionaria				sgn	parte entera				parte fraccionaria				x	y
1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1	0	1	0	10.5625	0.6250
2	1	1	0	1	0	1	0	1	1	1	1	0	0	0	1	0	1	1	-10.6875	-8.6875
3	1	1	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	0	-9.2500	2.5000
4	0	1	1	1	0	0	0	0	0	1	0	0	1	0	0	1	1	1	14.0000	-2.4375
5	0	0	1	1	1	0	1	0	1	0	0	0	0	0	1	1	0	1	7.3125	0.8125
6	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	0.3750	-6.0000
7	0	0	1	1	0	0	1	0	1	1	0	1	1	0	1	0	1	1	6.3125	-6.6875
8	1	0	1	1	0	1	1	1	0	1	1	1	0	1	0	0	0	0	-6.8750	-13.0000
9	0	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	13.3125	10.1250
10	0	1	0	0	1	0	1	0	1	1	1	1	0	1	1	1	1	0	9.3125	-13.8750

Tabla 1: Primera Generación de Individuos del Problema-Ejemplo.

[6] Debido a su naturaleza, ninguna computadora es capaz de generar secuencias de números verdaderamente aleatorios (al azar). Por ello, se utilizan secuencias pseudo-aleatorias que cumplen con ciertas propiedades y que a los humanos nos parecen ser aleatorios. Para fines prácticos, estas secuencias son suficientes. En esta tesis, no haré diferencia al respecto. Si el lector desea conocer más sobre el tema, puede consultar [KNUT69].

1.4.3 Evaluación

El siguiente paso del Algoritmo Genético consiste en definir una función que dada una propuesta de solución al problema (i.e. un individuo), indique qué tan buena o mala es ésta con respecto a los demás individuos de la población[9]. En casos como éste, de maximización de funciones numéricas, la decisión es simple: se utiliza la función a optimizar como función de evaluación. Puede optarse, también, por una función con un comportamiento similar. Por ejemplo, para optimizar la función f(x)=5x+2 en un rango de valores positivos puede utilizarse simplemente $g(x) = x$, ya que:

$$\forall(x1,x2)\,(f(x1) = f(x2) \Leftrightarrow g(x1) = g(x2)) \text{y } (f(x1) > f(x2) \Leftrightarrow g(x1) > g(x2)).$$

La selección de la función de evaluación (o en inglés *fitness*) determina la velocidad con la cual se encontrará una solución final. En el esquema del SGA, es conveniente que las mejores soluciones (i.e. individuos mejor adaptados) tengan un valor de evaluación mayor. Es decir, en el caso de querer minimizar una función conviene utilizar como función de evaluación la función de optimización multiplicada por menos uno.

En este ejemplo, ambas funciones son equivalentes. A continuación se presentan los resultados de la evaluación de cada individuo.

no.	x	y	evaluación (e)
1	10.5625	0.6250	11.8125
2	-10.6875	-8.6875	-28.0625
3	-9.2500	2.5000	-4.2500
4	14.0000	-2.4375	9.1250
5	7.3125	0.8125	8.9375
6	0.3750	-6.0000	-11.6250
7	6.3125	-6.6875	-7.0625
8	-6.8750	-13.0000	-32.8750
9	13.3125	10.1250	33.5625
10	9.3125	-13.8750	-18.4375

Tabla 2: Valores de Evaluación de los Individuos en el Problema-Ejemplo.

1.4.4 Evolución

Un Algoritmo Genético procura explotar la información existente en una población para generar nuevas propuestas de solución con la esperanza de que éstas sean, en promedio, mejores que las de la generación[10] anterior. Como podemos darnos cuenta, se trata de un proceso repetido de generación y explotación de datos.

La manera en la cual se explota la información en el SGA es empírica e implícita. Se utilizan tres operadores que pretenden simular a los principales procesos de la genética: selección, cruza y mutación. De manera general, se eligen dos individuos de la población (o cromosomas en términos genéticos), se combinan sus características y se modifican ligeramente, produciéndose

[9] Nótese que, aunque la mayoría de las veces se utiliza una función global que diferencia la calidad de la solución con respecto a todas las posibles soluciones, es suficiente con una función relativa a la población actual.

[10] En esta tesis utilizaré indistintamente los términos "población" y "generación" para referirme al mismo concepto.

dos nuevos individuos. Este proceso se repite tantas veces como sea necesario para alcanzar el tamaño de la población.

A continuación explicaré cada uno de los operadores de evolución, tal como se utilizan en el SGA.

1.4.4.1 Selección

Existen diversos esquemas con los cuales se "aparean" los individuos. Se reconocen dos grandes categorías: determinísticos y probabilísticos. Los esquemas determinísticos definen de antemano cuales propuestas se combinarán, de acuerdo con los resultados obtenidos en la evaluación. Los esquemas probabilísticos definen para cada individuo un valor de probabilidad asociado con su evaluación y utilizan estos valores en un ámbito de competencia. Para mayor información respecto a los diferentes esquemas de selección, véase [KURI99a, pp.16-20]

El SGA utiliza un esquema probabilístico conocido como Selección Proporcional, coloquialmente Selección de la Ruleta. El método consiste en encontrar una distribución de probabilidad (i.e. valores en el intervalo (0,1) tal que su suma es igual a uno) relacionada con la población. Se genera un valor aleatorio[11] en (0,1] y se compara con la distribución. El individuo cuyo rango de probabilidad cubra el valor aleatorio, es seleccionado.

La distribución de probabilidad se obtiene al normalizar los valores de la función de evaluación en el rango (0,1). Primeramente, se alteran los valores, de tal forma que se obtengan exclusivamente números positivos. Una manera de hacerlo sumar a cada uno, la suma del valor absoluto del menor valor más la media de todos los valores. Es decir,

$$e'_i = e_i + \left|\min(e_i)\right| + \frac{1}{N}\left(\Sigma\left|e_i\right|\right)$$

Es importante cuidar que el valor agregado no reduzca demasiado la diferencia proporcional entre los valores.

Posteriormente, cada uno de estos nuevos valores es dividido entre la suma de ellos. Obteniendo finalmente el valor normalizado.

$$\Phi_i = \frac{e'_i}{\Sigma e'_i}$$

Con este procedimiento obtenemos los resultados mostrados[12] en la Tabla 3. La constante sumada para obtener e' es 49.45. La suma de los valores e' es 455.625. La última columna indica el límite superior del rango de probabilidad cubierto, obtenido de la suma acumulada de los valores Φ de los individuos anteriores.

[11] Se asume que todos los números aleatorios utilizados están distribuidos de manera uniforme (i.e. existe la misma probabilidad de elegir cualquier valor en el intervalo).

[12] Los resultados de la evaluación aparecen en las tablas con cuatro o más dígitos decimales, de los cuales solamente el primero es significativo. Esto de acuerdo con la representación elegida en un principio.

no.	e	e'	Φ	rango (r)
1	11.8125	61.2625	0.13446	0.13446
2	-28.0625	21.3875	0.04694	0.18140
3	-4.2500	45.2000	0.09920	0.28060
4	9.1250	58.5750	0.12856	0.40916
5	8.9375	58.3875	0.12815	0.53731
6	-11.6250	37.8250	0.08302	0.62033
7	-7.0625	42.3875	0.09303	0.71336
8	-32.8750	16.5750	0.03638	0.74974
9	33.5625	83.0125	0.18219	0.93193
10	-18.4375	31.0125	0.06807	1

Tabla 3: Valores de Normalización y Rango de Probabilidad.

Supongamos que obtenemos al azar el número a = 0.557260. Buscamos en la tabla aquel individuo cuyo límite superior de rango sea el mínimo mayor o igual que a. En este caso se trata de r_6= 0.62033. Por lo tanto, el sexto individuo es seleccionado.

Nótese que dada esta distribución, aquellas propuestas con mejor evaluación tienen mayor probabilidad de ser seleccionados. En términos genéticos, se dice que aquellos individuos mejor adaptados tienen mayor probabilidad de supervivencia. Esta propiedad permite que las características (o los genes) de las mejores soluciones permanezcan en la siguiente generación y por lo tanto, se evolucione. En otras palabras, la selección guía la búsqueda procurando mantener las propiedades que conforman una "buena" solución.

En el ejemplo que consideramos, se desea mantener el mismo número de individuos a través de las generaciones. Por tanto, el número de cadenas elegidas deberá ser el mínimo número par mayor o igual que el número total de cadenas existentes; en este caso, diez. En la siguiente tabla se enlistan las cadenas seleccionadas y el valor aleatorio a por el cual se eligieron. En términos estadísticos, se dice que se trata de una selección proporcional con reemplazo. Nótese que los individuos con mejor evaluación fueron seleccionados en más ocasiones, tal como se esperaba.

no.	valor aleatorio (a)
4	0.32432
5	0.41981
7	0.67256
9	0.84555
5	0.49197
4	0.35436
10	0.95205
7	0.62685
1	0.01097
3	0.22208

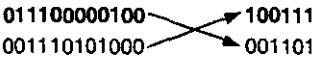
Tabla 4: Selección de Individuos de acuerdo con su rango de probabilidad.

1.4.4.2 Cruza

El siguiente paso consiste en tomar de dos en dos las cadenas seleccionadas (la primera con la segunda, la tercera con la cuarta, etc.) y se procede a combinar sus genes. Nuevamente, existen diversos métodos para realizar esta combinación. La cruza utilizada en SGA es conocida como cruza de un punto. Consiste en seleccionar aleatoriamente un número entero c en el intervalo $(0, l)$, donde l es la longitud de la cadena que representa a la solución. En este caso: l = 18. Al valor c se le conoce como punto de cruza.

Se forman dos nuevas cadenas combinando todos los caracteres de una cadena cuya posición sea menor o igual que *c* y con aquellos caracteres de la otra cadena cuya posición sea mayor que *c*. Las nuevas cadenas forman parte de una nueva población. La generación anterior es reemplazada por los nuevos individuos.

Si los individuos seleccionados son el cuarto y el quinto, y $c = 12$, obtendremos las propuestas mostradas en la siguiente figura:



a) Padres, antes de la cruza

011100000100 001101
001110101000 **100111**

b) Hijos, después de la cruza

Figura 2: Cruza de un solo punto de dos cadenas genéticas.

Un dato interesante de este mecanismo, es que no utiliza ninguna información respecto a la interpretación de los bits. No se distinguen a las variables ni a sus partes. Se considera que a pesar de ello, se pueden crear individuos mejor adaptados.

La siguiente tabla muestra el resultado de la cruza de los individuos seleccionados previamente.

No.	Nuevos Individuos	Padres	Pto.Cruza
11	011100000100001101	4,5	10
12	001110101000100111	4,5	10
13	001010101010100010	7,9	3
14	011100101101101011	7,9	3
15	001110000100100111	5,4	6
16	011100101000001101	5,4	6
17	011100101101101011	10,7	2
18	000010101111011110	10,7	2
19	010101001000001000	1,3	13
20	110010100000101010	1,3	13

Tabla 5: Segunda Generación, después de la cruza.

1.4.4.3 Mutación

Debido a que el operador de cruza procura mantener las características de los individuos más destacados, se corre el riesgo de sesgar demasiado de búsqueda. Por ello, es necesario un operador que permita explorar nuevas regiones del espacio de búsqueda. El balance correcto entre estas dos fuerzas determina la calidad del resultado final y la rapidez con la cual se alcanza.

En el SGA, el operador utilizado como inyector de variabilidad es la mutación. Consiste en modificar el valor de los bits de la población dada cierta probabilidad. Nuevamente, el lector podrá encontrar en la literatura diversas formas de realizar la mutación (p.ej [MITC96]). En el estudio de esta tesis se considerará exclusivamente la mutación uniforme.

En la mutación uniforme, todos los bits, no importando su posición o alguna otra propiedad, tienen la misma probabilidad de mutación (P_m). La manera más simple, aunque no la más eficiente, de implementar este operador genera un número al azar en [0,1) para cada bit y compara dicho número contra P_m, si es menor entonces se modifica el valor del bit, de lo contrario permanece igual. Esto es:

$$\forall b \in G, b = \begin{cases} \neg b & \text{si}(\text{aleatorio}(0,1) < P_m) \\ b & \text{de lo contrario} \end{cases}$$

donde b es un bit cualquiera y G es la población o generación.

En la siguiente tabla se muestran los bits mutados con negritas e itálicas, habiendo utilizado una probabilidad de mutación de 0.003. Este valor fue seleccionado arbitrariamente. Se pueden observar también en la tabla el resultado de la evaluación de las cadenas genéticas.

No.	Individuos	x	y	e
11	011100000100001101	14.0000	-0.8125	12.3750
12	001110101000100111	7.3125	2.4375	12.1875
13	001010101010100010	5.3125	10.1250	25.5625
14	011100101100101011	14.3125	-2.6875	8.9375
15	001110000100100111	7.0000	-2.4375	2.1250
16	001100101000001101	6.3125	0.8125	7.9375
17	011100101101001001	14.3125	-4.5625	5.1875
18	000010101111011110	1.3125	-13.8750	-26.4375
19	010100001000001000	10.0625	0.5000	11.0625
20	110010100000101010	-9.2500	2.6250	-4.0000

Tabla 6: Segunda Generación, después de la Mutación.

Al finalizar la mutación, obtenemos una población que en promedio mejora a la población anterior. En la siguiente tabla podemos ver que la nueva generación tiene una media mayor y una menor desviación estándar. Otra observación interesante es que el valor máximo de la segunda generación no superó a aquel de la primera. Este es un comportamiento normal del SGA. Recuerde que en este algoritmo, el comportamiento promedio es más importante que los valores máximos. Existen algunas variantes que conservan los mejores valores de una generación a otra. Se dice que estas variantes utilizan elitismo [KURI99a]. El elitismo puede ser parcial o completo dependiendo de la cantidad de individuos que perduren.

Generac.	Mínimo	Máximo	Media (μ)	Desv.Est.(σ)
1	-32.8750	33.5625	-3.88750	20.2465
2	-26.4375	25.5625	5.49375	13.6012

Tabla 7:Valores estadísticos de las dos primeras generaciones.

1.4.5 Terminación

Como se dijo anteriormente, un Algoritmo Genético es un proceso repetido de generación y explotación de datos. En las secciones anteriores vimos cómo explotar la información contenida en una población para generar una nueva. Surge ahora la pregunta: ¿cuántas veces debe repetirse este proceso?

El proceso se repite hasta que se cumpla alguna condición definida por el usuario. Algunas de las condiciones más comunes se enlistan a continuación.

- Convergencia: Esta condición se presenta cuando el algoritmo no es capaz de generar nuevos individuos por medio de la cruza. Lo anterior ocurre cuando son idénticas todas las cadenas que representan a la población. Al ocurrir esta condición, la capacidad de exploración se reduce exclusivamente a lo que el operador de mutación pueda lograr[13]. La probabilidad de mutación en el SGA es estática (i.e. no varía conforme avanza el algoritmo) y normalmente pequeña[14]. Esto provoca que la probabilidad de encontrar nuevos patrones sea muy reducida y se requiera gran esfuerzo para lograrlo. Por ello, en términos generales se considera que una vez habiendo convergido el algoritmo, no tiene sentido continuar la búsqueda.

- Costo Máximo de Proceso: La etapa más costosa, en términos de proceso computacional, de un AG es la evaluación. En algunos caso es posible que se requiera de un procedimiento muy complejo para obtener el valor de *fitness*. Debido a esto, el usuario puede optar por definir previamente un costo máximo de proceso para el algoritmo. Este costo puede establecerse por medio de un número máximo de generaciones o bien, un número máximo de evaluaciones. El algoritmo termina al alcanzarse este máximo, si antes no ocurre alguna otra condición de terminación.

- Resultado Satisfactorio: Otra condición suficiente para finalizar el proceso se da cuando alguna de las soluciones propuestas en la población actual es satisfactoria para los fines para los cuales se haya aplicado el algoritmo. La medida de lo satisfactorio se define para cada problema en particular. Por ejemplo, si queremos conocer las medidas que maximizan el volumen de una caja de cartón dada cierta cantidad de material, puede ser suficiente con alcanzar un volumen determinado aún cuando éste no sea el máximo.

En el problema-ejemplo se utilizó la convergencia como condición de terminación. Ésta se alcanzó en la generación ciento catorce. A continuación se presentan los resultados obtenidos en una ejecución del algoritmo.

1.4.6 Resultados

El algoritmo genético, utilizado como lo acabamos de presentar, tiene un comportamiento general como el que se muestra en la Figura 3. Conforme pasan las generaciones, el valor promedio de la evaluación de los individuos tiene incrementos cada vez menores y la desviación estándar tiene ciclos en los cuales su valor se va reduciendo paulatinamente. Dada la naturaleza aleatoria del algoritmo, en ocasiones se encuentran cadenas cuya evaluación produce saltos en la desviación, valores grandes que poco a poco se reducen.

Lo anterior nos indica que, efectivamente, el algoritmo encuentra implícitamente patrones que mejoran la evaluación de los individuos (véase Gráfica A de la Figura 3). También podemos observar que, al repetir con mayor frecuencia dichos patrones, las diferencias entre las propuestas de solución se reducen (véase Gráfica B de la Figura 3) y esto nos lleva eventualmente a la convergencia (i.e. cuando todos los individuos de la población son idénticos).

Entre los investigadores de computación, a los patrones se les conoce con el nombre de esquemas.

---

[13] Existen métodos, conocidos como escaladores (o en inglés *Hill-Climbers*), que basan su comportamiento únicamente en la mutación. Serán estudiados posteriormente, al referirnos al Algoritmo Genético Estadístico.

[14] Si la probabilidad de mutación no fuera pequeña, desviaría demasiado la búsqueda y se perderían los patrones de los mejores individuos encontrados con la cruza.

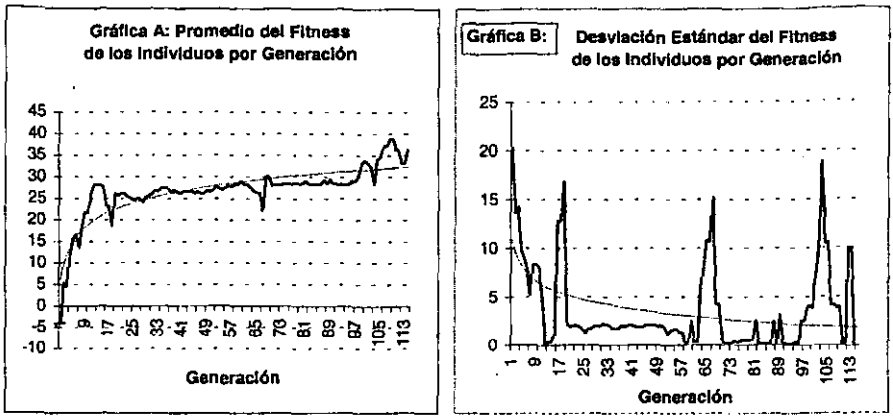


Figura 3: Gráficas del promedio y desviación estándar del fitness por generación del Algoritmo Genético Simple.

En la siguiente tabla se presentan los resultados finales obtenidos después de una ejecución del algoritmo. Recuerde que se trata de un sistema en el que interviene en buen medida el azar, por lo cual los resultados de diferentes ejecuciones del algoritmo para el mismo problema pueden variar.

	Generac.	Individuo	x	y	e
Convergencia	114	0111111010100100101	15.8125	10.3125	36.4375
Máximo Global	106	0111111010111001011	15.8125	14.3125	44.4375
Máximo Teórico	-	0111111110111111111	15.9375	15.9375	47.8125

Tabla 8: Resultados alcanzados y máximo teórico del Problema-Ejemplo.

En este caso, no se obtuvo el mejor resultado posible mostrado en el tercer renglón de la Tabla 8. Esto puede deberse a la selección arbitraria que se hizo de los parámetros del sistema. A pesar de ello, existe una diferencia muy pequeña entre el máximo encontrado y el valor teórico. Solamente varían cuatro bits, lo que equivale a un error aproximado del veintidós por ciento. Nótese, también que de acuerdo con la representación definida los bits errados se encuentran en posiciones de bajo peso para las variables del problema.

1.4.7 Pseudo-código

Habiendo estudiado con ejemplo sencillo el comportamiento general de un algoritmo genético, estamos en posición de definir un pseudo-código. Recuérdese que se trata de una versión simplificada, siendo ésta suficiente para efectos de esta tesis. La siguiente figura describe el pseudo-código.

INICIALIZACIÓN

Sea P la población actual, N el número de individuos en la población, p_m la probabilidad de mutación, l la longitud de la cadena genética. Sea f la función de evaluación e i el número de generación actual

POBLACIÓN INICIAL

1. Sea i = 0
2. **Genere** N cadenas binarias de longitud l, eligiendo cada bit con la misma probabilidad (i.e. 0.5) de ser cero o uno.

EVALUACIÓN

3. Para cada individuo de la población actual P.
   3.1. Decodifique el valor propuesto de las variables.
   3.2. **Evalúe** al individuo de acuerdo con sus variables y la función f.

CONDICIÓN DE TERMINACIÓN

4. Verifique si se ha cumplido alguna condición de **terminación.** En caso afirmativo, vaya al paso 10.

EVOLUCIÓN

5. **Seleccione** con reemplazo N^1 cadenas, teniendo mayor probabilidad de ser elegidas aquellas con mejor evaluación.
6. Para cada par de cadenas:
   6.1. Genere un número c en $[1,l]$
   6.2. **Cruce** los primeros c bits de ambos individuos.
   6.3. Agregue estos dos individuos[2] a un nuevo conjunto Q.
7. Con probabilidad P_m **mute** el valor de los bits de Q.
8. Reemplace P con los individuos de Q.
9. Vaya al paso 3.

RESULTADO

10. Muestre el resultado.
11. Termine.

[1] N+1 si N es impar.
[2] Solamente uno si N es impar y es la última pareja.

Figura 4: Pseudo-código de un Algoritmo Genético Simple.

2 Algoritmo Genético Estadístico

El Algoritmo Genético Estadístico (STA[15]), que propongo en esta tesis, representa un esfuerzo por diseñar un algoritmo evolutivo eficiente y efectivo de aplicación general para resolver problemas de optimización. A la vez, se presenta como una plataforma para el estudio de una técnica nueva de evolución que consiste en el uso de un modelo probabilístico de la población para guiar la búsqueda.

El STA imita, de manera estadística, el comportamiento de los operadores genéticos tradicionales (i.e. selección, cruza y mutación) evitando su uso y su consecuente definición de parámetros que en muchas ocasiones resulta tediosa y heurística. Este algoritmo trabaja en dos fases: una genética – encargada del proceso de búsqueda propiamente dicho y una fase del escalador – encargada de refinar la mejor solución alcanzada en la etapa anterior.

En los experimentos realizados sobre un conjunto de problemas de optimización numérica y diversas variantes del algoritmo se obtuvieron resultados satisfactorios en comparación con el SGA tanto en la calidad de los resultados como en cuanto al tiempo de proceso y uso de memoria.

En la siguiente sección se mostrará la fase evolutiva del sistema. Se describirá su comportamiento y se presentará el pseudo-código asociado. En una sección posterior, se mostrará la fase del escalador y las razones por las cuales ésta es necesaria.

2.1 Fase Genética

La fase genética del Algoritmo Genético Estadístico, como cualquier algoritmo evolutivo, cuenta con un ciclo de generación-explotación de la información. La diferencia de este método con los tradicionales radica en la manera en la cual se generan nuevas propuestas de solución al problema.

En un algoritmo genético tradicional, la población contiene información referente al espacio de búsqueda que ha sido explorado previamente. Los operadores de selección y cruza se utilizan para explotar esta información, generando nuevas soluciones (o individuos) potencialmente mejores. La selección permite guiar al sistema en la búsqueda, replicando mayor cantidad de veces aquellos patrones (o esquemas) que forman una buena solución. La cruza combina las características encontradas para explorar diferentes zonas del espacio de búsqueda.

En el STA, la información obtenida de la población es modelada mediante un vector de probabilidad[16]. Este vector tiene tantas variables como genes (bits) existan en un individuo. Cada variable es una estimación de la probabilidad de que el bit correspondiente a la posición de la variable sea uno en la solución óptima. Dado que los valores se encuentran entre cero y uno, podemos ver al vector de probabilidad como un genoma difuso. En realidad, estamos definiendo un conjunto de funciones de distribución de probabilidad, una por cada bit en el genoma (véase la nota 16 al pie).

---

[15] Por sus siglas en inglés: *Statistical Genetic Algorithm* [KURI99b].

[16] El término "vector de probabilidad" se refiere a vectores que contienen valores positivos y cuya suma es igual a uno. La estructura que se utiliza en el algoritmo STA es en realidad una matriz con tantas columnas como bits en el genoma y tantos renglones como caracteres en el código utilizado. Cada columna de esta matriz es un vector de probabilidad. Debido a que en toda la tesis solamente se utiliza código binario, la matriz puede simplificarse tomando solamente la probabilidad de ser uno, ya que la otra es complementaria P(0)=1-P(1). Por ello, se abusará del término vector de probabilidad para referirse a esta matriz simplificada. Nótese que los términos del vector renglón de la matriz simplificada no suman uno.

El uso del vector de probabilidad reemplaza a la selección y la cruza en el proceso de creación de nuevas soluciones. Para cada bit *k* de un nuevo individuo, se genera un número aleatorio ρ_k entre cero y uno a partir de una distribución uniforme, si éste es menor que el *k*-ésimo elemento del vector de probabilidad, entonces el bit *k* del nuevo individuo será igual a uno o de lo contrario será cero. En la siguiente figura se muestra un ejemplo de la creación de un individuo.

	1	2	3	4	5	6	7
Vector de probabilidad	0.3590	0.8694	0.5000	0.1461	0.2774	0.6613	0.7823
ρ_k	0.3924	0.2595	0.5213	0.5868	0.9739	0.0975	0.7092
Nuevo Individuo	0	1	0	0	0	1	1

Tabla 9: Ejemplo de la creación de un individuo basado en un vector de probabilidad.

En la primera generación, se inicializan las variables del vector de probabilidad en 0.5. Esto produce el mismo efecto que en el SGA al crear la población inicial. En las generaciones subsecuentes, el vector de probabilidad *P* se calcula a partir del valor normalizado Φ de la evaluación de los individuos de la población (véase el tema Selección, pág.11), de acuerdo con la siguiente fórmula:

$$P_k = \sum_{j=1}^{n} \Phi_j b_{jk} \qquad k = 1..l$$

donde *l* es la longitud del genoma, *n* es el número de individuos en la población y b_{jk} denota el *k*-ésimo bit del individuo *j*.

Si consideramos a la población utilizada en el ejemplo del SGA (Tabla 1) y su evaluación (Tabla 3), podemos calcular el vector de probabilidad que la modela como se muestra en la siguiente tabla[17], en la cual se multiplicó cada bit por el valor Φ del individuo. La suma de estos valores da como resultado al vector de probabilidad, registrado en el último renglón de la tabla.

x									y								
sgn	parte entera				parte fraccionaria				sgn	parte entera				parte fraccionaria			
000	0.134	0.000	0.134	0.000	0.134	0.000	0.000	0.134	0.000	0.000	0.000	0.000	0.000	0.134	0.000	0.134	0.000
047	0.047	0.000	0.047	0.000	0.047	0.000	0.047	0.047	0.047	0.047	0.000	0.000	0.000	0.047	0.000	0.047	0.047
099	0.099	0.000	0.000	0.099	0.000	0.099	0.000	0.000	0.000	0.000	0.000	0.099	0.000	0.099	0.000	0.000	0.000
000	0.129	0.129	0.129	0.000	0.000	0.000	0.000	0.000	0.129	0.000	0.000	0.129	0.000	0.000	0.129	0.129	0.129
000	0.000	0.128	0.128	0.128	0.000	0.128	0.000	0.128	0.000	0.000	0.000	0.000	0.000	0.128	0.128	0.000	0.128
000	0.000	0.000	0.000	0.000	0.000	0.083	0.083	0.000	0.083	0.000	0.083	0.083	0.000	0.000	0.000	0.000	0.000
000	0.000	0.093	0.093	0.000	0.000	0.093	0.000	0.093	0.093	0.000	0.093	0.093	0.000	0.093	0.000	0.093	0.093
036	0.000	0.036	0.036	0.000	0.036	0.036	0.036	0.000	0.036	0.036	0.036	0.000	0.036	0.000	0.000	0.000	0.000
000	0.182	0.182	0.000	0.182	0.000	0.182	0.000	0.182	0.000	0.182	0.000	0.182	0.000	0.000	0.000	0.182	0.000
000	0.068	0.000	0.000	0.068	0.000	0.068	0.000	0.068	0.068	0.068	0.068	0.000	0.068	0.068	0.068	0.068	0.000
83	**0.659**	**0.568**	**0.568**	**0.478**	**0.218**	**0.690**	**0.166**	**0.653**	**0.456**	**0.334**	**0.281**	**0.586**	**0.104**	**0.570**	**0.325**	**0.653**	**0.397**

Tabla 10: Cálculo del vector de probabilidad correspondiente a la población del Problema-Ejemplo de la sección 1.4

El operador de mutación no es necesario teóricamente, ya que las variaciones estadísticas envueltas en el proceso proveen la capacidad de explorar zonas desconocidas del espacio de búsqueda y evitar la convergencia prematura. Sin embargo, en la práctica se observó que un

[17] Debido a razones de espacio en la hoja, los valores fueron redondeados a tres dígitos.

factor pequeño de mutación es útil en el proceso. En este esquema, la mutación es un proceso estadístico pues depende no sólo de la probabilidad de mutación sino también de la probabilidad del gen denotada en el vector de probabilidad. Por lo mismo, típicamente el valor de la probabilidad de mutación es menor que el que se utilizaría en el SGA.

Los procesos de evaluación y normalización pueden ser iguales a los que se definieron en la sección 1.4. Las condiciones de terminación definidas para los algoritmos genéticos siguen siendo válidas. La convergencia puede verificarse ahora también en el vector de probabilidad. Esta condición ocurre cuando el vector deja de ser difuso, es decir, cuando todos los valores del vector de probabilidad son cero o uno.

En la práctica, no es frecuente que en el Algoritmo Genético Estadístico ocurra una convergencia completa debido a las fluctuaciones estadísticas que mencionamos anteriormente y a limitantes en la expresividad del vector de probabilidad (véase Sección 6.3). Sin embargo, es posible definir una medida de convergencia dada por el porcentaje de genes en el vector de probabilidad cuyo valor está definido en cero o uno.

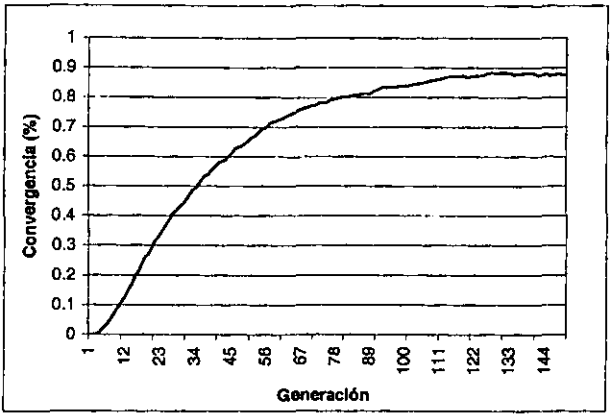


Figura 5: Gráfica del comportamiento típico del porcentaje de convergencia en el STA.

Típicamente la curva de convergencia tiene crecimientos con incrementos cada vez menores. El porcentaje de convergencia está directamente relacionado con el *grado de aprendizaje* del sistema. Esta relación explica el comportamiento de la curva. Inicialmente existe una etapa de exploración en la cual es relativamente simple discriminar los patrones útiles, situación que se refleja en un crecimiento inicial acelerado en la curva de convergencia. Conforme transcurre el tiempo, los patrones entre los cuales se debe discernir tienen mayor similitud. Lo anterior produce una desaceleración en la tasa de aprendizaje y consecuentemente un crecimiento menor en el porcentaje de convergencia.

Haciendo un análisis de la gráfica de convergencia, nos damos cuenta que existe un punto en la ejecución del algoritmo a partir del cual el esfuerzo requerido para avanzar en el aprendizaje es considerablemente mayor en comparación con las primeras generaciones produciendo un desperdicio enorme de recursos computacionales. Conviene, entonces, un cambio en la estrategia de optimización – la fase del escalador.

El punto exacto para el cambio de estrategia puede determinarse de diversas formas. Por ejemplo, puede definirse un costo máximo de proceso en relación con el número de generaciones, de evaluaciones o cualquier otra medida. Puede también analizarse el tiempo transcurrido a partir de la última mejora en el resultado hasta el momento actual y terminarse el algoritmo si no existen cambios o éstos son mínimos en un periodo determinado.

Se encontró que las medidas que determinan inicialmente un costo de proceso requieren del conocimiento previo del problema que tenga el operador. Su definición depende de la complejidad de cada problema. Aquellas que analizan los resultados no son plenamente confiables ya que se ven afectadas por las variaciones estadísticas propias del método de evolución.

Por lo anterior, se decidió utilizar la curva de convergencia para determinar el momento del cambio de estrategia. El análisis del porcentaje de convergencia está directamente relacionado con la tasa de aprendizaje, no requiere ninguna heurística particular a cada problema y el costo computacional asociado es muy bajo. Entonces, la fase genética del STA finalizará cuando la pendiente de la curva de convergencia alcance un valor cercano a cero.

Contamos ahora con un algoritmo que utiliza un vector como modelo de la población para generar nuevas propuestas y actualiza dicho vector al finalizar la creación de cada generación. En la siguiente figura se muestra el pseudo-código correspondiente.

INICIALIZACIÓN

Sea Π la población actual, N el número de individuos i en la población, m la probabilidad de mutación, l la longitud de la cadena genética. Sea f la función de evaluación, g el número de generación actual y P el vector de probabilidad

1. Sea g = 0 y $P_k = 0.5$ para toda k tal que $1 <= k <= l$

POBLACIÓN INICIAL

2. **Genere** N cadenas binarias Π_i de longitud l, eligiendo cada bit k con probabilidad P_k (i.e. 0.5) de ser cero o uno.

EVALUACIÓN

3. Para cada individuo i de la población actual Π.
   3.1. **Decodifique** el valor propuesto de las variables.
   3.2. **Evalúe** al individuo de acuerdo con sus variables y la función f. Llame a este valor de evaluación e.

CONDICIÓN DE TERMINACIÓN

4. Verifique si se ha cumplido alguna condición de **terminación**. En caso afirmativo, vaya al paso 11.

EVOLUCIÓN

5. **Desplace** los valores e de evaluación de las N cadenas a un rango de números positivos, según la siguiente fórmula:

$$e'_i = e_i - mín(e_i) \qquad \text{para maximizar o}$$

$$e'_i = máx(e_i) - e_i \qquad \text{para minimizar}$$

6. **Normalice** los valores obtenidos entre cero y uno: $\Phi_i = \frac{e'_i}{\sum e'_i}$

CONTINUA...

CONTINUACIÓN

7. **Actualice** los valores del vector de probabilidad de acuerdo con la siguiente fórmula:

$$P_k = \sum_{j=1}^{n} \Phi_j b_{jk} \qquad k = 1..l$$

donde l es la longitud del genoma, n es el número de individuos en la población y b_{jk} denota el k-ésimo bit del individuo j.

8. Repita N veces:
   - 8.1. Para cada j desde 1 hasta l:
     - 8.1.1. Genere dos valores aleatorios $0 <= \rho_j, \rho_k < 1$de una distribución uniforme
     - 8.1.2. Si $\rho_j > m$:

       $b_j = 1$ si $\rho_k <= P_j$, 0 de otro modo

       sino **mute** estadísticamente:

       $b_j = 0$ si $\rho_k <= P_j$, 1 de otro modo
   - 8.2. Agregue un individuo compuesto por la concatenación de los l bits generados en el paso anterior a una nueva población Γ
9. Reemplace Π con los individuos de Γ
10. Incremente el número g de generación actual. Vaya al paso 3.

RESULTADO

11. Almacene el resultado.
12. Continúe con la fase del escalador.

Figura 6: Pseudo-código de la fase genética del Algoritmo Genético Estadístico.

2.2 Fase del Escalador

La fase estadística del STA puede considerarse como un algoritmo completo, ya que adecuando correctamente sus parámetros puede dar respuestas efectivas. Sin embargo, su eficiencia va en decremento conforme avanza el algoritmo, dada su naturaleza estadística. En un principio, las cadenas son generadas de manera aleatoria. Por ello, es común que exista una varianza (σ^2) muy grande entre los valores de evaluación de estas cadenas. El algoritmo de la fase estadística explota eficientemente estas diferencias para guiar la búsqueda. Al hacerlo, provoca que las nuevas generaciones tengan varianzas cada vez menores hasta el punto en que ésta sea muy cercana a cero ($\sigma^2 \approx 0$) implicando convergencia. Una varianza menor indica que existen menos diferencias estadísticas entre los valores y por ende, menor información que explotar.

Debido a lo anterior y con objeto de mejorar la eficiencia del algoritmo global, se decidió finalizar prematuramente la fase estadística y ejecutar un algoritmo escalador sobre la mejor respuesta encontrada para llevarla al punto más alto (i.e. óptimo local o posiblemente global).

Existen diversos tipos de escaladores [MITC96] (conocidos en inglés como *Hill-climbers*) que son una variación evolutiva especial. En estos sistemas, se modifica sistemáticamente una cadena genética y se compara su evaluación contra la del genoma sin modificar continuando el mismo proceso con la mejor de ellas hasta que no se puedan encontrar mejoras. Un ciclo del escalador termina en este momento, considerado como el punto en el que cualquier modificación a la cadena resulta en un deterioro de la evaluación. El algoritmo puede realizar varios ciclos utilizando en cada caso una cadena inicial distinta. El proceso termina cuando se haya encontrado una secuencia de bits óptima o se haya realizado un determinado número de evaluaciones.

La fase del escalador del STA utiliza solamente un ciclo que toma como cadena inicial a la mejor propuesta de solución encontrada en la fase genética. Dado que a la fase del escalador precede una fase genética en la que se encontró una buena solución, la intención es llevar a esta cadena al punto más alto de su vecindad.

Existen diferentes escaladores que es posible utilizar. En esta tesis se optó por el conocido en inglés como *Next-Ascent Hill Climbing*[18] (NAHC). A diferencia del escalador *Steepest-Ascent*, no necesita almacenar tantas cadenas como bits modificados para su posterior comparación. Esta característica provoca que los requerimientos de memoria se mantengan bajos, lo cual como veremos más adelante es deseable en algunos problemas (véase capítulo 5). Se trata de un algoritmo sistemático con un ciclo de modificaciones bien definido que nos lleva a una pronta terminación - situación deseable en la realización de esta tesis.

En los ejercicios de optimización realizados, la fase genética del STA nos acercó lo suficiente a un buen resultado, por lo que no fue necesario utilizar un escalador exhaustivo[19] como lo es el *Random Mutation Hill Climber* (RMHC). En algunos casos podría ser necesario aplicar este método.

El pseudo-código asociado con un ciclo del NAHC se muestra en la siguiente figura.

INICIALIZACIÓN
Sea β la mejor solución encontrada por la fase genética, α una cadena auxiliar, l la longitud de la cadena genética, f la función de evaluación y *mdf* un valor booleano que indica si hubo alguna mejora.

1. Copia la cadena β en α
2. *mdf* = verdadero
3. Mientras que *mdf* sea verdadero:

MODIFICACIÓN
3.1. *mdf* = falso
3.2. Para todo bit i desde 1 hasta l:
3.2.1. **Altere** el bit i de α: $\alpha_i = \sim \alpha_i$

EVALUACIÓN
3.2.2. **Decodifique** el valor propuesto de las variables en α
3.2.3. **Evalúe** al individuo α de acuerdo con sus variables y la función f.

EVOLUCIÓN
3.2.4. Si $f(\alpha)$ es mejor que $f(\beta)$ entonces:
Copie la cadena α en β para guardar el mejor valor
mdf = verdadero
sino:
Devuelva al bit i de α su valor original: $\alpha_i = \sim \alpha_i$
3.2.5. fin del para todo, vaya al paso 3.2
3.3. fin del mientras, vaya al paso 3

TERMINACIÓN
4. Muestre el resultado β y $f(\beta)$

Figura 7: Pseudo-código de la fase del escalador del STA.

[18] Utilizaré el término en inglés debido a que no existe una traducción adecuada.

[19] El termino "exhaustivo" en este contexto se aplica a todo algoritmo que a lo largo del tiempo puede analizar todas las posibilidades o combinaciones.

3 Experimentos y Resultados

Esta sección presenta los experimentos realizados para observar el comportamiento de la fase evolutiva genética del Algoritmo Estadístico en comparación con el Algoritmo Genético Simple con un punto de cruza[20] y con cruza uniforme con probabilidad de 0.5 en sus versiones elitistas[21] (una comparación entre las versiones elitistas y no-elitistas puede encontrarse en [KURI99b]).

Los valores presentados son el resultado de promediar cincuenta ejecuciones. La representación utilizada tiene un formato de punto fijo en el cual se concatena un bit S para denotar el signo, una serie de bits E que indican la parte entera del valor representado y una serie D extra, comúnmente de mayor tamaño, para señalar la parte fraccionaria: SEF. Las variables de los problemas se codificaron en Gray [HAMM80] debido a que la distancia Hamming[22] entre dos números consecutivos es en promedio menor que la existente en el código Binario Ponderado y esta característica es relevante en los Algoritmos Genéticos.

Todos los ejercicios inician con la misma población obtenida a partir de la misma semilla aleatoria. La misma secuencia de semillas aleatorias para las cincuenta repeticiones es utilizada en todos los experimentos. Se utilizó en la cruza en todo momento (i.e. probabilidad 1) y la probabilidad de mutación se estableció en 0.005 en todos los casos. Todas las ejecuciones finalizan al alcanzarse un máximo de 300 generaciones.

El conjunto de funciones utilizadas en estos experimentos corresponde a aquel presentado en [DEJO75]. Este conjunto de funciones ha sido utilizado ampliamente en diferentes algoritmos de optimización. Cada uno de ellos es ejemplo de alguna dificultad encontrada en problemas de optimización. Al hacer comparaciones entre estas funciones, es posible notar las fuerzas y debilidades de los algoritmos.

A continuación se muestran los resultados obtenidos para estos problemas de minimalización. En cada una de las funciones se muestran tres gráficas: una que corresponde al valor máximo obtenido en cada generación, una segunda con los valores promedio por generación y una última con el porcentaje de convergencia[23].

3.1 Modelo Esférico

La primer función presentada por De Jong es un modelo esférico n-dimensional simple. Esta función representa el caso ideal para cualquier algoritmo de optimización, ya que su comportamiento es suave, unimodal y simétrico. El desempeño en este problema da la medida de la eficiencia del sistema en general.

La ecuación del modelo esférico es la siguiente:

$$f_1(\bar{x}) = \sum_{i=1}^{3} x_i^2, \qquad -5.12 \le x_i \le 5.12$$

[20] Comparación tradicional que permite ser punto de referencia para la posterior comparación contra otros algoritmos.

[21] Se eligió este algoritmo debido a que su estructura y modelo de cruza se asemeja bastante a la utilizada en el Algoritmo Estadístico.

[22] La distancia Hamming entre dos secuencias de bits (i.e. palabras) se define como el mínimo número de bits en los cuales difieren ambas palabras. Por ejemplo, la distancia Hamming entre '001011' y '101111' es igual a dos.

[23] El porcentaje de convergencia se obtuvo al generar un vector de probabilidad para todos los algoritmos y calcular el porcentaje de las variables de este vector que sean menores que 0.1

El valor mínimo se encuentra cuando todas las variables son cero:

$$\min(f_1) = f_1(0,0,0) = 0$$

Para que el lector pueda darse una idea del modelo, se incluye en la siguiente figura una gráfica de la función tridimensional. Nótese que esta gráfica carece de una dimensión con respecto a la ecuación del problema.

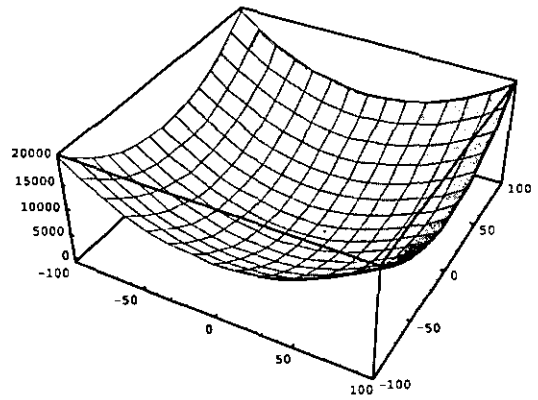


Figura 8: Gráfica tridimensional del Modelo Esférico.

El tamaño de la población, la élite y la representación utilizados en estos ejercicios se muestran en la siguiente tabla:

Tam.Pob.	Élite	Signo	Enteros	Fraccionales	Num.Var.	Long.Indiv.
60	6	Sí	7	15	3	69

Tabla 11: Tamaño de Población, Élite y Representación en el Modelo Esférico.

La siguiente gráfica muestra un comportamiento similar entre la versión elitista con cruza uniforme (SgaUE) del Algoritmo Genético Simple y el Algoritmo Estadístico (StaE) con respecto a los valores mínimos encontrados en cada generación[24], ambos obteniendo mejores resultados que los del algoritmo con un punto de cruza (SgaE) en este ejercicio.

[24] Nótese que a pesar de haber ejecutado los algoritmos hasta la generación trescientos, solamente se graficaron las primeras cien generaciones ya que el comportamiento a partir de ésta es monótono.

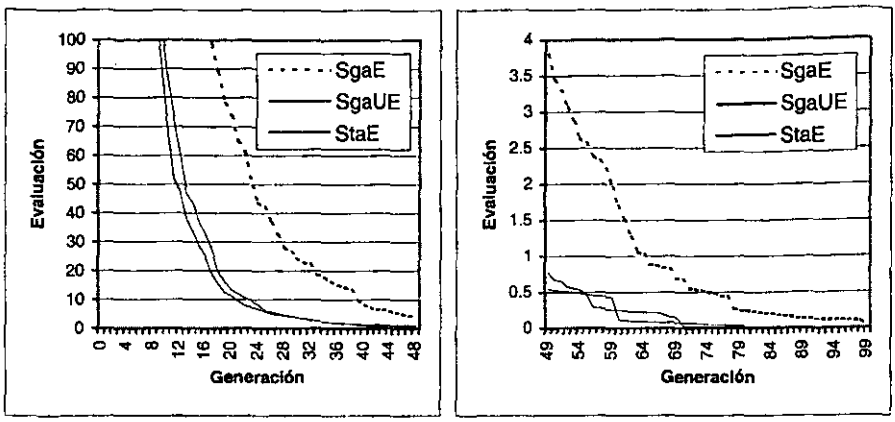


Figura 9: Gráficas del valor mínimo por generación en el Modelo Esférico.

En los casos en los que los valores de las variables de una propuesta genética estuvieron fuera de rango, se aplicó un valor de castigo K proporcional al número de restricciones no satisfechas (cada límite establece una restricción). De esta forma, la evaluación de un individuo se determina de la siguiente forma:

$$f(\vec{x}) = \begin{cases} f(\vec{x}) & \text{si cumple con todas las restricciones} \\ K - \sum_{i=1}^{s} \left(\frac{K}{m} \right) & \text{de lo contrario} \end{cases}$$

donde m es el número de restricciones, s es el número de restricciones satisfechas y K es un valor de castigo determinado arbitrariamente. Éste debe ser un número mucho mayor que aquellos provenientes de la función, por ejemplo, si sabemos que la función regresa valores entre cero y mil, un valor mucho mayor sería 10^9.

Cabe mencionar que el tratamiento de la función recién descrito, es heurístico. La intención es separar los valores de evaluación según el número de restricciones cumplidas, de tal forma que aquellas que cumplan con más restricciones serán mejor evaluadas. El lector podría optar por el uso de otra heurística si esto fuera conveniente para el problema en el cual se utilice.

Debido al tratamiento descrito anteriormente, los valores promedio de la generación son comúnmente mucho mayores que los de la gráfica previa.

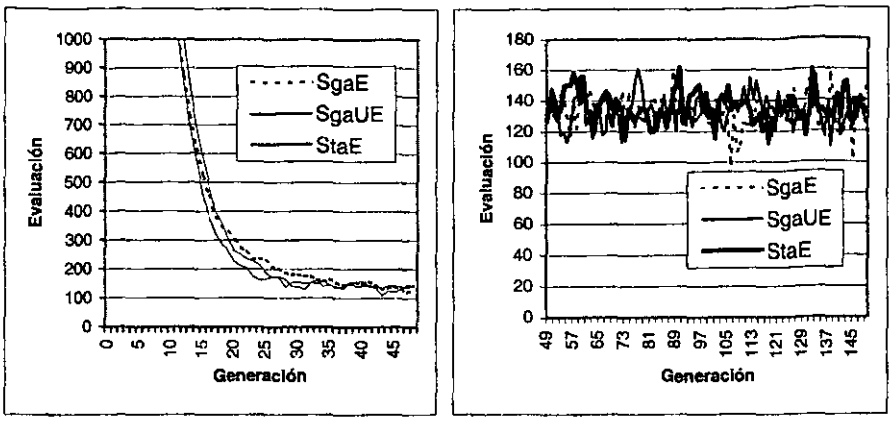


Figura 10: Gráficas del valor promedio por generación en el Modelo Esférico

Una diferencia notable en cuanto al comportamiento de los algoritmos comparados se encuentra al estudiar el nivel de convergencia de cada generación. El porcentaje de convergencia está directamente relacionado con la cantidad de información que aun puede ser explotada en los individuos de la población. Un algoritmo cuyo porcentaje de convergencia crezca rápidamente puede llevar a soluciones no óptimas.

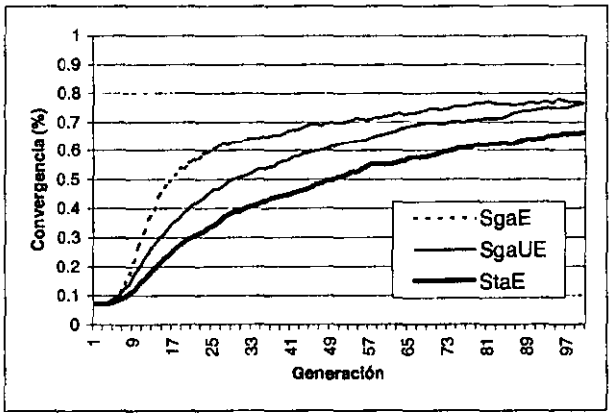


Figura 11: Gráfica del porcentaje de convergencia en el Modelo Esférico.

3.2 Función Generalizada de Rosenbrock

La segunda función de prueba utilizada por De Jong es conocida como Función de Rosenbrock. De manera contraria el modelo esférico, se trata de un problema complicado de optimización. Los algoritmos que no son capaces de descubrir direcciones adecuadas tienen un desempeño muy pobre en este ejercicio.

En esta tesis utilizamos la generalización a cinco dimensiones respetando los límites. De esta forma, se representa por la siguiente ecuación:

$$f_2(\bar{x}) = \sum_{i=1}^{4} 100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2, \qquad -2.048 \le x_i \le 2.048$$

Esta ecuación tridimensional, se minimiza cuando todas sus variables equivalen a uno, esto es:

$$\min(f_2) = f_2(1,1,1,1,1) = 0$$

La gráfica tridimensional de esta función entre los límites señalados se muestra a continuación:

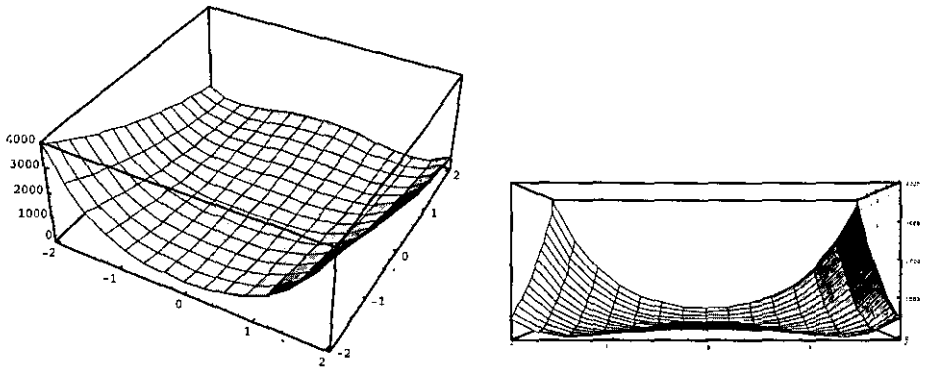


Figura 12: Gráfica Tridimensional de la Función Generalizada de Rosenbrock (dos vistas).

En este experimento se utilizaron los siguientes valores:

Tam.Pob.	Élite	Signo	Enteros	Fraccionales	Num.Var.	Long.Indiv.
100	10	Sí	2	15	5	90

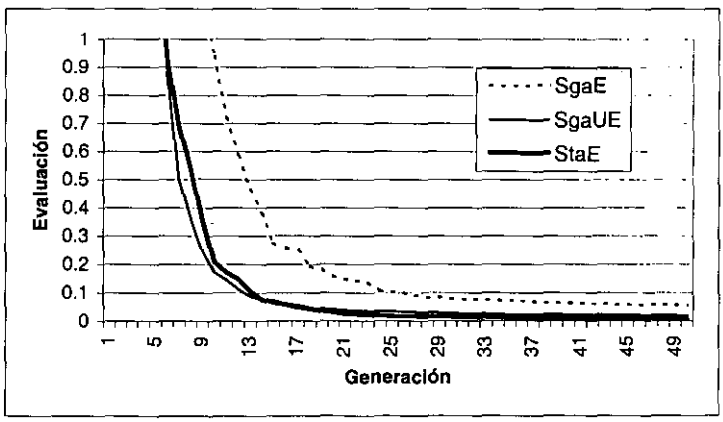


Figura 13: Gráfica del valor mínimo por generación en la Función Generalizada de Rosenbrock.

En la Figura 13 encontramos un comportamiento similar al obtenido en el experimento anterior. El Algoritmo Genético Simple con cruza uniforme (SgaUE) es superior en general al de un punto de cruza (SgaE), esto ha sido comprobado también en otros resultados teóricos [SPEA91]. El Algoritmo Genético Estadístico (StaE) compite con el SgaUE y logra superar su comportamiento solamente después de la vigésima generación.

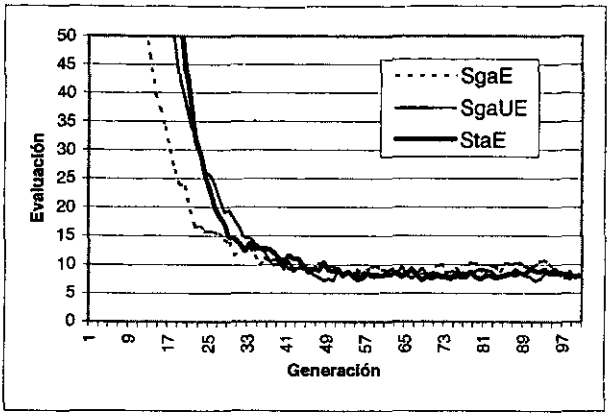


Figura 14: Gráfica del valor promedio por generación en la Función Generalizada de Rosenbrock.

Nuevamente encontramos que el valor promedio del SgaE es mejor que el de los otros dos algoritmos en las primeras generaciones (véase Figura 14). Este comportamiento se explica al observar la siguiente gráfica en la que se nota que tanto SgaUE como StaE tienen mayor variabilidad en sus propuestas, implicado por el menor porcentaje de convergencia, lo que conduce a un valor promedio mayor.

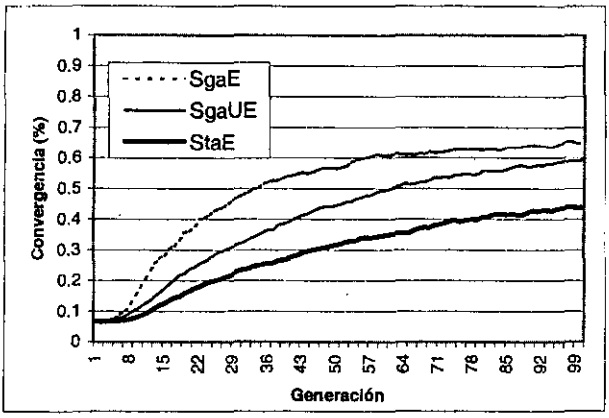


Figura 15: Gráfica del porcentaje de convergencia en la Función Generalizada de Rosenbrock.

3.3 Modelo Entero

El modelo entero es una función escalonada creciente que se representa por la suma de los valores enteros de sus variables. El modelo entero es el caso representativo de las superficies planas. Las superficies planas son un obstáculo para la optimización debido a que no proporcionan información respecto a cuál dirección es favorable. Los algoritmos que utilizan tamaños de paso fijo pueden encontrar grandes problemas en el modelo entero.

$$f_3(\vec{x}) = \sum_{i=1}^{5} \text{integer}(x_i), \qquad -5.12 \le x_i \le 5.12$$

Fácilmente puede observarse que el valor mínimo de esta función se encuentra cuando el entero todas las variables equivale al mínimo entero posible dentro del dominio especificado, en este caso –5. Lo anterior implica que no es un punto único el que minimiza la función sino un escalón consistente de todas las combinaciones de las variables en el intervalo [-5.12, -5]

$$\min(f_3) = f_3(-5,-5,-5,-5,-5) = -25$$

En la siguiente figura podemos observar dos vistas del modelo entero para el caso de tres dimensiones.

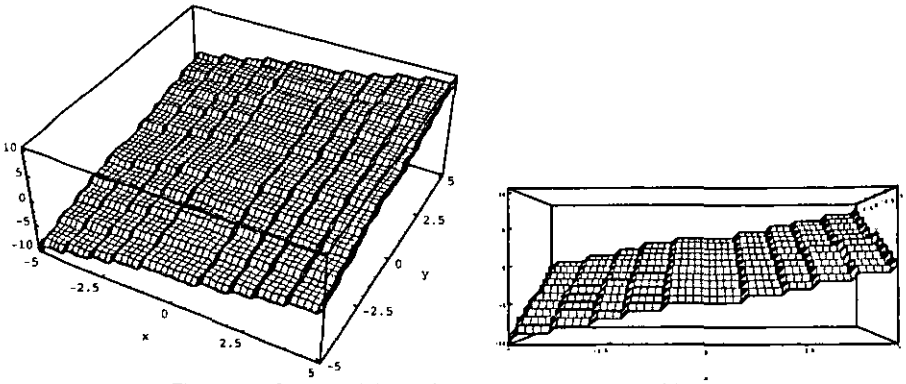


Figura 16: Gráfica tridimensional del modelo entero (dos vistas).

En este experimento se utilizaron los siguientes valores:

Tam.Pob.	Élite	Signo	Enteros	Fraccionales	Num.Var.	Long.Indiv.
60	6	Sí	3	15	5	95

Esta representación es adecuada si para su elección se considera exclusivamente el dominio del problema, sin considerar la ecuación de la función. De antemano, sabemos que una gran cantidad de bits no proporcionan información alguna para determinar el valor mínimo de la función. Específicamente, existen setenta y cinco bits que representan valores fraccionales que son desechados por la función entera. Solamente el veintiuno por ciento de los bits proporcionan información útil lo cual dificulta la búsqueda del valor mínimo.

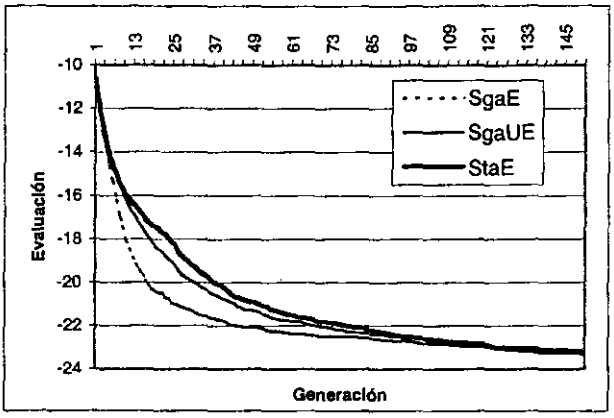


Figura 17: Gráfica del valor mínimo por generación en el Modelo Entero.

De acuerdo con los experimentos realizados, el Algoritmo Genético Simple con un punto de cruza obtiene mejores resultados en las primeras generaciones con respecto al valor mínimo y en general con respecto al valor promedio en comparación con los otros dos algoritmos.

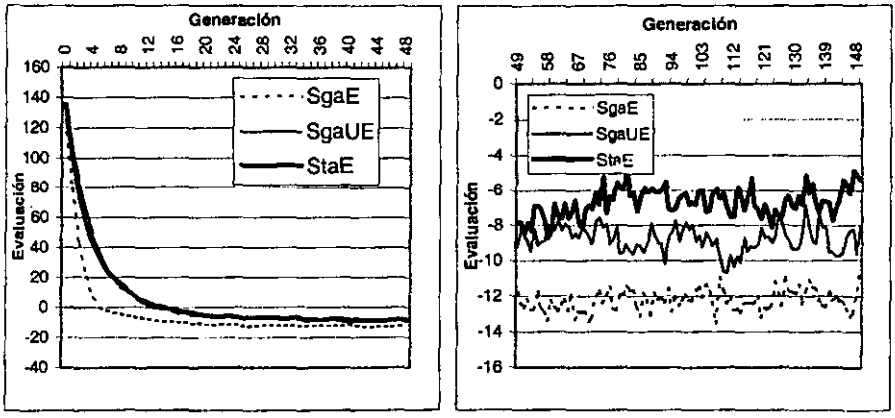


Figura 18: Gráficas del valor promedio por generación en el Modelo Entero.

Es de llamar la atención que a pesar de que el comportamiento con respecto al valor mínimo y promedio del SgaUE y el STAE es similar, no lo es así cuando se observa la gráfica de convergencia (Figura 19). Existe una relación proporcionalmente inversa entre el porcentaje de convergencia y la desviación estándar de la población. Esto indica que entre los individuos del STAE existe mayor diversidad que la encontrada en los otros algoritmos.

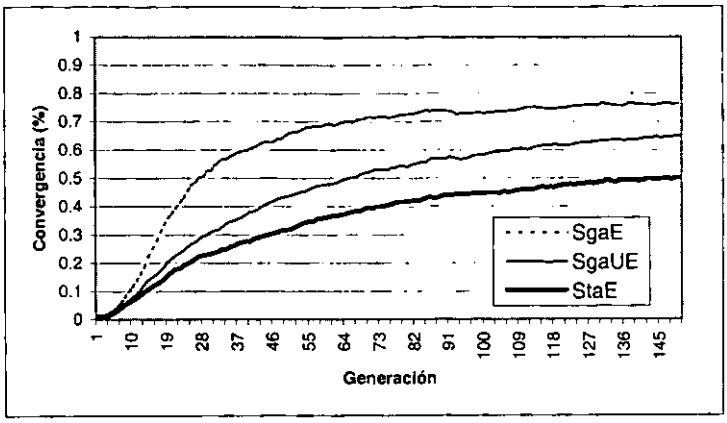


Figura 19: Gráfica del porcentaje de convergencia en el Modelo Entero.

3.4 Función Cuártica con Ruido

Un problema extra para los algoritmos de optimización se encuentra cuando algunas de los puntos encontrados proporcionan información falsa al sistema, conocida como ruido. Este es el caso de la cuarta función utilizada por De Jong. Se trata de una función cuártica ponderada a la cual se le ha introducido ruido con distribución de Gauss.

A diferencia de las funciones anteriores, se utilizó un nivel alto de dimensiones que provoca una mayor dificultad en la búsqueda de una solución óptima.

$$f_4(\bar{x}) = \sum_{i=1}^{30} i x_i^4 + \text{Gauss}(0,1) \qquad -1.28 \le x_i \le 1.28$$

Debido al ruido no es posible determinar con precisión el valor mínimo, sin embargo, sabemos que este se encuentra en o muy cercano al origen.

$$\min(f_4) = f_4(0,0,0,\ldots,0) = 0$$

La figura que se muestra a continuación nos da idea de la forma de la función.

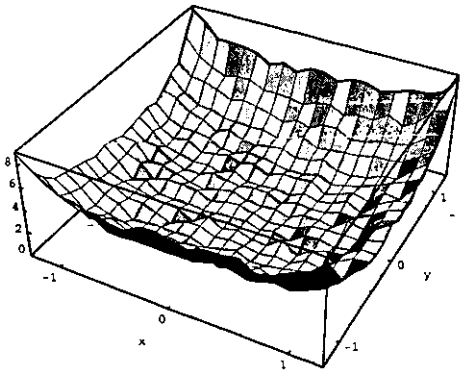


Figura 20: Gráfica tridimensional de la Función Cuártica con Ruido.

En esta ocasión se seleccionaron los siguientes parámetros:

Tam.Pob.	Élite	Signo	Enteros	Fraccionales	Num.Var.	Long.Indiv.
100	10	Sí	1	15	30	510

Los resultados obtenidos fueron similares a los de los primeros experimentos. El valor mínimo del SgaUE y del STAE es significativamente menor que el de SgaE en todas las generaciones.

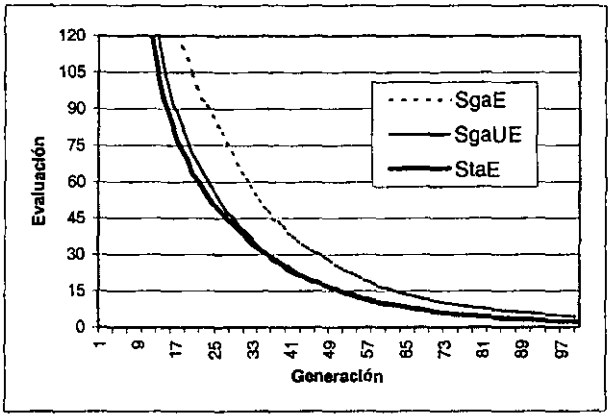


Figura 21: Gráfica del valor mínimo por generación en la Función Cuártica con Ruido.

El valor promedio del SgaE es el mejor en las primeras generaciones y es alcanzado por los otros dos algoritmos en generaciones posteriores.

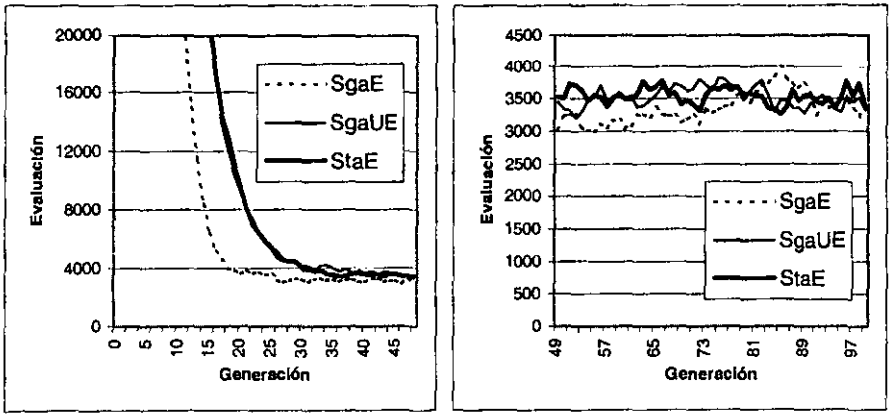


Figura 22: Gráficas del valor promedio por generación en la Función Cuártica con Ruido.

La dificultad del problema debida a su alta dimensionalidad se ve reflejada en el bajo porcentaje de convergencia que alcanzan los algoritmos. Aún así, se mantiene la relación encontrada en los demás experimentos.

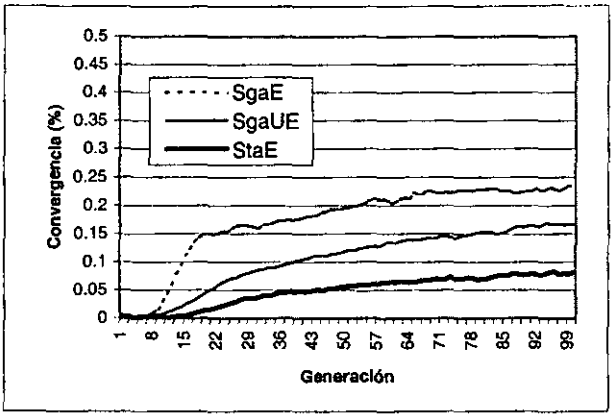


Figura 23: Gráfica del porcentaje de convergencia en la Función Cuártica con Ruido.

3.5 Función Foxholes de Shekel

La función Foxholes de Shekel tiene un comportamiento multimodal contiene veinticinco mínimos locales. Está descrita por la siguiente ecuación:

$$f_5(\vec{x}) = \frac{1}{0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}} \qquad -65.536 \le x_i \le 65.536$$

donde,

$$a_{ij} = \begin{cases} -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, \\ -32, -32, -32, -32, -32, -16, -16, -16, -16, -16, 0, 0, 0, 0, 0, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32 \end{cases}$$

El valor mínimo global es cercano a uno

$$\min(f_{14}) = f_{14}(-32,-32) \approx 1$$

La siguiente Figura muestra el comportamiento de esta función.

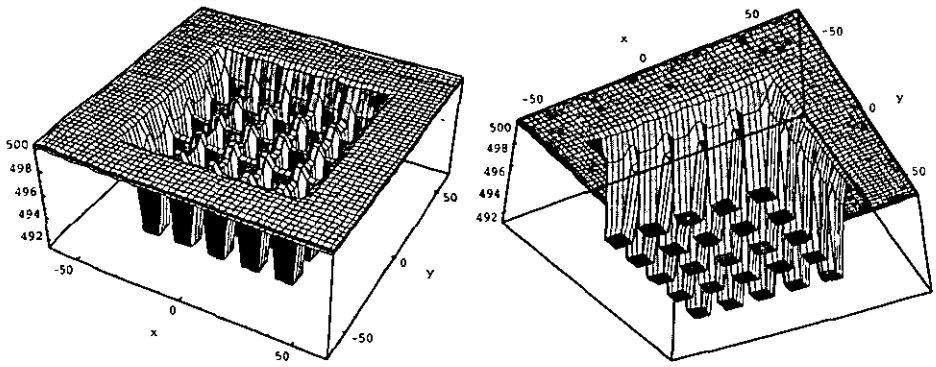


Figura 24: Gráfica de la Función Foxholes de Shekel (dos vistas).

La función Foxholes de Shekel tiene mayor complejidad para su optimización que las funciones anteriores debido a sus múltiples óptimos locales[25]. Los óptimos locales son puntos de atracción fuerte que desvían la atención de los algoritmos hacia respuestas incorrectas evitando la exploración de otras áreas del espacio de búsqueda.

En una implementación adecuada de un algoritmo genético es común encontrar valores grandes de mutación para evitar caer en la trampa de los óptimos locales. En este ejercicio se utilizó el mismo valor de mutación que en los experimentos anteriores para efectos de estandarización. Los parámetros utilizados fueron:

Tam.Pob.	Élite	Signo	Enteros	Fraccionales	Num.Var.	Long.Indiv.
40	4	Sí	7	15	2	46

En esta ocasión, los algoritmos SgaUE y STAE obtuvieron mejores resultados tanto en el valor mínimo como en el promedio en todas las generaciones.

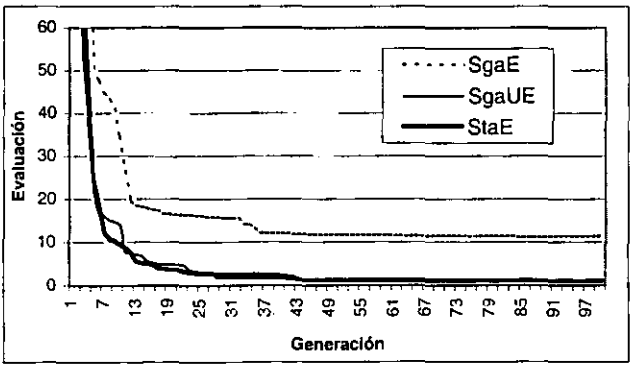


Figura 25: Gráfica del valor mínimo por generación en la Función Foxholes de Shekel.-

[25] En la Figura 24, los óptimos locales aparentan tener el mismo valor. Esto se debe a que el rango representado en la gráfica es muy grande comparado con la diferencia entre estos valores.

La cruza uniforme produce mayor variabilidad entre los individuos de la población que la encontrada con la cruza de un solo punto. Posiblemente esta sea la razón por la cual los algoritmos SgaUE y STAE se vean menos afectados por el efecto de los mínimos locales.

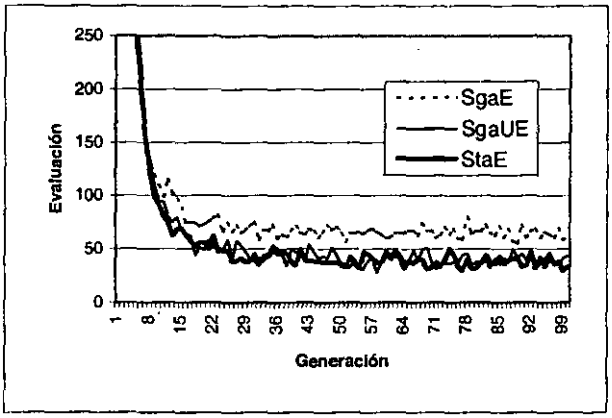


Figura 26: Gráfica del valor promedio por generación en la Función Foxholes de Shekel.

El SgaE al verse atraído por un mínimo local y tener poca probabilidad de mutación, incrementa considerablemente su porcentaje de convergencia.

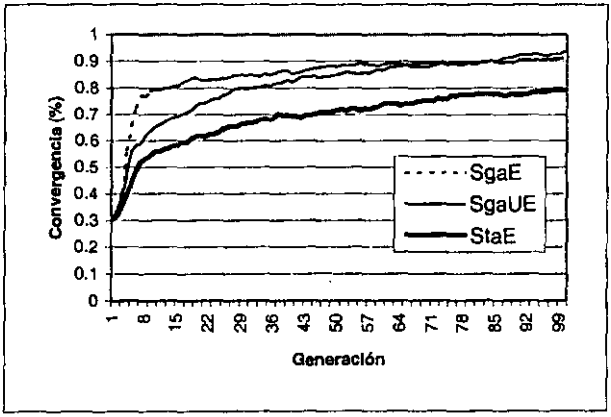


Figura 27: Gráfica del porcentaje de convergencia en la Función Foxholes de Shekel.

4 Parámetros

Al momento de definir un algoritmo evolutivo para la solución de un problema es preciso determinar sus componentes, tales como los operadores de exploración adecuados a la representación utilizada (p.ej. cruza y mutación), los mecanismos para seleccionar y utilizar la información relevante (p.ej. elitismo) y la cantidad de muestras en cada etapa (p.ej. generacional). Cada uno de los componentes puede contener parámetros que modifican su comportamiento, por ejemplo la probabilidad de cruza, el tamaño de la población y la élite.

El valor de los parámetros determina en gran medida si el algoritmo encontrará una solución óptima y si el proceso será eficiente. La selección de parámetros adecuados varía de acuerdo con cada problema. Es una tarea que requiere, por lo general, de un esfuerzo considerable.

En el ambiente de investigación evolutiva son comunes dos prácticas: la afinación y el control de parámetros. La afinación se realiza cuando se determinan los valores de los parámetros antes de ejecutar el algoritmo y éstos no se modifican durante el proceso. El control de los parámetros pretende ajustar los valores de los mismos conforme se requiera en la ejecución[26].

El Algoritmo Genético Estadístico se diseñó teniendo como principio la utilización del menor número de parámetros posible. La idea fue que al no ocupar los operadores genéticos clásicos, no se tendrían que determinar o controlar sus parámetros.

En esta tesis, definimos a priori el valor de los parámetros, manteniéndolos estáticos durante la ejecución. Sin embargo, es posible utilizar algunas de las técnicas propuestas en la literatura para controlar dinámicamente estos valores.

Los parámetros que determinan el comportamiento del Algoritmo Genético Estadístico son: la probabilidad de mutación, el tamaño de la población y el tamaño de la élite. En la literatura de Algoritmos Genético podemos encontrar los efectos que estos parámetros tienen sobre el sistema. Dado que el Algoritmo Genético Estadístico no sigue la estructura tradicional de los Algoritmos Genéticos, se consideró prudente hacer algunos experimentos al respecto. En las siguientes secciones se analiza la manera en la cual los valores de estos parámetros modifican al sistema. Para ello, se utilizó la siguiente función:

$$f_6(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

cuyo valor mínimo es:

$$\min(f_6) = f_6(1,1) = 0$$

En todos los experimentos tomamos como base los siguientes parámetros elegidos arbitrariamente. Naturalmente, para cada caso se modificó sólo uno de los parámetros.

[26] Para una clasificación más precisa, véase [ElBEN99].

Parámetro	Valor
Repeticiones	50
Máximo de Generaciones	300
Tamaño de la Población	100
Élite	1
Probabilidad de Mutación	0.005
Límite de Convergencia[27]	0.01

Tabla 12: Parámetros base para los experimentos de f6.

4.1 Mutación

La mutación tiene como objetivo generar nuevas combinaciones en la cadena genética que permitan explorar nuevas áreas del espacio de búsqueda. Esto se logra modificando un porcentaje de los bits de una población original. La posición de los bits mutados puede distribuirse de distintas formas. La distribución uniforme es la más común.

Teóricamente, las fluctuaciones estadísticas producidas en la creación y utilización del vector probabilístico del STAE son suficientes para explorar continuamente las vecindades de la zona actual de búsqueda. Sin embargo, observamos que en la práctica resultaba benéfico introducir un factor pequeño de mutación.

De acuerdo con los experimentos realizados y como era esperado, encontramos que existe un valor límite de probabilidad de mutación arriba del cual la búsqueda se deteriora. Esto se debe a que una mutación muy alta degenera demasiado los patrones hasta el momento encontrados y no permite la correcta explotación de la información. El valor límite de mutación varía de un problema a otro y se determina todavía de manera empírica.

En la Figura 28 observamos que el límite para la probabilidad de mutación es cercano a 0.05 para la función f6 ya que valores mayores de mutación no mejoran los resultados obtenidos en el límite.

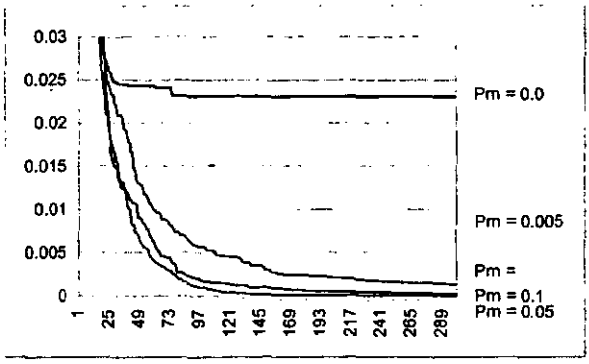


Figura 28: Comparación del efecto producido en el valor mínimo por diferentes porcentajes de mutación en f6.

[27] Se considera que una dimensión del vector de probabilidad (i.e. bit) ha convergido si su valor es menor que el límite de convergencia o mayor que (1-límite de convergencia).

En la siguiente gráfica vemos los valores mínimos obtenidos al final del algoritmo para cada porcentaje de mutación.

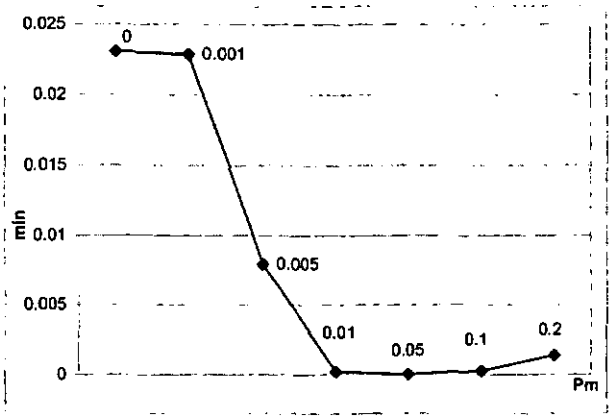


Figura 29: Gráfica de valores mínimos encontrados utilizando diferentes porcentajes de mutación en f6.

Un porcentaje de mutación mayor implica que se explora un área más amplia del espacio de búsqueda, lo que implica que la desviación estándar encontrada en cada generación es también mayor. Naturalmente al ser mayor la desviación estándar produce un efecto directamente proporcional en el promedio encontrado en cada generación. En la siguiente figura se observa claramente esta situación.

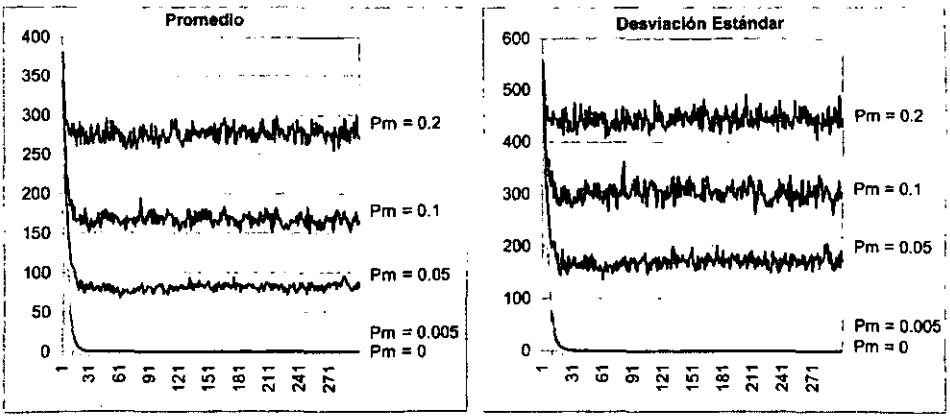


Figura 30: Comparación del efecto producido en el promedio y en la desviación estándar por diferentes probabilidades de mutación en f6.

El comportamiento encontrado en la desviación estándar es inverso al que ocurre con respecto al porcentaje de convergencia. Un sistema en el que la desviación estándar es pequeña en las primeras generaciones tendrá una convergencia temprana, posiblemente en mínimos locales.

Cabe hacer notar que a diferencia de un algoritmo genético tradicional, la fase estadística del algoritmo de esta tesis alcanza un nivel de estabilidad en cuanto al porcentaje de convergencia. Es decir, no está garantizada la convergencia. La razón de ello es debido a limitaciones en la representación del vector de probabilidad. Este tema se analizará posteriormente.

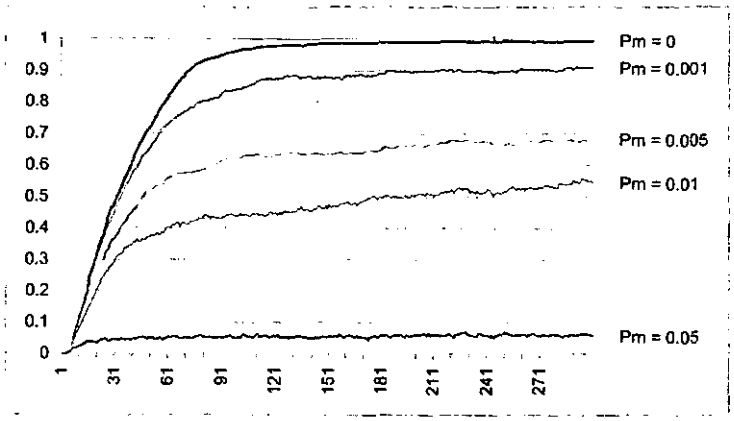


Figura 31: Comparación del efecto producido en el porcentaje de convergencia por diferentes probabilidades de mutación en f6.

4.2 Tamaño de la Población

El tamaño de la población es uno de los parámetros más importantes que determinan la calidad de la solución. Las poblaciones de gran tamaño llevan a mejores resultados, sin embargo en ocasiones desperdician una gran cantidad de recursos de cómputo. Por el contrario, poblaciones pequeñas no logran representar adecuadamente el espacio de búsqueda y pueden provocar estancamiento en óptimos locales.

Establecer un tamaño de población adecuado no es una tarea sencilla. A pesar de que existen diversos trabajos teóricos sobre el tema (p.ej. [GOLD92],[HARI97]), los investigadores en muchas ocasiones especifican todavía este parámetro de manera intuitiva, ya que los modelos teóricos contienen suposiciones que no son siempre válidas en los casos reales.

En estos desarrollos teóricos se reconoce que en ocasiones los algoritmos pueden cometer errores al discriminar los bloques constructores de soluciones. Al incrementar el tamaño de la población se obtiene un muestreo más representativo de los bloques constructores[28] y debido a ello se reducen los errores en la toma de decisión. Se sabe por estos estudios que el tamaño de la población debe ser proporcional al tamaño del espacio de búsqueda y a la relación de la respuesta de los bloques constructores con respecto al ruido.

Los experimentos realizados corroboran las afirmaciones anteriores. En la siguiente gráfica, observamos que el valor mínimo que se obtiene es menor conforme la población es mayor. Vemos también que la ganancia obtenida al incrementar el tamaño de la población es exponencialmente menor.

[28] La teoría de Algoritmos Genéticos está basada en el concepto de los esquemas (patrones que describen a un conjunto de cadenas con similitudes en ciertas posiciones). El teorema fundamental de Algoritmos Genéticos indica que los esquemas de orden pequeño (aquellos con pocos valores fijos) dan lugar a esquemas de orden mayor. Los esquemas de orden pequeño son conocidos también como bloques constructores.

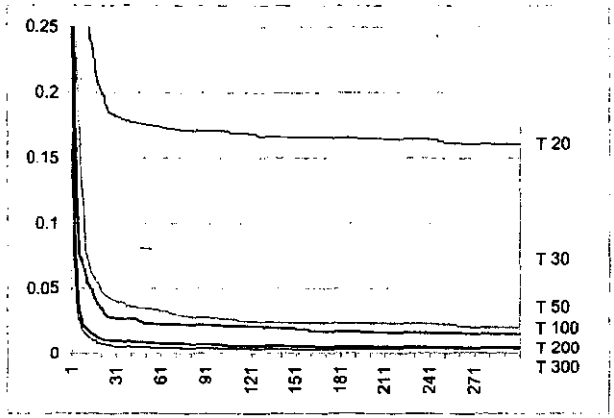


Figura 32: Comparación del efecto producido en el valor mínimo con diferentes tamaños de población en f6.

El Algoritmo Genético Estadístico no utiliza el operador de cruza, lo que le permite encontrar patrones que no se encontraban en la generación anterior con mayor facilidad. Este método basado en un vector de probabilidad provoca que la desviación estándar se incremente a medida que se incrementa el tamaño de la población. Este efecto se ve reflejado también en el valor promedio de cada generación. A diferencia de lo ocurrido en la sección anterior al variar la mutación, observamos que las variaciones en los valores del promedio y la desviación estándar son inversamente proporcionales al tamaño de la población.

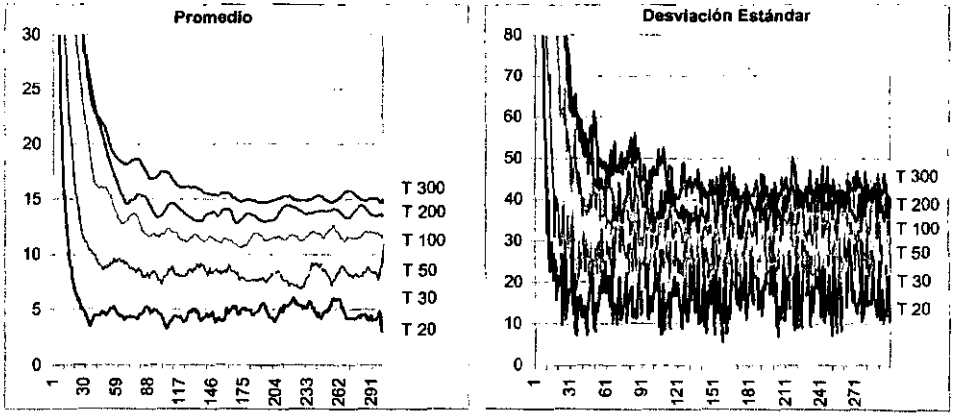


Figura 33: Comparación del efecto producido en el promedio y la desviación estándar por diferentes tamaños de población en f6.

En la fase genética del STAE, el tamaño de la población tiene una influencia directa en el porcentaje de convergencia. Así como la desviación estándar llega a un nivel promedio estable, también lo hace el porcentaje de convergencia, sin poderse garantizar que ésta ocurrirá. De ahí la importancia de la fase del escalador.

El vector de probabilidad tal como está planteado en esta tesis, no es más que un modelo de la población. Entre mayor cantidad de propuestas de solución existan en la población, los cambios producidos en el vector de probabilidad son menores, es decir, el tamaño del paso en cada

dirección es menor. Esto implica que los cambios en el porcentaje de convergencia son también más pequeños y el nivel de convergencia alcanzado es menor.

Desde este punto de vista, tiene sentido pensar en un algoritmo con tamaños de población variable. Este es un tema para futuras investigaciones.

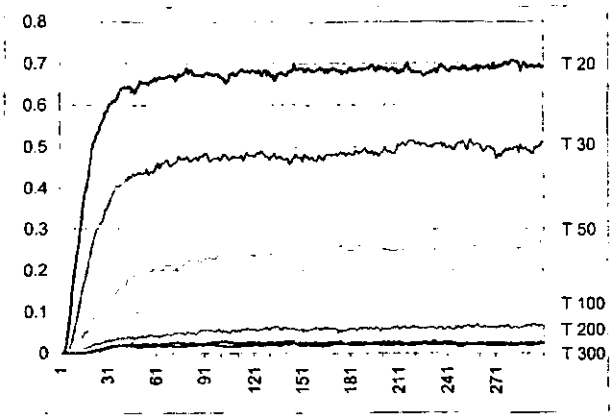


Figura 34: Comparación del efecto producido en el porcentaje de convergencia por diferentes tamaños de población en f6.

4.3 Élite

El elistismo es una técnica mediante la cual los algoritmos preservan a uno o más de los individuos mejor adaptados de una generación a otra. Los sistemas que no utilizan elitismo "olvidan" los resultados obtenidos en generaciones anteriores posiblemente mejores que los obtenidos en la nueva generación. En algunos casos este olvido puede ser deseable, por ejemplo si se trata de sistemas dinámicos cuyo valor de adaptación varía en el tiempo.

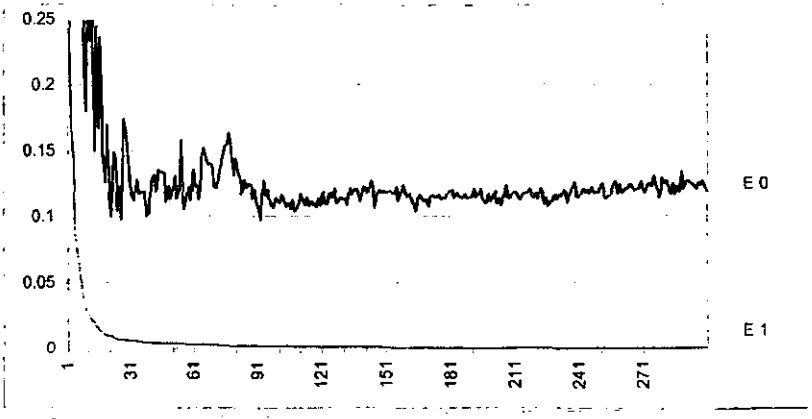


Figura 35: Comparación del efecto producido por el elitismo en f6.

La gráfica anterior nos muestra el efecto del elitismo con respecto al valor mínimo. Claramente podemos observar las variaciones que existen en la línea correspondiente al sistema sin elitismo (i.e. denotada por E0). Un efecto extra que produce el elitismo son los resultados considerablemente menores que se obtienen.

El elitismo del que hablamos, también conocido como elitismo parcial, puede retomar de la generación anterior no solamente al mejor individuo, sino a un conjunto de ellos. Típicamente el tamaño del conjunto de mejores individuos escogido es pequeño, ya que de lo contrario se corre el riesgo de sesgar demasiado la búsqueda en lugar de guiarla solamente. Esta situación se observa en los experimentos realizados con diferentes tamaños de grupo élite y cuyos resultados se muestran a continuación.

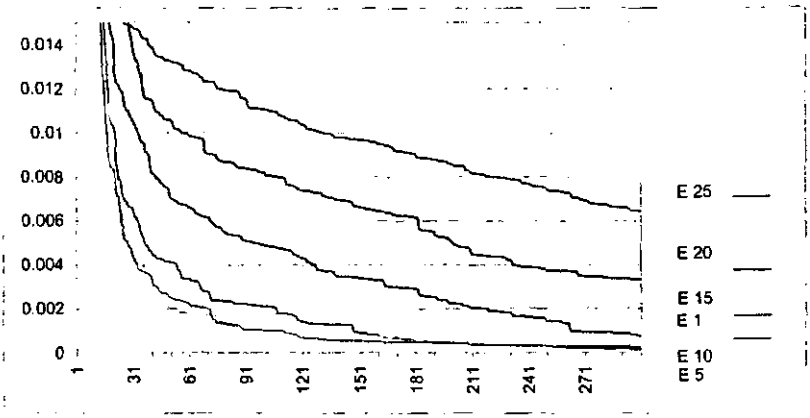


Figura 36: Comparación del efecto producido en el valor mínimo por diferentes tamaños de élite en f6.

Con respecto al comportamiento general de la población, típicamente entre mayor sea el número de muestras dentro de la élite, el promedio y la desviación estándar tenderán a ser menores. Esta situación es favorable cuando se trabaja con funciones sencillas unimodales ya que el sesgo acelerará el proceso. En funciones con muchos mínimos locales, existe una probabilidad alta de terminar el algoritmo en un subóptimo si se trabaja con una élite grande.

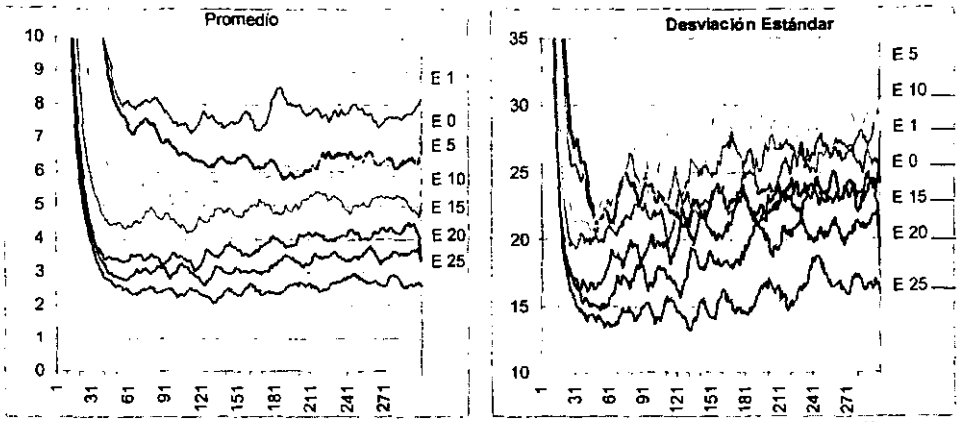


Figura 37: Comparación del efecto producido en el promedio y la desviación estándar[29] por diferentes tamaños de élite en f6.

En los algoritmos genéticos tradicionales, es posible demostrar que utilizar elitismo es una condición necesaria y suficiente para que el sistema converja al óptimo global (véase [KURI99a,pp.206]). Debido al modelo probabilístico utilizado en la fase genética del STAE, el elitismo es una condición necesaria, pero no suficiente. Esta representación tiene limitantes, mismas que analizaremos posteriormente, que no permiten garantizar la convergencia.

En la siguiente gráfica observamos cómo el sesgo inducido por las élites de gran tamaño contribuye en el porcentaje de convergencia.

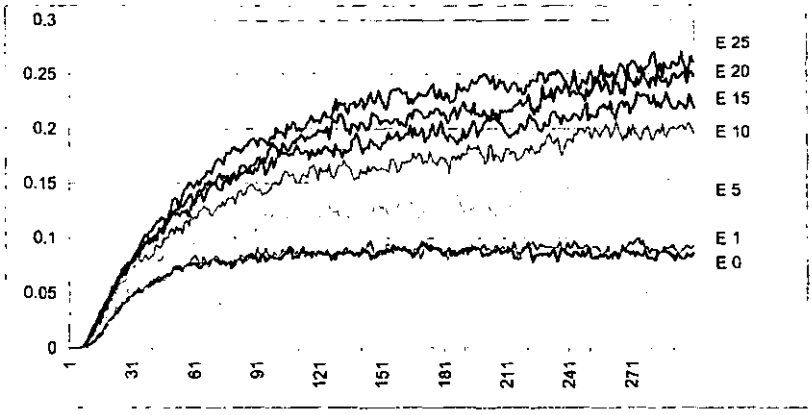


Figura 38:Comparación del efecto producido en el porcentaje de convergencia por diferentes tamaños de élite en f6.

[29] Debido a que existen variaciones grandes de una generación a otra, no se grafican los datos sino líneas de tendencia obtenidas al promediar a los valores contenidos en una ventana móvil de tamaño 10.

5 Uso de Memoria

Los avances tecnológicos alcanzados hasta ahora han permitido que los circuitos de memoria sean más compactos y más baratos comparados con lo que se tenía hace algunos años. Hoy contamos con computadoras personales que cuentan con capacidades de memoria que superan por mucho a los servidores más potentes que existían. Esta revolución tecnológica ha producido que los programadores dejen de preocuparse, en la mayoría de los casos, por la cantidad de memoria que consumen sus sistemas.

Si bien lo anterior es cierto, en el caso de los problemas que se tratan en Inteligencia Artificial no es difícil imaginar situaciones en las que se tenga que tratar con espacios de búsqueda inmensos que superan las capacidades de almacenamiento actuales.

Debido a lo anterior, es importante contar con algoritmos que puedan reducir sus requerimientos de memoria sin que ello implique un incremento significativo en el tiempo de proceso. A continuación se presenta la manera en que el Algoritmo Genético Estadístico puede decrementar la cantidad de memoria necesaria en problemas con poblaciones grandes

Dado que el vector de probabilidad es un modelo de la distribución de la población y que este modelo es suficiente para guiar la búsqueda, podemos evitar almacenar toda la población si se realizan cálculos parciales que consideren la contribución de cada individuo al nuevo vector de probabilidad.

Los cálculos parciales requieren de una estructura igual a la del vector de probabilidad para su almacenamiento. Debido a que el vector de probabilidad está calculado a partir de los valores normalizados y la normalización se basa en el valor mínimo o máximo, conocido solamente hasta haber generado todos los individuos de la población, es preciso una estructura adicional del mismo tamaño para contabilizar el número de datos que se han considerado por cada bit.

Si suponemos un problema con cadenas genéticas de longitud l y un tamaño de población n, podemos considerar que un algoritmo genético consumiría típicamente nl bits[30]. El Algoritmo Genético Estadístico, como está planteado en el capítulo 2 requiere adicionalmente $32l$ bits para el vector de probabilidad suponiendo una representación de punto flotante estándar (i.e. 32 bits).

Con las modificaciones planteadas, solamente se almacena un individuo[31], el vector de probabilidad y dos vectores auxiliares. Lo que resulta en una reducción considerable en poblaciones grandes:

SGA:			nl
STA:			$(n+32)l$
STA compacto:	1 individuo:	l	
	vector de probabilidad:	$32l$	
	dos vectores auxiliares:	$\underline{64l}$	
			$97l$

Nótese que la reducción obtenida $97l$ no depende de n, lo que significa que sin importar el tamaño de la población, el espacio de memoria utilizado es el mismo.

[30] En una implementación real podría haber un desperdicio extra debido a un redondeo a un múltiplo del tamaño de palabra y algunos bytes adicionales para almacenar los parámetros del sistema y estructuras auxiliares.

[31] En caso de utilizar elitismo tendrán que almacenarse todos los individuos que pertenezcan a la élite. Los vectores auxiliares son independientes del elitismo.

En la siguiente figura se muestran las modificaciones propuestas para reducir el espacio de memoria consumido.

INICIALIZACIÓN GLOBAL
Sea c una cadena genética de longitud *l*, P y Q dos vectores de probabilidad con *l* variables, U un vector de valores enteros con la misma longitud.
Sea *N* el número de individuos i en la población, m la probabilidad de mutación, g el número de generación actual y *f* la función de evaluación.

1. Sea g = 0 y $P_k = 0.5$ para toda k tal que $1 <= k <= l$

CONDICIÓN DE TERMINACIÓN
2. Verifique si se ha cumplido alguna condición de **terminación**. En caso afirmativo, vaya al paso 7; de lo contrario continue en el paso 3.

INICIALIZACIÓN GENERACIONAL
3. Sea $Q_k = 0$ y $U_k = 0$ para toda k tal que $1 <= k <= l$
4. Para cada i desde 1 hasta N:

GENERACIÓN
4.1. Para cada j desde 1 hasta *l*:
4.1.1. **Genere** dos valores aleatorios $0 <= \rho_j, \rho_k < 1$ de una distribución uniforme
4.1.2. Si $\rho_j > m$:
$c_j = 1$ si $\rho_k <= P_j$, 0 de otro modo
sino **mute** estadísticamente:
$c_j = 0$ si $\rho_k <= P_j$, 1 de otro modo

EVALUACIÓN
4.2. Decodifique el valor propuesto de las variables.
4.3. Evalúe al individuo de acuerdo con sus variables y la función *f*. Llame a este valor de evaluación e.

EVOLUCIÓN ANTICIPADA
4.4. Si i=1 entonces mín=e, suma=e, de lo contrario:
4.4.1. Si e<mín entonces mín=e
4.4.2. suma = suma + e
4.5. Para cada k desde 1 hasta *l*:
4.5.1. Si $c_k = 1$ entonces:
4.5.1.1. $Q_k = Q_k + e$
4.5.1.2. $U_k = U_k + 1$

ACTUALIZACIÓN DEL VECTOR
5. Para cada k desde 1 hasta *l*:
5.1. $P_k = \frac{Q_k + U_k * mín}{suma + N * mín}$
6. Incremente el número g de generación actual. Vaya al paso 2.

RESULTADO
7. Almacene el resultado.
8. Continúe con la fase del escalador.

Figura 39: Fase Genética del STA compacto.

6 Investigación Futura

La búsqueda guiada por un modelo probabilístico de la población presenta algunas ventajas sobre los algoritmos genéticos tradicionales. Reduce el número de parámetros definidos empíricamente

por el usuario. Permite utilizar menor cantidad de recursos computacionales y facilita el estudio del comportamiento del sistema dado el modelo explícito existente en cada generación.

Sin embargo, la definición del modelo probabilístico, como está planteado en el Algoritmo Genético Estadístico, impone algunas dificultades y limitantes. En este capítulo se describen los problemas encontrados con el fin de que sirvan como guía para futuras investigaciones.

6.1 Escalamiento

El método utilizado por el Algoritmo Genético Estadístico en la creación del modelo de la población está basado en un esquema de selección proporcional, coloquialmente conocido como selección de ruleta, usado en el Algoritmo Genético Simple. Este esquema está asociado con la idea de la supervivencia de los individuos mejor adaptados.

Para su utilización se requiere que los valores de evaluación se escalen en un rango de valores positivos menores que uno. Existen diferentes formas de escalar los valores [BÄCK96], por ejemplo, de manera lineal estática o dinámica, logarítmica o exponencial, entre otras. Con los estudios actuales, no es claro cómo afectan estos métodos a los resultados finales del algoritmo.

El método de escalamiento propuesto en el STA es lineal sin desplazamiento:

$$\Phi_i = \frac{e_i - \min(e_i)}{\Sigma(e_i - \min(e_i))}$$

A diferencia de los métodos utilizados tradicionalmente, éste no impone como condición para el escalamiento que todos los valores sean distintos de cero. De hecho, uno de ellos siempre es igual que cero (i.e. el mínimo). Esto provoca que dicho individuo no contribuya en la creación del modelo de probabilidad. La ventaja de ello radica en que la diferencia existente entre los valores se maximiza y esto es benéfico en la búsqueda.

Cuando la diferencia entre los valores es grande, el sistema puede discriminar estadísticamente con facilidad los esquemas favorables de los que no lo son. Conforme avanza el algoritmo, los valores de evaluación tienden a ser más y más similares. Lo anterior provoca que las diferencias estadísticas sean muy pobres, y por ende, la información necesaria para discriminar también se reduce. Es en este momento cuando es preciso continuar el algoritmo en su fase del escalador, ya que de lo contrario no se puede garantizar la convergencia.

6.2 Función de Actualización

Durante el desarrollo de esta tesis, se encontraron otros algoritmos que hacen una estimación de la distribución de probabilidad de las soluciones prometedoras [PELI99]. De acuerdo con los resultados reportados, el comportamiento de estos algoritmos es similar al presentado en esta tesis. Estos sistemas son conocidos como *Population-Based Incremental Learning* (PBIL) [BALU94] y *Compact Genetic Algorithm* (CGA) [HARI99b]. Las principales diferencias radican en el momento y la manera en que se actualiza el vector de probabilidad.

PBIL actualiza el vector de probabilidad al término de cada generación. Basado exclusivamente en la información que proporciona el mejor individuo, aplica una regla Hebbiana de aprendizaje [HERT91]:

$$P_{nueva} = P_{vieja} * (1 - LR) + b * LR,$$

donde LR representa un factor de aprendizaje (*learning rate* en inglés) y
b es el valor del bit analizado.

Por el contrario, CGA actualiza el vector con cada nuevo individuo por lo cual no requiere almacenar a la población completa. El valor de actualización se calcula de acuerdo con el aporte del individuo a la frecuencia total suponiendo un tamaño de población específico:

$$P_{nueva} = P_{vieja} \pm \frac{1}{n},$$

donde n es el tamaño de la población y la operación suma o resta depende si el esquema es favorable o no.

Como podemos observar, a diferencia de STA, ambos algoritmos requieren la definición previa de un parámetro que determina el factor de aprendizaje. Un valor muy grande sobreexplotará la información parcial obtenida en cada iteración y provocará una convergencia prematura; un valor muy pequeño producirá un consumo innecesario de recursos computacionales. El valor óptimo de este parámetro depende de cada problema. Su definición es empírica y requiere de conocimiento previo del problema.

En los algoritmos de estimación de distribución [MÜHL96] no es claro el efecto que tiene el número de individuos que interviene en la actualización del vector de probabilidad, el método de actualización o el momento de actualización sobre el comportamiento global del sistema.

Es posible generalizar el Algoritmo Genético Estadístico para incluir los conceptos aportados por otros sistemas y facilitar el estudio futuro de estos métodos de actualización. Otro estudio interesante consistiría en utilizar diferentes funciones de actualización para cada bit. En la siguiente figura se muestra la generalización del STA.

INICIALIZACIÓN

Sea Π la población actual, N el número de individuos i en la población, m la probabilidad de mutación, l la longitud de la cadena genética. Sea f la función de evaluación, g el número de generación actual y P el vector de probabilidad. Sea nv el número de individuos que intervienen en la actualización del vector de probabilidad, LR el factor de aprendizaje y g_k una o varias funciones de actualización para cada bit.

1. Sea g = 0 y $P_k = 0.5$ para toda k tal que $1 <= k <= l$

GENERACIÓN

2. Repita N veces:
   2.1. Para cada j desde 1 hasta l:
      2.1.1. Genere dos valores aleatorios $0 <= \rho_j, \rho_k < 1$ de una distribución uniforme
      2.1.2. Si $\rho_j > m$:
         $b_j = 1$ si $\rho_k <= P_j$, 0 de otro modo
         sino **mute** estadísticamente:
         $b_j = 0$ si $\rho_k <= P_j$, 1 de otro modo
   2.2. Agregue un individuo compuesto por la concatenación de los l bits generados en el paso anterior a una nueva población Γ
3. Reemplace Π con los individuos de Γ

EVALUACIÓN

4. Para cada individuo i de la población actual Π.
   4.1. **Decodifique** el valor propuesto de las variables.
   4.2. **Evalúe** al individuo de acuerdo con sus variables y la función f. Llame a este valor de evaluación e.

EVOLUCIÓN

5. **Desplace** los valores e de evaluación de las N cadenas a un rango de números positivos, según la siguiente fórmula:

$$e'_i = e_i - mín(e_i) \qquad \text{para maximizar o}$$

$$e'_i = máx(e_i) - e_i \qquad \text{para minimizar}$$

6. **Normalice** los valores obtenidos entre cero y uno: $\Phi_i = \frac{e'_i}{\sum e'_i}$

7. **Ordene** los individuos de Π de acuerdo con Φ_i

ACTUALIZACIÓN

8. **Actualice** los valores del vector de probabilidad según se indica a continuación:
   8.1. Para cada i desde 1 hasta nv
      8.1.1. Para cada k desde 1 hasta l

$$P_k = g_k(P_k, b_{ik}, \text{LR})$$

CONDICIÓN DE TERMINACIÓN

9. Verifique si se ha cumplido alguna condición de **terminación**. En caso afirmativo, vaya al paso 11.

10. Incremente el número g de generación actual. Vaya al paso 2.

RESULTADO

11. Almacene el resultado.
12. Continúe con la fase del escalador.

Figura 40: Generalización del STA.

De acuerdo con esta generalización, es posible modificar los parámetros para acercarnos a uno u otro algoritmo, de la siguiente forma:

	PBIL	CGA	STA
n	Tam. Población	2	Tam. Población
nv	1	1	Tam. Población
LR	Definible	1/Tam. Población	Φ_i
$g_k(P_k, b_{ik}, LR)$	$P_k*(1-LR) + b_{ik}*LR$	$P_k + (2*b_{ik} - 1) * LR$	Si i=1, $b_{ik}*LR$ Sino $P_k+b_{ik}*LR$

6.3 Modelo Probabilístico

El modelo del vector de probabilidad utilizado en estos algoritmos asume que no existe interacción entre las variables. Por ello, es muy eficiente en problemas lineales y tienen un comportamiento pobre en funciones con interacciones fuertes entre las variables.

El vector de probabilidad pretende responder para cada bit la pregunta ¿qué es mejor en esta posición, un cero o un uno? Sin embargo, en ocasiones no es posible dar una respuesta concisa, sino un "depende". Veamos el siguiente ejemplo:

Supóngase una función que toma los siguientes valores:

B1	B0	$f(x)$
0	0	1
0	1	2
1	0	6
1	1	4

En el caso de B1, es posible determinar de acuerdo con los datos y suponiendo un problema de maximización que es mejor un uno que un cero para esa posición, dado que para cualquier valor de B0 se cumple que:

$$f(1,*) > f(0,*), \quad \text{donde } * \text{ implica cualquier valor.}$$

Para B0 no es posible llegar a una solución similar ya que el valor de la función es mayor o menor dependiendo no sólo de B0 sino también de B1:

$$f(0,1) > f(0,0) \quad \text{pero} \quad f(1,0) > f(1,1)$$

En casos como éste resultaría conveniente utilizar un modelo de la distribución de probabilidad que considere estas dependencias. Por ejemplo, en vez de utilizar el vector:

P(1): | 0.7 | 0.2 | 0.1 | 0.6 | y su contraparte P(0): | 0.3 | 0.8 | 0.9 | 0.4 |

podríamos definir la matriz de probabilidad:

P(00):	0.3*0.8	0.9*0.4	=	0.24	0.36
P(01):	0.3*0.2	0.9*0.6	=	0.06	0.54
P(10):	0.7*0.8	0.1*0.4	=	0.56	0.04
P(11):	0.7*0.2	0.1*0.6	=	0.14	0.06

suponiendo una interacción entre parejas adyacentes. De la misma manera, podríamos asumir una dependencia entre cuartetos o cualquier otra estructura no necesariamente entre variables adyacentes:

P(0000):	0.3*0.8*0.9*0.4	=	0.0864
P(0001):	0.3*0.8*0.9*0.6	=	0.1296
P(0010):	0.3*0.8*0.1*0.4	=	0.0096
P(0011):	0.3*0.8*0.1*0.6	=	0.0144
P(0100):	0.3*0.2*0.9*0.4	=	0.0216
P(0101):	0.3*0.2*0.9*0.6	=	0.0324
P(0110):	0.3*0.2*0.1*0.4	=	0.0024
P(0111):	0.3*0.2*0.1*0.6	=	0.0036
P(1000):	0.7*0.8*0.9*0.4	=	0.2016
P(1001):	0.7*0.8*0.9*0.6	=	0.3024
P(1010):	0.7*0.8*0.1*0.4	=	0.0224
P(1011):	0.7*0.8*0.1*0.6	=	0.0336
P(1100):	0.7*0.2*0.9*0.4	=	0.0504
P(1101):	0.7*0.2*0.9*0.6	=	0.0756
P(1110):	0.7*0.2*0.1*0.4	=	0.0056
P(1111):	0.7*0.2*0.1*0.6	=	0.0084

El estudio y la modificación del Algoritmo Genético Estadístico que permita el manejo adecuado de estas interacciones forma líneas de investigación futura.

7 Conclusiones

En esta tesis se estudió un método novedoso de evolución estadística mediante el diseño y análisis de comportamiento del Algoritmo Genético Estadístico (STA). El método propone el uso de un vector de distribución de probabilidad como modelo de la población genética. Este vector puede considerarse como una descripción difusa de las características que hacen que un individuo sea mejor evaluado. Debido a ello, es posible utilizar al vector de distribución como guía en la búsqueda durante la creación de nuevos individuos.

El Algoritmo Genético Estadístico consta de dos fases: la genética que resulta de la aplicación del modelo de probabilidad y la fase del escalador, utilizada para detallar los valores obtenidos en la etapa anterior. Durante el desarrollo de esta tesis se probó el sistema sobre un conjunto de funciones ampliamente utilizadas en el campo de la optimización numérica.

Posteriormente al análisis de resultados, se encontró que el comportamiento obtenido por la fase genética del STA es equiparable al del Algoritmo Genético Simple con cruza uniforme en términos del valor mínimo y promedio de cada generación. La demostración teórica de estos resultados experimentales queda como línea de trabajo futuro.

A pesar de haber obtenido valores de optimización similares en ambos algoritmos, se encontró que el uso de un modelo probabilístico de la población presenta cualidades interesantes y ciertas ventajas en comparación con los métodos evolutivos tradicionales.

El STA no utiliza los operadores de selección y cruza, simplemente simula su comportamiento. Debido a ello, evita la definición heurística de los parámetros de estos operadores por parte del usuario. Esta situación simplifica el uso del sistema y permite que usuarios menos experimentados puedan obtener resultados adecuados.

Otra característica del uso del vector de distribución de probabilidad es relevante en problemas con tamaños de población grande y genomas largos. Dado que el vector modela la población, la cantidad de memoria utilizada puede reducirse considerablemente. Un cálculo anticipado de los componentes del vector que se utilizará en la siguiente generación permite desechar el almacén poblacional. El costo en tiempo de proceso es ligeramente mayor que la versión sin optimización de memoria, manteniéndose directamente proporcional al número de individuos.

A diferencia de los métodos tradicionales, el Algoritmo Genético Estadístico facilita el estudio del comportamiento del sistema, ya que se cuenta con un modelo explícito de la población en cada generación. Esta ventaja podría simplificar el estudio matemático del proceso evolutivo en desarrollos posteriores.

En el estudio de esta tesis, se encontró que el método de actualización del vector de probabilidad propuesto por el Algoritmo Genético Estadístico tiene una tasa de aprendizaje decreciente observable en las gráficas del porcentaje de convergencia y es comprensible dada la naturaleza estadística del sistema. Esta característica provoque que la fase genética, por sí misma, no garantice la convergencia sin desperdicio de recursos computacionales.

Se encontró, además, que debido a que el vector de probabilidad asume que no existe interacción entre las variables, el algoritmo es muy eficiente en problemas lineales y tiene un comportamiento pobre en funciones con interacciones fuertes entre las variables. Una situación similar ocurre en sistemas con desarrollo paralelo a esta tesis. La generalización del STA permite introducir algunos conceptos propuestos por estos desarrollos.

Esta tesis introdujo ideas que definen nuevas e importantes ramificaciones en el diseño de los algoritmos genéticos. De esta forma, se señalan líneas de investigación que llevarán al desarrollo de sistemas más efectivos y eficientes.

8 Referencias

[ANGE95] Angeline P.J., "Adaptive and self-adaptive evolutionary computation", Computational Intelligence: A Dynamic System Perspective, Palaniswami, M., Attikiouzel Y., Marks R.J, Fogel D. Fukuda T., (eds.) NY, IEEE Press, pp. 152-161, 1995.

[BÄCK92] Bäck T., "The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm", Proceedings of the 2nd Conf. Parallel Problem Solving from Nature, R. Mäner and B. Manderick, (eds.), Amsterdam, North-Holland, pp. 85-94,1992.

[BÄCK96] Bäck T., "Evolutionary Algorithms in Theory and Practice",Oxford University Press, 1996.

[BALU94] Baluja S., "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning", Report No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, P.A., 1994.

[BURK84] Burkhard R.E., "Quadratic Assignment Problems", European Journal of Operations Research, 15, pp. 283-289,1984.

[COLO91] Colorni A., Dorigo M., Maniezzo V., "Distributed Optimization by Ant Colonies", Proceedings of the First European Conference on Artificial Life, Paris, France, Varela F. and Bourgine P., (eds.), Elsevier Publishing, pp. 134-142, 1991.

[COLO93] Colorni A., Dorigo M., Maniezzo, Trubian M., "Ant system for Job-Shop Scheduling", JORBEL – Belgian Journal of Operations Research, Statistics and Computer Science, 34, 1, pp. 39-53, 1993.

[DAVI91] Davis L., "Handbook of Genetic Algorithms", New York: Van Nostrand Reinhold, 1991.

[DEBK92] Deb K., Clark J.H., Goldberg D.E., "Genetic Algorithms, Noise and the Sizing of Populations", Complex Systems, Vol. 6, pp. 333-362.

[DENE83] Deneubourg J.L., Pasteels J.M., Verhaeghe J.C., "Probabilistic Behaviour in Ants: a Strategy of Errors?", Journal of Theoretical Biology, 105, pp. 259-271, 1983.

[DENE89] Deneubourg J.L., Goss S., "Collective patterns and decision-making", Ethology, Ecology & Evolution,Vol. 1, pp. 295-311, 1989.

[DEIT95] De Ita Luna, G., Morales-Luna G., "Algoritmos Genéticos para el Problema SAT", Memorias de la XII Reunión Nacional de Inteligencia Artificial", Cuernavaca, MX, Septiembre, 1995.

[DEJO75] De Jong, K.A., "An analysis of the behavior of a class of genetic adaptive systems", Tesis doctoral,University of Michigan, Ann Arbor, (University Microfilms No. 76-9381).

[DORI96] Dorigo M., Maniezzo V., Colorni A., "The Ant System: Optimization by a colony of cooperating agents", IEEE Transactions on Systems, Man and Cybernetics-Part B, Vol. 26, No. 1, pp. 1-13, 1996.

[DORI97] Dorigo M., Gambardella L.C., "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 53-66, Abril 1997.

[EIBE99] Eiben, A.E., Hinterding, R., Michalewicz, Z., "Parameter Control in Evolutionary Algorithms", IEEE Transactions on Evolutionary Computation, Vol. 3, No.2, pp.124-141, Julio 1999.

[GOLD89] Goldberg, D.E., "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.

[GOLD91] Goldberg, D.E., Rudnick M., "Genetic Algorithms and the Variance of Fitness", Computer Systems, Vol. 5, No. 3, pp. 265-278, 1991.

[GOLD92] Goldberg, D.E., Deb, K., Clark, J.H., "Genetic Algorithms, noise and the sizing of populations.", Complex Systems, Vol. 6, pp. 333-362, 1992.

[GOLD93] Goldberg, D.E., Deb, K., Thierens, D. "Toward a better understanding of mixing in genetic algorithms", Journal of the Society of Instrument and Control Engineers, pp.10-16.

[GOSS90] Goss S., Beckers R., Deneubourg J.L., Aron S., Pasteels J.M., "How Trail Laying and Trail Following Can Solve Foraging Problems for Ant Colonies", in Behavioural Mechanisms of Food Selection, Hughes R.N. (ed.), NATO-ASI Series, vol. G 20, Berlin: Springer-Verlag, 1990.

[GREF86] Grefenstette J.J., "Optimization of control parameters for genetic algorithms", IEEE Transactions Systems, Man, Cybern., Vol. 16, No. 1, pp. 122-128, 1986.

[HAMM80] Hamming R., "Coding and Information Theory", Prentice-Hall, 1980.

[HARI97] Harik G., Cantú Paz E., Goldberg D.E., Miller B., "The Gambler's Ruin Problem, Genetic Algorithms and the Sizing of Populations", Proceedings of the Fourth International Conference on Evolutionary Computation, New York, IEEE Press, pp. 7-12, 1997.

[HARI99a] Harik G., Lobo, F., "A parameter-less genetic algorithm", Illinois Genetic Algorithms Lab., University of Illinois at Urbana-Champaign, Report No. 99009, Enero 1999.

[HARI99b] Harik G., Lobo F., Goldberg D.E., "The Compact Genetic Algorithm", IEEE Transactions on Evolutionary Computation, Vol. 3, No. 4, pp. 287-297, Noviembre 1999.

[HERT91] Krogh J.A., Palmer R.G., "Introduction to the theory of neural computation", Reading, MA: Addison-Wesley, 1991.

[HINT97a] Hinterding R., "Self-adaptation using multi-chromosomes",Proceedings 4th Conf. Evolutionary Computation, Piscataway, NJ, pp. 87-91, 1997.

[HINT97b] Hinterding R., Michalewicz Z., Eiben E., "Adaptation in evolutionary computation: A survey", Proceedings 4th Conf. Evolutionary Computation, Piscataway, NJ, pp. 65-69, 1997.

[HOLL75] Holland, J., "Adaptation in Natural and Artificial Systems",Ann Arbor, MI; University of Michigan Press, 1975.

[KNUT69] Knuth, D., "The Art of Computer Programming" Vol 1, Seminumerical Algorithms, Addison-Wesley, 1969.

[KURI99a] Kuri Morales, A., "A Comprehensive Approach to Genetic Algorithms in Optimization and Learning: Theory and Applications", Centro de Investigación en Computación, Instituto Politécnico Nacional, Vol. I, pp.160-163, 1999.

[KURI99b] Kuri Morales, A., "A Statistical Genetic Algorithm", Memorias del Segundo Encuentro Nacional de Computación, Hidalgo, México, 1999.

[LAWL85] Lawler E.L., Lenstra J.K., Rinnooy-Kan A.H.G., Shmoys D.B. eds., "The Traveling Salesman Problem", New York: Wiley, 1985.

[MITC96] Mitchell, M., "Introduction to Genetic Algorithms", MIT Press, 1996.

[MÜHL92] Mühlenbein, H. "How Genetic Algorithms Really Work: Mutation and Hill climbing", R. Manner and B. Manderick (eds.), Parallel problem solving from Nature 2, North-Holland, 1992.

[MÜHL96] Mühlenbein, H., Paaβ G., "From recombination of genes to the estimation of distributions". pp. 178-187, 1996.

[PELI99] Pelikan, M., Goldberg, D.E., Lobo, F., "A Survey of Optimization by Building and Using Probabilistic Models", Illinois Genetic Algorithms Lab., University of Illinois at Urbana-Champaign, Report No. 99018, Septiembre 1999.

[PINT99] Pinto Elías, R., Sossa Azuela, J.H., "Algoritmos Genéticos para el Reconocimiento de Patrones", Memorias del Taller de Inteligencia Artificial TAINA 99, DF, MX, Octubre 1999.

[RUDO97] Rudolph, G., "Convergence Analysis of Canonical Genetic Algorithms", IEEE Transactions on Neural Networks, Enero 1994.

[SCHA89] Schaffer, J., Caruna, R., Eshelman, L., Das, R.,"A study of control parameters affecting online performance of genetic algorithms for function optimization", Proceedings of the 3rd International Conference on Genetic Algorithms and Their Applications. Morgan Kaufman, 1989.

[SPEA91] Spears, W., De Jong K., "An Analysis of Multi-Point Crossover", Rawlins, G., editor, Foundations of Genetic Algorithms, pp. 301-315, Morgan Kauffman, 1991.

[VOSE96] Vose, D., "The Walsh Transform and the Theory of the Simple Genetic Algorithm". Pal, S. And Wang, P. (eds.), Genetic Algorithms for Pattern Recognition, CRC Press, 1996.

Apéndice A: Sistema de Hormigas

El Sistema de Hormigas (SH), como su nombre lo indica, fue inspirado en la manera en la cual trabajan y se comunican las colonias de hormigas (véase [DENE83],[DENE89],[GOSS90]). Se trata de un sistema de optimización combinatoria estocástica, que presentaron Marco Dorigo, Vittorio Maniezzo y Alberto Colorni, por primera vez, en la Primer Conferencia Europea de Inteligencia Artificial realizada en Paris en 1991[COLO91].

Entre las principales características del SH se encuentra la retroalimentación positiva (*positive feedback*, en inglés), el cómputo distribuido y el uso de una heurística constructiva (*constructive greedy heuristic*, en inglés). La retroalimentación sirve para el pronto descubrimiento de soluciones buenas, el cómputo distribuido evita la convergencia prematura[32] y la heurística ayuda a encontrar soluciones aceptables en las primeras etapas del proceso de búsqueda.

En 1997, Marco Dorigo y Luca M. Gambardella modificaron el algoritmo para realizar búsquedas más eficientes en problemas de mayor tamaño y con ello crearon el Sistema Colonia de Hormigas [DORI97]. Al igual que con los Algoritmos Genéticos, se considerará únicamente la versión original (i.e. Sistema de Hormigas) para efectos de esta tesis.

En la siguiente sección, retomaré un ejemplo presentado por los autores en [DORI96] para dar al lector una intuición general del funcionamiento del sistema. Posteriormente, explicaré la representación del problema utilizada en el SH para después describir los mecanismos de solución del sistema, definiendo sus parámetros y funciones. Finalmente presentaré el pseudo-código que define al algoritmo.

Analogía

Uno de los hechos más notables en la naturaleza es la manera como se comunican los animales entre sí, especialmente los insectos. El entomólogo Edward O. Wilson escribió: "Las formas de comunicación entre los insectos sociales son extrañamente diversas: toquecitos, estridulaciones, golpecillos, asimientos, rozamientos de antenas, gustos o probadas y vaharadas, y emisiones de substancias químicas que evocan respuestas diversas"[33]. En este caso, estamos interesados en las señales químicas – exudaciones, que se conocen con el nombre de feromonas, que se gustan o huelen.

Las feromonas son el medio que permite a las hormigas, animales casi ciegos, encontrar la ruta más corta desde su hormiguero hasta su fuente de alimento. Cada hormiga deposita en su caminar cantidades variables de feromonas, marcando así la ruta seguida. Esta señal sirve a la hormiga para reconocer su vía de regreso, pero más importante, indica a la colonia un camino factible. Al encontrarse un camino previamente recorrido por una hormiga, detecta la marca y decide con mayor probabilidad seguir dicha ruta. De la misma manera que la hormiga anterior, marca y refuerza el camino con feromona. Entre más hormigas sigan una trayectoria, ésta se convierte más atractiva para las hormigas que siguen.

Se trata, entonces, de un proceso cíclico de retroalimentación, donde la probabilidad con la cual cada hormiga selecciona un camino se incrementa proporcionalmente al número de hormigas que hayan elegido esa ruta previamente.

[32] La convergencia, como fue explicado con anterioridad, es una etapa de estancamiento donde el algoritmo no puede explotar adecuadamente su información. La convergencia prematura ocurre cuando dicho estado aparece en etapas tempranas del algoritmo y por lo tanto, produce resultados erróneos.

[33] Tomado del libro "Los Insectos", Colección de la Naturaleza de Time-Life por Peter Farb, versión en español, 1981.

Considérese el ejemplo de la Figura 41. Existe una trayectoria que siguen las hormigas entre los puntos A y E. Estos puntos pueden ser la fuente de alimento y el hormiguero; no importa el orden (véase Figura 41a). Repentinamente, se coloca un obstáculo a media ruta. Dicho obstáculo es perpendicular a la trayectoria AE y se ubica de tal manera que un extremo C esté más cercano al camino original que el otro extremo H (véase Figura 41b). El obstáculo recién colocado fuerza a las hormigas a tomar una decisión: girar a la derecha o a la izquierda. Una de estas opciones las llevará a rodear el obstáculo por el camino más corto – a través del extremo C.

La primer hormiga en encontrarse con la barrera elegirá al azar cualquiera de los caminos, ya que no hay señales químicas previas. Una hormiga que camine por BCD llegará primero al punto D que una que tome el rumbo BHD, debido a que la distancia es menor (véase Figura 41c). Esto ocasiona que la siguiente hormiga que regrese de E, al llegar al punto D, encuentre mayor cantidad de feromona en el camino DCB que en DHB. Entonces, la probabilidad de que esta hormiga elija la ruta más corta se incrementa. Lo mismo ocurre en el sentido contrario. Como consecuencia, el número de hormigas por unidad de tiempo que andan por BCD será mayor que el de BHD. La cantidad de substancia química en el camino corto crecerá más rápidamente y la probabilidad de seleccionar BCD será mayor. Eventualmente, todas las hormigas recorrerán ABCDE y regresarán por EDCBA.

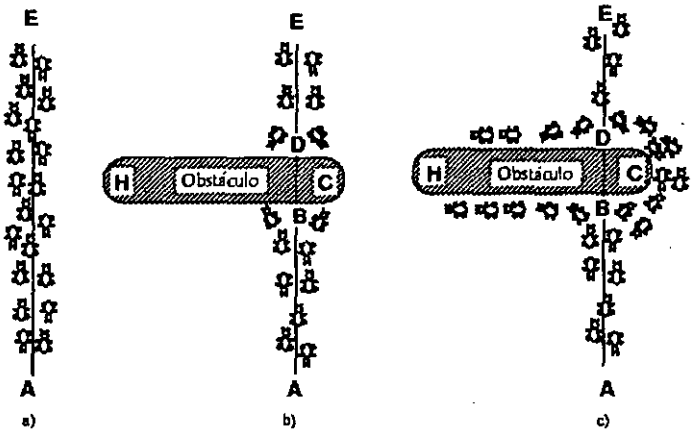


Figura 41: Hormigas Reales, a) Ruta AE b) Obstáculo HC a medio camino, fuerza una decisión c) Mayor cantidad de feromona en la ruta más corta.

Representación

El Sistema de Hormigas se concentra primordialmente en problemas de optimización combinatoria. Es decir, problemas en los cuales se debe determinar cuál de las posibles combinaciones finitas de las variables minimiza o maximiza cierta función característica. Algunos ejemplos bien conocidos por los investigadores de las ciencias de la Computación son: el problema del Agente Viajero (*Traveling Salesman problem*), el problema de la Asignación Cuadrática de Recursos (*Quadratic Assignment problem*) y el problema de la Calendarización de Tareas (*Job-Shop Scheduling problem*)[34].

[34] Para conocer la descripción de estos problemas véase [LAW85],[BURK84] y [COLOR93] respectivamente.

El SH representa al espacio de búsqueda del problema por medio de un grafo (i.e. nodos y arcos) con conexiones ponderadas. Los nodos representan los valores discretos de las variables del sistema, mientras que los arcos describen a las posibles combinaciones de estas variables. Además, los arcos cuentan con una propiedad de peso que afecta en la evaluación de cada combinación.

Una representación adecuada para la analogía presentada en la sección anterior es la que se muestra en la Figura 42. Los valores de ponderación de los arcos fueron elegidos arbitrariamente, respetando únicamente la restricción BH + HD > BC + CD.

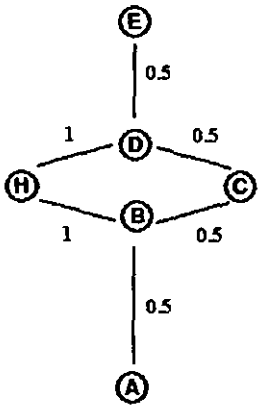


Figura 42: Representación Gráfica del Ejemplo Analogía.

En la metodología propuesta, las hormigas son las entidades encargadas de realizar la búsqueda propiamente dicha. Estos agentes con capacidades limitadas recorren el grafo desde un nodo inicial (que puede ser distinto para cada hormiga) hasta algún nodo final. El trayecto recorrido denota una propuesta de solución – una combinación específica de nodos.

En este tipo de problemas suelen existir restricciones que no pueden representarse en la gráfica. Por ello, es necesaria una estructura de datos adicional para asegurar la satisfacción de estas reglas. La estructura a utilizar y el proceso de validación de restricciones pueden variar entre diferentes problemas.

En la siguiente sección, se describen los mecanismos de solución del SH en el contexto del problema del Agente Viajero; se definen los parámetros del sistema, así como algunas funciones utilizadas en el proceso.

Descripción

Los autores del artículo "*The Ant System: Optimization by a colony of cooperating agents*" basan la definición del SH de acuerdo con la estructura del problema del Agente Viajero[35]. Eligieron este problema porque es suficientemente descriptivo y el mecanismo descrito en la analogía de las hormigas es aplicable directamente.

[35] Desde mi punto de vista, la influencia del problema del Agente Viajero en la definición del Sistema de Hormigas facilita la comprensión del sistema, pero reduce su generalidad ya que cada problema nuevo impone modificaciones mayores en la programación del algoritmo.

El problema consiste en encontrar la ruta más corta para que el agente pueda visitar un total de n ciudades y regresar al punto de partida. En este ejemplo, los nodos representan a las ciudades, los arcos simbolizan las carreteras que unen a las ciudades, y los pesos de los arcos son los kilómetros que mide la carretera.

El Sistema de Hormigas cuenta con un ciclo peculiar de generación y explotación de la información. Inicialmente se ubica a cada hormiga en un nodo o ciudad cualquiera. Estos agentes deciden cual será el siguiente nodo a visitar de acuerdo con una probabilidad calculada a partir de un rastro de feromona en el trayecto y de un valor heurístico específico del problema. Cuando las hormigas terminan de recorrer todas las ciudades, comienza la etapa de explotación de la información obtenida. En este momento, se actualizan los valores de las marcas químicas dejadas en todas las rutas andadas. Es decir, las hormigas modifican la cantidad de feromona exclusivamente al finalizar su tour. Veamos con mayor detalle este procedimiento.

Sea $\tau_{ij}(t)$ la intensidad de la feromona en el camino (i,j) en el tiempo t. La actualización ocurre en t+n debido a que toma n tiempos recorrer n ciudades. De esta forma:

$$\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \Delta\tau_{ij}$$

donde ρ es un coeficiente de evaporación de la feromona entre los tiempos t y t+n ($0<\rho<1$), y $\Delta\tau_{ij}$ es el refuerzo ocasionado por el paso de las hormigas en el arco (i,j) calculado:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

tal que m es la cantidad total de hormigas, y $\Delta\tau_{ij}^{k}$ es la cantidad de substancia dejada por la k-ésima hormiga entre los tiempos especificados y está dado por:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{si la hormiga k usó el camino (i, j)} \\ 0 & \text{de otro modo} \end{cases}$$

En la ecuación anterior, Q es una constante definida de antemano y L_k es la evaluación de la solución k propuesta, en este caso, la longitud del recorrido total de la hormiga k.

Como se dijo anteriormente, la probabilidad de que una hormiga elija una ruta depende en parte de la intensidad de feromona en el camino. La otra parte corresponde a una heurística definida para el problema. En este caso, la heurística dice que siempre se prefieren las menores distancias. Si d_{ij} es la distancia entre el nodo i y el nodo j (i.e. d_{ij} es el peso del arco), entonces podemos definir un valor η_{ij} como el inverso de la distancia y nombrarlo visibilidad.

$$\eta_{ij} = \frac{1}{d_{ij}}$$

De esta manera, la probabilidad se calcula con la siguiente fórmula:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k \in \{N - tabu_k\}} ([\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}]^{\beta})} & \text{si } j \in \{N - tabu_k\} \\ 0 & \text{de otro modo} \end{cases}$$

donde α y β son parámetros que controlan la influencia de la feromona con relación a la heurística. Los autores reportan que de acuerdo con sus experimentos, los mejores valores para los

parámetros son: α=1, β=5, ρ=0.5, Q = cualquier valor. De acuerdo con mi experiencia, α y β podrían ser valores dinámicos, es decir, que se pueden modificar conforme avanza el algoritmo. Esta opción no ha sido investigada.

Terminación

Al igual que ocurre con un Algoritmo Genético, es preciso definir una condición de terminación del algoritmo. Ésta puede ser cualquiera de las descritas en la Sección 1.4.5. Los autores utilizan un número máximo de ciclos (NC_{max}).

Un hecho notable es que en este algoritmo se puede producir un comportamiento que hace referencia a la convergencia en los algoritmos genéticos. En la terminología del Sistema de Hormigas se le conoce como estancamiento. Se produce cuando todas las hormigas realizan el mismo recorrido. En este momento, el algoritmo deja de buscar soluciones alternas. Es también una condición de terminación.

Pseudo-Código

El lector habrá podido darse cuenta del parecido que tiene el SH con el SGA. Son precisamente estas similitudes las que permitieron la realización de esta tesis. Para reforzar aun más estas ideas, presento en la siguiente figura el pseudo-código correspondiente al SH habiendo titulado las etapas del ciclo generación-explotación.

INICIALIZACIÓN

1. Sea t = 0 (contador de tiempo)
2. Sea NC=0 (contador de ciclos)
3. Para cada arista (i,j):
   3.1. Sea $\tau_{ij} = c$ donde c es una constante que define la intensidad de feromona en el camino (i,j)
   3.2. Sea $\Delta_{ij} = 0$
4. Ubique cada una de las m hormigas en alguno de los n nodos.

INICIALIZACIÓN DE RESTRICCIONES

5. Sea s=1.
   5.1. Para cada hormiga k numerada desde 1 hasta m:
      5.1.1. Agregue el nodo inicial de la hormiga k a la lista $tabu_k(s)$

CREACIÓN DE SOLUCIONES

6. Repita hasta llenar la lista tabu. (n-1 veces)
   6.1. Sea s=s+1.
   6.2. Para cada hormiga k numerada desde 1 hasta m:
      6.2.1. Elija el nodo destino j con probabilidad $p_{ij}^k(t)$, donde:

$$p_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{k \in \{N - tabu_k\}} ([\tau_{ik}(t)]^\alpha \cdot [\eta_{ik}]^\beta)} & \text{si } j \in \{N - tabu_k\} \\ 0 \quad \text{de otro modo} \end{cases}$$

   6.3. Mueva la hormiga k-ésima al nodo j.
   6.4. Inserte el nodo j en la lista $tabu_k(s)$

EVALUACIÓN

7. Para cada hormiga k numerada desde 1 hasta m:
   7.1. Calcule la longitud L_k de la ruta descrita por $tabu_k$
   7.2. Actualice la ruta más corta

EXPLOTACIÓN

8. Para cada arista (i,j):
   8.1. Para cada k desde 1 hasta m:

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{Q}{L_k} & \text{si } (i,j) \in tabu_k \\ 0 \quad \text{de otro modo} \end{cases}$$

$$\Delta\tau_{ij}^k = \Delta\tau_{ij} + \Delta\tau_{ij}^k$$

9. Para cada arista (i,j):
   9.1. $\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \Delta\tau_{ij}$
10. Sea t = t+n
11. Sea NC = NC + 1
12. Para cada arista (i,j): $\Delta\tau_{ij} = 0$

TERMINACIÓN

13. Si ($NC < NC_{max}$) y (no hay estancamiento):
   13.1. Vacíe todas las listas tabu.
   13.2. Vaya al paso 5.
14. de lo contrario:
   14.1. Muestre el resultado.