UNIVERSIDAD DE BELGRANO

Las tesinas de Belgrano

Facultad de Ingeniería y Tecnología Informática
Licenciatura en Sistemas

Una aplicación de programación genética al área de control.

Nº 203 **Santiago Luis Pettis**

Tutor: Raimundo O. D'Aquila

Departamento de Investigación
Noviembre 2005

Resumen

La generación de una estrategia de control óptimo para controlar adecuadamente un determinado sistema es usualmente una tarea compleja. Este trabajo propone generar automáticamente una estrategia de control para un problema específico del área de control haciendo uso de técnicas que simulan los principios que rigen la evolución de los seres vivos en la naturaleza. El problema mencionado se refiere al centrado de un carro en movimiento. Las técnicas mencionadas provienen de la computación evolutiva, más precisamente del campo de la programación genética.

Los resultados obtenidos demuestran que, a través de una adecuada parametrización del algoritmo que implementa el paradigma de la programación genética, es posible la obtención automática de programas que constituyen estrategias de control subóptimas para el problema mencionado, sin que en este proceso intervenga la inteligencia humana.

Palabras clave: inteligencia artificial; sistemas inteligentes; computación evolutiva; programación genética; control óptimo.

Abstract

The generation of an optimal control strategy to adequately control a specified system is usually a complex task. This work proposes the automatic creation of a control strategy for a specified problem of the control area by means of techniques that simulate the principles that command evolution of living entities in nature. The mentioned problem refers to the cart centering problem. The mentioned techniques come from evolutionary computing, more precisely form the field of genetic programming.

The obtained results show that, by means of an adequate parametrization of the algorithm that implements the genetic programming paradigm, it is possible to automatically create programs that constitute suboptimal control strategies for the specified problem, without any intervention of human intelligence in this process.

Keywords: artificial intelligence; intelligent systems; evolutionary computing; genetic programming; optimal control.

Índice

1. Introducción

La computación evolutiva constituye un método de búsqueda y optimización estocástica, ciega y múltiple de posibles soluciones a un problema determinado. Este proceso es guiado por los principios que rigen la evolución natural. Combina los conceptos de adaptación y supervivencia de los más aptos para producir soluciones subóptimas al problema dado. Esto es logrado mediante la aplicación de operadores de reproducción y cruce entre los individuos de una población que representan soluciones potenciales.

La programación genética es un campo de la computación evolutiva en la que los individuos de la población son programas de computadora. Estos programas tratan de resolver un problema determinado. Se constituyen por la composición de un número de funciones que operan sobre un número de terminales.

Este trabajo propone el uso de programación genética para dar solución a un problema del área de control. Dicho problema trata acerca de la generación de una estrategia de control (en la forma de un programa de computadora) que permita centrar un carro que se encuentra en movimiento sobre un camino infinito, unidimensional y sin fricción.

El uso de la programación genética aplicada a este problema permite la obtención de estrategias de control subóptimas que representan una solución para dicho problema. La característica innovadora es que estas estrategias son programas de computadora generados automáticamente sin participación de inteligencia humana.

El trabajo está compuesta por las siguientes secciones:

· Estado del arte:

Aquí se hará una breve introducción a los fundamentos teóricos sobre los cuales se basa la computación evolutiva en general. Se nombrarán campos de aplicación y trabajos realizados en el campo de la computación evolutiva tradicional (algoritmos genéticos y programación evolutiva) y en el campo de la programación genética. Para el caso de esta última, se hará mención de dos problemas del área de control óptimo cuya solución fue alcanzada mediante programación genética.

· Definición del problema:

En esta sección se definirá, con máximo detalle, el problema de centrado del carro que este trabajo propone resolver.

· Programación genética:

Aquí se explicarán, en detalle, los fundamentos teóricos sobre los cuales se basa la programación genética, a fin de comprender cómo opera la misma. Esta sección será dividida en dos subsecciones. Una de ellas se referirá al uso de la programación genética para la creación evolutiva de programas algorítmicos (este es el uso que se le dará a este paradigma en la solución al problema planteado en este trabajo); la otra subsección se referirá al uso de la programación genética para la creación evolutiva de programas de conocimientos.

· Solución propuesta:

En esta sección se explicará, en detalle, cómo fue utilizado el paradigma de la programación genética explicado en la sección anterior para dar solución al problema planteado.

· Resultados experimentales:

Se analizarán en profundidad los resultados que fueron obtenidos haciendo uso de la solución propuesta.

· Conclusiones y futuras líneas de investigación.

2. Estado del arte

2.1 Computación evolutiva

2.1.1 Introducción

La evolución se define como el proceso mediante el cual las generaciones descendientes se van adaptando mejor al entorno que sus progenitores. Por ejemplo, el desarrollo observado de la resistencia a los insecticidas en las plagas de cultivos, a los antibióticos en las bacterias, a la quimioterapia en las células cancerosas, y a los fármacos en virus como el VIH, es una consecuencia abierta del proceso de evolución [7].

La evolución a nivel biológico opera de la siguiente forma:

1. Se generan descendientes a imagen de sus progenitores, pero con algunas modificaciones y mediante un proceso de selección natural (los mejores adaptados tienen mayor probabilidad de sobrevivir).
2. Los descendientes supervivientes producen a su vez más descendientes, siguiendo siempre el lineamiento trazado en el punto 1.

Estos dos aspectos de la evolución (la modificación por medio de la reproducción y la selección natural) son suficientes para generar individuos que cada vez desarrollan más eficientemente las acciones que contribuyen a su habilidad reproductora.

La siguiente analogía con la geometría nos puede ayudar a comprender este proceso [8]. Imaginemos un «paisaje» matemático compuesto por montañas, valles y terrenos llanos que está poblado por una serie de individuos situados en forma aleatoria en el paisaje. La altura a la que se sitúa un individuo es una medida que indica lo bien que desarrolla su tarea en relación con el resto de los individuos. La probabilidad de que un individuo desaparezca es inversamente proporcional a su altura, mientras que la probabilidad de que un individuo se reproduzca es proporcional a ella. El proceso de reproducción consiste en la creación de un nuevo individuo cuya posición en el paisaje será distinta, aunque estará relacionada con la de sus progenitores. El tipo de reproducción más interesante y eficiente es aquella en la que, para generar un nuevo individuo, es necesario la cooperación de dos progenitores. De esta forma, la localización del nuevo descendiente se calculará en función de la localización de sus progenitores. En la mayoría de los casos, el proceso de evolución colocará al nuevo descendiente en algún lugar entre los progenitores, pero existirá una pequeña probabilidad de que el nuevo descendiente sea colocado en un lugar más alto que el de los padres.

Este proceso descrito en la analogía anterior puede ser considerado como un medio de encontrar las cimas de las montañas del paisaje. Cada nueva generación de individuos examina el terreno que le rodea. Aquellos individuos que se encuentren en zonas bajas morirán, ya que tienen una alta probabilidad de desaparecer. Al cabo de cierto tiempo, encontraremos algunos individuos situados en las cimas del paisaje. La eficiencia del proceso de exploración dependerá de cuan diferentes sean los individuos respecto a sus progenitores y de la orografía del paisaje. Algunos descendientes se colocarán a menor altura que sus progenitores pero, ocasionalmente, algunos podrán colocarse a más altura.

2.1.2 ¿Qué es la computación evolutiva?

El hombre ha llegado a utilizar con grandes resultados el principio de cría selectiva para crear organismos personalizados, distintos a cualquiera que se pueda encontrar en la naturaleza, para beneficio propio. En las últimas décadas, el continuo avance de la tecnología moderna ha producido algo nuevo. Ahora la evolución está produciendo beneficios prácticos en un campo que a priori puede parecer extraño, pero que en realidad no lo es. Este campo es la informática, a través de lo que se denomina «computación evolutiva» [7].

La computación evolutiva es la disciplina de la inteligencia artificial que hace uso de modelos computacionales de procesos naturales de evolución como base en el diseño e implementación de algoritmos para la resolución de problemas. Es una técnica de resolución de problemas inspirada en la evolución natural, orientada a la resolución de problemas de optimización, teniendo como objetivo entregar una solución cuasi-óptima: buena, aunque no necesariamente la mejor.

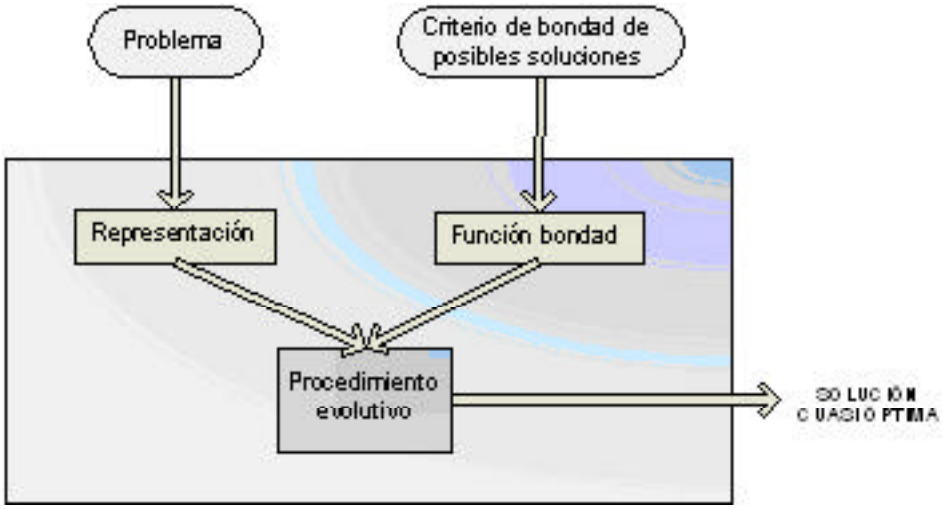

Figura 1. Esquema básico de la computación evolutiva.

2.1.3 Algoritmo evolutivo

El algoritmo evolutivo es aquel sobre el cual se asienta la computación evolutiva para simular el proceso biológico de evolución. Este algoritmo es común para cualquiera sea el problema que se esté intentando solucionar mediante la computación evolutiva. Aplica dos principios básicos de la evolución natural:

- Los más aptos tienen mayor probabilidad de descendencia y de supervivencia.
- Los malos de hoy pueden ser necesarios para generar los mejores de mañana.

En primer lugar, se genera una población de tamaño *n* compuesta por un conjunto aleatoriamente creado de *n* individuos. Cada individuo constituye un candidato a solución del problema y cada uno es representado en una estructura de datos mediante una codificación.

Cada individuo tiene una aptitud *(fitness)* asociada, la cual es determinada por la función bondad. En la naturaleza la función bondad está representada por el ambiente donde acciona el ser, el cual «mide» su adaptación al mismo. Es necesario que la distribución de aptitudes sea uniforme para evitar la convergencia prematura, es decir, para evitar que unos pocos individuos más o menos aptos en las poblaciones iniciales estanquen la búsqueda en un óptimo local. La función bondad, junto con la representación de los individuos, constituyen la única dedicación del procedimiento evolutivo al problema.

La aptitud de la totalidad de los individuos de la población es evaluada y se determina si se ha alcanzado cierta condición de parada. Si la misma ha sido alcanzada, se retorna el mejor individuo de la población (el cual puede representar la solución óptima o la solución cuasi-óptima al problema).

Si la condición de parada no ha sido alcanzada, se lleva a cabo el ciclo evolutivo, el cual dará como resultado una nueva generación de individuos que son descendientes de la generación actual. Cada ciclo evolutivo consiste en un proceso de selección (fase de selección) y de búsqueda (fase de reproducción, cruzamiento y mutación). Por último, se lleva a cabo una selección de supervivientes para que la población vuelva a tener *n* individuos.

El algoritmo itera siguiendo estos lineamientos, hasta que cierta condición de parada es alcanzada. El mismo puede ser ilustrado del siguiente modo:

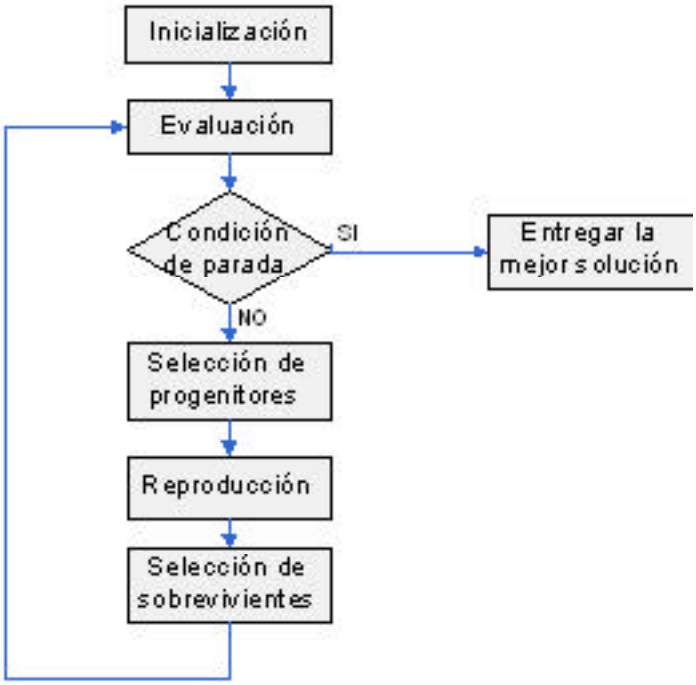

Figura 2. Diagrama de flujo del algoritmo evolutivo.

2.1.4 La computación evolutiva y las características de su proceso de búsqueda de posibles soluciones

La computación evolutiva está basada en una búsqueda ciega, estocástica y múltiple en el espacio de posibles soluciones a un problema dado. Es necesario comprender los siguientes conceptos elementales acerca de la teoría de búsqueda y optimización para comprender las características anteriormente nombradas de la búsqueda de posibles soluciones en la computación evolutiva.

El proceso de búsqueda y optimización puede ser descrito en base a distintos criterios de clasificación. Los que nos interesan aquí son:

1. Si tomamos como base la información que se encuentra disponible acerca de la función a optimizar, las búsquedas pueden ser:
   a. *Ciegas:* el proceso a optimizar funciona como una caja negra que ante ciertos valores de los parámetros devuelve un valor del objetivo. Por lo tanto no se dispone de ninguna información explícita sobre la aplicación f(x). A estos métodos de optimización se los denomina «optimización mediante cajas negras'». La ventaja de estos métodos es que proporcionan algoritmos de búsqueda de propósito general, los cuales son muy fáciles de implementar para un problema específico.
   b. *Heurísticas:* se dispone de cierta información explícita sobre el proceso a optimizar, la cual se puede aprovechar para guiar la búsqueda. A dicha información útil para la búsqueda se le llama «conocimiento específico». Las técnicas heurísticas proporcionan algoritmos dedicados de búsqueda, esto es algoritmos específicos para un problema concreto y difícilmente adaptables para cualquier otro. La búsqueda heurística implica utilizar la componente heurística para podar el espacio de búsqueda.
2. Si tomamos como base el grado de aleatoriedad que se le da al proceso de búsqueda, las búsquedas pueden ser:
   a. *Deterministas o dirigidas:* el procedimiento de búsqueda es completamente determinista. Esto significa que en las mismas condiciones de partida proporciona idénticos resultados.
   b. *Aleatorias o al azar:* el procedimiento de búsqueda es completamente aleatorio. Por lo general, se delimita una región de búsqueda y se toman puntos al azar dentro de ella.
   c. *Estocásticas u orientadas:* se combinan en proporción variable la búsqueda determinista con la búsqueda aleatoria. La componente determinista orienta la dirección de búsqueda y la aleatoria se encarga de la búsqueda local.

3. Si tomamos como base el número de candidatos a solución que se mantienen simultáneamente, las búsquedas pueden ser:
   a. *Simples:* se mantiene un solo candidato a solución que se va actualizando sucesivamente para proporcionar, presumiblemente, soluciones cada vez más exactas del problema.

b. Múltiples: se mantienen simultáneamente varios candidatos a solución con los cuales se va acotando cada vez con más precisión la región (o regiones) donde se encuentra el/los óptimo/s. Son las más apropiadas para implantaciones en paralelo.

La búsqueda de la computación evolutiva en el espacio de soluciones posibles es entonces:

- *ciega* porque no se hace uso de ninguna información para podar el espacio de búsqueda,
- *estocástica* porque se combina el determinismo provisto por el propio algoritmo que implementa el paradigma de la computación evolutiva, con la componente aleatoria determinada por la selección al azar de progenitores, de descendientes y de operadores,
- *múltiple* porque la población en cada generación está compuesta simultáneamente por varios individuos, cada uno de los cuales es un candidato a solución.

2.1.5 Modelos de computación evolutiva

Existe una variedad de propuestas sobre estos modelos, los cuales reciben el nombre genérico de algoritmos evolutivos. Estos tienen como característica común su inspiración en la simulación de la evolución de poblaciones de individuos (los cuales representan cada uno una solución a un problema) a través de procesos de selección y reproducción. Sobre el esquema general de los algoritmos evolutivos, los sistemas de computación evolutiva pueden clasificarse en:

- Algoritmos genéticos [4] [7]: en este modelo, cada individuo está constituido por una secuencia de binits.
- Programación evolutiva: es una generalización de los algoritmos genéticos. Se utilizan otras estructuras de datos que permitan lograr representaciones más eficientes de acuerdo al problema.
- Programación genética [1]: en este modelo, cada individuo es un programa de computadora escrito en un lenguaje determinado. Está usualmente constituido por secuencias de instrucciones del lenguaje.

Los primeros dos modelos (algoritmos genéticos y programación evolutiva) constituyen la denominada computación evolutiva tradicional. La programación genética, por su parte, es un concepto más avanzado y más reciente que nace sobre la base de los dos modelos anteriormente mencionados.

2.2 Computación evolutiva tradicional

Como ya se ha dicho anteriormente, los algoritmos genéticos y la programación evolutiva constituyen la denominada computación evolutiva tradicional (siendo la programación evolutiva simplemente una generalización de los algoritmos genéticos).

Es recomendable conocer cómo opera la computación evolutiva tradicional (en particular los algoritmos genéticos) antes de adentrarse en el mundo de la programación genética. Dado que este trabajo propone brindar una aplicación de programación genética para solucionar un problema del área de control, el estudio de la computación evolutiva tradicional escapa a su alcance. Por lo tanto, en caso que el lector no tenga conocimientos acerca de de la misma, se recomienda la lectura del apéndice «Algoritmos genéticos» antes de adentrarse en el tema de la programación genética. Aquel lector que posea un conocimiento básico acerca de los algoritmos genéticos, puede continuar tranquilamente la lectura de este trabajo.

2.2.1 Campos de aplicación y trabajos realizados

Algunos campos sobre los cuales se han realizado trabajos basados en la computación evolutiva tradicional son los siguientes [4]:

Acústica:

Ejemplos de la aplicación de algoritmos genéticos en acústica es el uso de los mismos para el diseño de una sala de conciertos, procurando maximizar la calidad del sonido para la audiencia y la atenuación de ruidos mediante la generación de ondas artificiales destructoras del ruido.

Ingeniería aeroespacial:

Diseño de alas de aviones supersónicos, obteniendo una solución de compromiso entre los parámetros fundamentales tales como minimizar la resistencia aerodinámica a velocidades de vuelo supersónicas, minimizar la resistencia a velocidades subsónicas y minimizar la carga aerodinámica (la fuerza que tiende a doblar el ala). Otros ejemplos del uso de algoritmos genéticos en ingeniería aeroespacial es el uso de estos para la construcción de brazos mecánicos para operar con satélites en el espacio, o el problema de optimizar las rutas de los satélites de comunicaciones teniendo como objetivo minimizar las situaciones de pérdida de cobertura con los centros de comunicación.

Ingeniería eléctrica:
Un ejemplo clásico y el primer caso en que se registrara una patente de un invento desarrollado por una máquina fue el desarrollo de una antena mediante algoritmos genéticos para su aplicación en un ámbito muy concreto. El desarrollo de antenas eficientes no es en absoluto una tarea fácil y requiere no sólo aplicación de cálculos muy complicados, sino que además requiere de varios intentos para aproximar una antena adecuada.

Ingeniería de materiales:
Diseño de nuevos polímeros útiles para el tránsito de la corriente eléctrica basados en el carbono (más conocidos como polianilinas).

Química:
Cálculo de pulsos de láser para la destrucción de moléculas y formación de elementos simples. Desarrollo de nuevos aminoácidos y proteínas mediante la llamada química combinatoria.

Mercados financieros:
Un problema recurrente en cualquier técnica de inteligencia artificial es la modelización de mercados financieros para realizar previsiones y evaluaciones. Cálculos de los tipos de cambio y de acciones bursátiles mediante algoritmos genéticos son ejemplos claros de la aplicación de los algoritmos genéticos a la industria.

Juegos:
Los juegos siempre han sido base de experimentación de muchos de los métodos en inteligencia artificial. En los algoritmos genéticos esta política se sigue cumpliendo. Se han aplicado dichos algoritmos en la resolución de juegos tales como las damas.

Geofísica:
Búsqueda de hipocentros (punto bajo la superficie terrestre en el que se origina un terremoto) a partir de las mediciones del terremoto.

Matemática y algoritmia:
Uso de algoritmos genéticos para resolver ecuaciones de derivadas parciales no lineales de alto orden. Resolución de problemas de ordenación.

Aprendizaje automático:
La capacidad para favorecer a los individuos que se acercan al objetivo, a costa de los que no lo hacen, consigue una nueva generación con mejores reglas y, por lo tanto el sistema será capaz de ir aprendiendo a conseguir mejores resultados. Es aquí donde encuentran un estupendo marco de trabajo los algoritmos genéticos.

Optimización evolutiva de estrategias de control:
Un trabajo realizado en relación a esta área de aplicación es la utilización de algoritmos genéticos para la optimización del control de un sistema de aire acondicionado [5]. El objetivo es alcanzar niveles de temperatura especificados con un mínimo uso de energía eléctrica, basándose en predicciones realizadas por un modelo matemático del sistema.

Las variables incluidas en dicho modelo son la consumición de energía eléctrica por parte del sistema de aire acondicionado, las temperaturas interior y exterior, y la radiación solar. Este modelo permite hacer predicciones precisas de corto plazo acerca de la variación de la temperatura interna del ambiente en función del consumo de energía eléctrica del sistema y del clima externo. Una estrategia óptima de control basada en algoritmos genéticos fue desarrollada e implementada utilizando el modelo dinámico comentado.

Se hicieron simulaciones en computadora de la performance obtenida por el controlador generado con algoritmos genéticos en comparación con dos esquemas tradicionales de control. Se demostró que el controlador generado por dichos algoritmos es superior a los dos enfoques de control convencionales cuando la calidad del control y la consumición de energía son tomados en cuenta.

Optimización evolutiva de tablas de horarios:
Los algoritmos genéticos son utilizados en la optimización evolutiva de tablas de horarios [9]. Un ejemplo típico donde suele aplicarse esta solución es en la construcción de dichas tablas para asignar cursos y exámenes a períodos y aulas en instituciones educativas. Esta tarea representa un problema recurrente.

Generalmente es realizada a mano o con la ayuda limitada de un sistema administrativo simple, requiriendo hacer uso de la tabla de horarios del año anterior, modificándola de acuerdo a las nuevas exigencias. Cada año una nueva tabla de horarios debe ser producida para administrar cambios del staff, de los estudiantes y de los cursos, requiriendo esto una gran cantidad de trabajo administrativo. Para simplificar esta tarea, se han realizado trabajos que hacen uso de algoritmos genéticos para resolver la generación de tablas de horarios tanto en lo referido a la formación de mesas de exámenes como así también al dictado de cursos.

Optimización evolutiva de planificaciones:

Aquí entran en juego áreas donde es útil la planificación automática de tareas. Un ejemplo concreto es la distribución de tareas entre múltiples robots. Este es un problema complejo que requiere una planificación cuidadosa. El objetivo de la optimización evolutiva de planificaciones es, en este caso, planificar de manera óptima la secuencia de movimientos de un sistema de múltiples robots con el fin de realizar una tarea específica.

Un área en particular donde se han realizado trabajos basados en algoritmos genéticos es en la utilización de robots manipuladores en fotogrametría [12]. Dicha disciplina se refiere al uso de medios electrónicos de percepción artificial para el análisis de objetos reales. Por ejemplo, un sistema fotogramétrico puede estar conformado por cuatro robots, cada uno equipado con una cámara, los cuales deben analizar cierta característica visible de un objeto real. Se han desarrollado soluciones basadas en algoritmos genéticos con el fin de planificar la secuencia óptima de trabajo para dichos robots.

Optimización evolutiva de trayectorias:

Existen muchos casos de aplicación de algoritmos genéticos para determinar el mejor camino que un robot debe transitar para llegar a un determinado lugar partiendo de su posición inicial, minimizando la cantidad de movimientos a realizar y evitando la mayor cantidad de obstáculos, si es que los hay.

Procesamiento de imágenes digitales:

Existen trabajos relacionados con el procesamiento de imágenes digitales que hacen uso de algoritmos genéticos para llevar a cabo su propósito [10]. Entre estos encontramos: detección de rasgos faciales, aproximación poligonal de figuras en dos dimensiones, aproximación de curvas digitales por segmentos de recta y arcos, restauración de imágenes en escala de grises, realce de imágenes a color, inclusión de imágenes secretas en el bit menos significativo de una imagen.

Redes neuronales - ajuste de los pesos de conexión en una red neuronal:

Uno de los mayores problemas de las redes neuronales artificiales es la obtención de un buen algoritmo de aprendizaje. Los algoritmos genéticos han sido aplicados con éxito al entrenamiento de redes neuronales estimando el conjunto óptimo de los pesos de conexión de las redes. Se han estudiado numerosas aproximaciones: desde el simple proceso de evolución del conjunto de pesos comentado a continuación, hasta algoritmos con sofisticados operadores de recombinación de individuos con intercambios de partes de la topología de las redes.

En el caso del algoritmo más simple, se parte de que la topología de la red neuronal ya es fija (número de neuronas de entrada y salida, número de capas ocultas, número de neuronas en cada capa oculta y la conectividad entre todas las neuronas) y el algoritmo genético debe encontrar los valores óptimos de pesos; esto es, dicho algoritmo realiza el entrenamiento de la red neuronal. Ya se ha aplicado sin problemas a redes neuronales *«feed-forward»* o alimentadas hacia adelante, cambiando el algoritmo de aprendizaje de retropropagación por el algoritmo genético.

2.3 Programación genética

La programación genética es un concepto que nació a comienzos de la década del noventa gracias al trabajo de John Koza [1], quien es considerado el padre de esta disciplina. **La misma se basa sobre la computación evolutiva tradicional y representa una evolución respecto a esta última, en donde los individuos representan programas computacionales. Constituye un método para que una computadora automáticamente resuelva un problema especificándole «qué es necesario hacer» en vez de «cómo hacerlo».**

Dado a su menor tiempo de vida, existe menor cantidad de trabajos realizados con programación genética en relación a los hechos haciendo uso de algoritmos genéticos o de programación evolutiva. De todos modos, ya existe una abundante cantidad de logros obtenidos mediante programación genética, los cuales son competitivos con resultados obtenidos por inteligencia humana (más adelante se definirá qué significa esto exactamente).

Debido a que este trabajo propone hacer uso de programación genética para dar solución a un problema del área de control, su autor considera necesario brindar una explicación teórica detallada acerca de qué es la programación genética y de cómo opera la misma para lograr su objetivo de generación automática de programas. Esto es explicado en profundidad en la sección 4 «Programación genética».

A continuación se definirá qué significa que la programación genética permite obtener soluciones competitivas con las obtenidas por los seres humanos. Luego se hará un repaso de los principales campos donde se ha aplicado con éxito la programación genética hasta el momento, haciendo una reseña de los principales logros obtenidos en cada uno de esos campos mediante ella y que se han registrado hasta la fecha, a fin de informar al lector acerca del estado del arte de esta disciplina.

2.3.1 Resultados automatizados competitivos con los de seres humanos

En el intento por evaluar un método de resolución de problemas automatizado como lo es la programación genética, surge la pregunta de si los problemas demostrativos que son publicados en referencia al método son realmente dignos reflejos de lo que puede llegar a ser un problema en la práctica del mundo real. **Muchas veces ocurre que los problemas demostrativos en los campos de la inteligencia artificial y del aprendizaje de computadoras no tienen relevancia para las cuestiones que científicos e ingenieros necesitan resolver por fuera de los campos anteriormente nombrados.** Esto se debe a que dichos problemas suelen circular exclusivamente dentro de grupos académicos que estudian una metodología particular.

John Koza, padre de la programación genética, afirma que los problemas tratados en los grupos académicos son útiles como tutoriales o con propósitos introductorios. Sin embargo, Koza sostiene que es hora de que los campos de la inteligencia artificial y el aprendizaje de computadoras comiencen a ofrecer resultados no triviales que satisfagan la condición de ser competitivos con la performance alcanzada por un humano.

Ahora bien, **¿qué significa que el resultado de un método de resolución de problemas automatizado sea competitivo con el de los seres humanos? Esto significa que el resultado creado automáticamente satisface uno o más de los siguientes ocho criterios [6]:**

A. El resultado fue patentado como una invención en el pasado, es una mejora respecto a una invención patentada, o puede calificar en el presente como una nueva invención patentable.
B. El resultado es igual o mejor que uno que fue aceptado como un nuevo resultado científico en el momento en que fue publicado en un diario científico.
C. El resultado es igual o mejor que uno que fue ubicado en una base de datos o archivo de resultados mantenido por un panel internacional de científicos expertos.
D. El resultado es publicable como un nuevo resultado científico, independientemente del hecho de que el mismo fue creado mecánicamente.
E. El resultado es igual o mejor que la solución humana más reciente para un problema de larga data para el cual ha habido una sucesión de soluciones humanas cada vez mejores.
F. El resultado es igual o mejor que uno que fue considerado como un logro en su campo en el momento de su descubrimiento.
G. El resultado resuelve un problema de dificultad indiscutible en su campo.
H. El resultado demuestra fortaleza o triunfa en una competición regulada que involucra competidores humanos (ya sea en la forma de jugadores humanos o programas de computadora escritos por humanos).

Estos ocho criterios poseen el atributo deseable de requerir que el resultado se sostenga por su propio mérito y no por el hecho de que haya sido producido mecánicamente. Un resultado no puede adquirir la calificación de «competitivo con el realizado por humanos» simplemente por ser considerado como «interesante» por los investigadores dentro de campos especializados que están intentando crear inteligencia en una máquina. Por el contrario, un resultado producido por un método automatizado debe adquirir su calificación de «competitivo con el realizado por humanos» independientemente del hecho de que fue generado por un método automático.

2.3.2 Campos de aplicación y trabajos realizados

Es necesario hacernos una pregunta fundamental en este momento: ¿puede ser la programación genética utilizada como un método automatizado de resolución de problemas en la práctica del mundo real? La respuesta es que sí. A continuación, se justificará la anterior afirmación.

John Koza afirma la existencia de treinta y seis instancias registradas hasta la fecha en las cuales la programación genética ha producido un resultado competitivo con el de los seres humanos [6]. Estos resultados competitivos con el de seres humanos incluyen once instancias que infringen patentes anteriormente creadas y diez instancias que duplican la funcionalidad de invenciones patentadas

con anterioridad. Seis instancias se refieren a patentes para circuitos analógicos que fueron publicadas luego de enero del 2000. Veintiuno de estos resultados se refieren a invenciones previamente patentadas, con lo cual se puede afirmar que la programación genética es una máquina de invención automatizada.

Estos resultados competitivos con el de seres humanos provienen de los siguientes cuatro campos:

- *Biología molecular computacional:* la misma trata acerca de comprender las interacciones entre los distintos sistemas de una célula, incluyendo la interrelación del DNA, el RNA y la síntesis de proteínas, como así también el aprendizaje de cómo estas interacciones son reguladas.
- *Autómatas celulares:* un autómata celular es una colección de células coloreadas sobre una plantilla de forma específica que evolucionan a lo largo de un número discreto de pasos de tiempo de acuerdo a un conjunto de reglas basadas en los estados de las células vecinas. Por lo tanto, puede definirse como un grupo de células donde cada célula evoluciona sólo de acuerdo a la interacción con las células vecinas. Las reglas son aplicadas iterativamente para cuantos pasos de tiempo se desee. Los autómatas celulares comenzaron a ser estudiados en la década del cincuenta como un modelo posible para sistemas biológicos, ya que se las concibe como capaces de representar la evolución de organismos vivientes y de minerales. En una dimensión, las células constituyen una línea de puntos. Cada punto posee un valor representado por un color. La evolución de cada punto es determinado por su valor y por el de sus puntos vecinos.
- *Redes de clasificación:* una red de clasificación es un circuito con *n* conexiones de entrada para ítems de datos que ingresan en un orden arbitrario, y *n* conexiones de salida sobre los cuales dichos ítems salen en un determinado orden. Dentro de la red, los únicos elementos utilizados son comparadores, los cuales poseen cada uno dos entradas y dos salidas, y simplemente intercambian los ítems de entrada de datos si es que estos están en el orden equivocado.
- *Síntesis del diseño de la topología y el tamaño de componentes para estructuras complejas, tales como circuitos analógicos eléctricos, controladores y antenas.*

A continuación se presenta una tabla donde figuran algunas de las treinta y seis instancias registradas en las cuales la programación genética ha producido resultados competitivos con el de los seres humanos. Cada entrada de la tabla incluye el criterio por el cual se establece que la instancia correspondiente es competitiva a nivel humano (estos criterios se refieren a los ocho expuestos más arriba).

Invención	***Base de afirmación de resultado competitivo a nivel humano***
Creación de un algoritmo cuántico mejor que el clásico para el problema "early promise" de Deutsch-Jozsa.	B, F
Creación de un algoritmo cuántico mejor que el clásico para el problema de búsqueda en base de datos de Grover.	B, F
Creación de un algoritmo cuántico para el problema de consulta AND/OR de profundidad dos que es mejor que cualquier resultado anteriormente publicado.	D
Creación de un algoritmo cuántico para el problema de consulta OR de profundidad uno que es mejor que cualquier resultado anteriormente publicado. Creación de un protocolo para la comunicación de información a través de una puerta cuántica de la cual no se pensaba previamente que permitiría dicha comunicación.	D D
Creación de un programa de juego de fútbol que ganó sus primeros dos partidos en la competición de robots "Robo Cup" de 1997.	H
Creación de un programa de juego de fútbol que se ubicó en la	H

Creación de un programa de juego de fútbol que se ubicó en la zona media de los resultados finales de la competencia de robots "Robo Cup" en 1998, entre un total de 34 programas escritos por humanos.	H
Creación de cuatro diferentes algoritmos para el problema de identificación de segmentos transmembranales en proteínas.	B, E
Creación de una red de clasificación para siete ítems utilizando sólo dieciséis pasos.	A, D
Redescubrimiento de la topología en escalera de Campbell para filtros pasa bajos y pasa altos.	A, F
Redescubrimiento de la topología elíptica de Cauer para filtros.	A, F
Descomposición automática del problema de síntesis de un filtro cruzado.	A, F
Síntesis de amplificadores de 60 y 96 decibeles.	A, F
Síntesis de circuitos computaciones analógicos para ejecutar funciones cuadrada, cúbica, raíz cuadrada, raíz cúbica, logaritmo y Gaussiana.	A, D, G
Síntesis de un circuito analógico para el control en tiempo óptimo	G

El autor de este trabajo quiere dejar en claro aquí que no está en el alcance del mismo explicar en detalle cada uno de estos logros. Estos resultados pertenecen a campos muy específicos y complejos, por lo que la comprensión en profundidad de cada uno de ellos requeriría un estudio extenso en forma individual. Por lo tanto, el objetivo de enumerar estos logros aquí es simplemente informar acerca de los resultados que la programación genética ha alcanzado en ambientes científicos y de ingeniería, a fin de que el lector conozca el uso que se le da a este método automatizado de resolución de problemas por fuera de ambientes exclusivamente académicos. El lector interesado en profundizar su conocimiento acerca de cada resultado, puede consultar [6].

2.3.3 Otros campos de aplicación

Además de los mencionados anteriormente, existen más trabajos basados sobre el paradigma de programación genética correspondientes a los siguientes campos [1]:

Control óptimo

El control óptimo involucra encontrar una estrategia de control que use las variables del estado actual del sistema para elegir el valor de una o más variables de control que causen un movimiento del estado del sistema hacia un estado deseado de destino, todo esto al mismo tiempo que se minimice o maximice una medida de costo.

La estrategia de control deseada puede ser vista como un programa de computadora que toma las variables de control del sistema como sus entradas y produce valores de las variables de control como sus salidas. Las variables de control, a su vez, causan un cambio en el estado del sistema.

El problema que este trabajo propone resolver mediante programación genética es un problema de control óptimo.

Planeamiento

El planeamiento en la inteligencia artificial y en la robótica requiere encontrar un plan que reciba información de sensores ambientales acerca del estado de varios objetos en un sistema y que luego utilice esa información para seleccionar acciones que cambien el estado del sistema. Por ejemplo, un problema de planeamiento puede ser el de manejar una hormiga artificial a lo largo de un camino irregular para encontrar toda la comida que se encuentre sobre el mismo.

En un problema de planeamiento, el plan deseado puede ser visto como un programa de computadora que toma información de sensores como sus entradas y que produce acciones como sus salidas. Dichas acciones, a su vez, causan un cambio en el estado de los objetos del sistema.

Inducción de secuencias

La inducción de secuencias involucra encontrar una expresión matemática que genere el elemento de secuencia S_j para cualquier índice especificado j de una secuencia $S = S_0, S_1, \ldots, S_j, \ldots$ luego de ver sólo un número relativamente pequeño de ejemplos de valores de esa secuencia.

Por ejemplo, dados los primeros siete valores de la secuencia 2, 5, 10, 17, 26, 37, 50, …, el lector puede inducir que la expresión matemática $j^2 + 1$ es un modo de determinar el elemento de secuencia S_j para cualquier índice especificado *j* de la secuencia.

La expresión matemática que está siendo buscada en un problema de inducción de secuencias puede ser vista como un programa de computadora que toma la posición índice *j* como su entrada y produce el valor de elemento de secuencia correspondiente como su salida.

La inducción de secuencias es un caso especial de la regresión simbólica (que aparece a continuación), en donde la variable independiente (es decir, la posición índice *j*) corresponde sólo a los números naturales.

Regresión simbólica

La regresión simbólica (identificación de funciones) involucra encontrar una expresión matemática, en forma simbólica, que provea un ajuste bueno o perfecto entre una muestra finita de valores de variables independientes y los valores asociados de las variables dependientes. Por lo tanto, la regresión simbólica involucra encontrar un modelo que se ajuste a una muestra de datos dada.

La regresión simbólica involucra encontrar tanto la forma funcional como así también los coeficientes numéricos del modelo. La regresión simbólica difiere de las regresiones convencionales lineal, cuadrática o polinómica, las cuales involucran meramente encontrar los coeficientes numéricos para una función cuya forma (lineal, cuadrática o polinómica) ya ha sido especificada.

En cualquier caso, la expresión matemática que es buscada en la regresión simbólica puede ser vista como un programa de computadora que toma los valores de las variables independientes como sus entradas y produce los valores de las variables dependientes como sus salidas.

En el caso de datos muy dispersos del mundo real, este problema de encontrar un modelo a partir de los datos es usualmente llamado «descubrimiento empírico». Si las variables independientes corresponden al conjunto de los números enteros no negativos, entonces la regresión simbólica es usualmente llamada «inducción de secuencias» (como ya fue explicado más arriba). Si hay múltiples variables dependientes, el proceso es llamado «regresión simbólica múltiple».

Programación automática

Una fórmula matemática para resolver un problema en particular comienza con determinados datos dados (las entradas) y produce determinados resultados deseados (las salidas). En otras palabras, una fórmula matemática puede ser vista como un programa de computadora que toma los valores dados como sus entradas y produce el resultado deseado como su salida.

Por ejemplo, supongamos el siguiente par de ecuaciones lineales

$$a_{11}x_{11} + a_{12}x_2 = b_1$$

y

$$a_{21}x_1 + a_{22}x_2 = b_2$$

con dos desconocidos x_1 y x_2. Las dos fórmulas matemáticas dadas para resolver el par de ecuaciones lineales comienzan con seis valores dados: los cuatro coeficientes a_{11}, a_{12}, a_{21} y a_{22} y los dos términos constantes b_1 y b_2. Las dos fórmulas entonces producen, como su resultado, los valores de las dos variables desconocidas (x_1 y x_2) que satisfacen el par de ecuaciones. Los seis valores dados corresponden a las entradas del programa de computadora. Los resultados producidos por las fórmulas corresponden a la salida del programa de computadora.

Descubrimiento de estrategias de juego

Una estrategia de juego requiere encontrar una estrategia que especifique qué movimiento debe hacer un jugador a cada momento del juego, dada la información correspondiente acerca del mismo. Dicha informa-

ción puede ser un historial explícito de los movimientos previos del jugador, o un historial implícito de sus movimientos previos en la forma del estado actual del juego (por ejemplo, en ajedrez, la posición de cada pieza sobre el tablero).

La estrategia de juego puede ser vista como un programa de computadora que toma información conocida sobre el juego como su entrada y produce un movimiento como su salida.

Por ejemplo, el problema de encontrar una estrategia para que un individuo perseguidor atrape a otro evasor en un juego requiere encontrar el programa de computadora (es decir, la estrategia de juego) que tome las posiciones actuales del perseguidor y del evasor como sus entradas y produzca el movimiento del perseguidor como su salida.

Descubrimiento empírico y pronóstico

El descubrimiento empírico involucra encontrar un modelo que relacione un conjunto dado de valores de variables independientes y los valores asociados (usualmente dispersos) de las variables dependientes para un determinado sistema del mundo real.

Una vez que se haya encontrado un modelo para datos empíricos, el mismo puede ser utilizado para pronosticar los valores futuros de las variables del sistema.

El modelo que es buscado en problemas de descubrimiento empírico puede ser visto como un programa de computadora que toma varios valores de las variables independientes como sus entradas y produce los valores observados de las variables dependientes como sus salidas.

Integración y diferenciación simbólica

La integración y la diferenciación simbólica se refieren a encontrar una expresión matemática que sea la integral o derivada, en forma simbólica, de una curva dada.

La curva dada puede ser presentada como una expresión matemática en forma simbólica o como una muestra discreta de puntos de datos. Si la curva desconocida es presentada como una expresión matemática, primero se la convierte en una muestra finita de puntos, para lo cual se toma una muestra aleatoria de valores de la expresión matemática dada en el intervalo especificado de la variable independiente. Luego se asocia cada valor de la variable independiente con el resultado de evaluar la expresión matemática dada para ese valor de la variable independiente.

Si estamos considerando la integración, se comienza integrando numéricamente la curva desconocida. Esto significa que se determina el área por debajo de dicha curva desde el comienzo del intervalo para cada uno de los valores de la variable independiente. La expresión matemática que está siendo buscada puede ser vista como un programa de computadora que toma cada uno de los valores aleatorios de la variable independiente como su entrada y que produce el valor de la integral numérica para la curva desconocida como su salida.

La diferenciación simbólica es parecida con la excepción que lo que se realiza es diferenciación numérica.

Problemas inversos

Encontrar una función inversa para una curva dada implica encontrar la expresión matemática, en forma simbólica, que sea la inversa de la curva dada.

Se procede como en el caso de la regresión simbólica y se busca una expresión matemática (un programa de computadora) que se ajuste a los datos en la muestra finita. La función inversa para la función dada en un dominio especificado puede ser vista como un programa de computadora que toma los valores de la variable dependiente de la función matemática dada y que produce los valores de la variable independiente como su salida. Cuando encontramos la expresión matemática que se adapte a esta muestra, hemos encontrado la función inversa.

Descubrimiento de identidades matemáticas

Encontrar una identidad matemática involucra encontrar una nueva y no obvia expresión matemática, en forma simbólica, que siempre posea el mismo valor que una expresión matemática dada en un dominio especificado.

Para descubrir una identidad matemática, primero se empieza con la expresión matemática dada en forma simbólica. Luego se convierte dicha expresión en una muestra finita de puntos de datos, para lo cual se toma una muestra aleatoria de valores de la variable independiente que aparece en la expresión dada. Entonces se asocia cada valor de la variable independiente con el resultado de la evaluación de la expresión dada para cada valor de la variable independiente.

La nueva expresión matemática puede ser vista como un programa de computadora. Se procede como en la regresión simbólica y se busca una expresión matemática (un programa de computadora) que se ajuste a los pares de valores dados. Es decir, buscamos un programa de computadora que tome valores aleatorios de las variables independientes como sus entradas y que produzca el valor observado de la expresión matemática dada como su salida. Cuando encontramos una expresión matemática que se ajusta a los datos de muestra y que, por supuesto, sea diferente a la expresión dada, hemos descubierto una entidad.

Inducción de árboles de decisión

Un árbol de decisión es una manera de clasificar un objeto de un universo dado en una clase particular, utilizando sus atributos como base para dicha clasificación. La inducción de un árbol de decisión es, por lo tanto, un enfoque para la clasificación.

Un árbol de decisión corresponde a un programa de computadora que consiste en funciones que prueban los atributos del objeto. La entrada de dicho programa consiste en valores de ciertos atributos asociados a un punto de datos dado. Su salida es la clase en la que se clasifica el punto de datos dado.

Evolución de conducta emergente

La conducta emergente involucra la aplicación repetitiva de reglas simples que llevan a una conducta global de mayor complejidad. El descubrimiento de conjuntos de reglas que producen conducta emergente es un problema de la inducción de programas.

Consideremos, por ejemplo, el problema de encontrar un conjunto de reglas que controlen la conducta de una hormiga individual y que, al ser ejecutado en paralelo en forma simultánea sobre todas las hormigas de la colonia, cause que las hormigas trabajen juntas para localizar toda la comida disponible y para transportarla al nido. Las reglas para controlar la conducta de una hormiga en particular procesa las entradas sensoriales recibidas por ese insecto y determinan la acción que será llevada a cabo por el mismo. La conducta de alto nivel emergerá como el efecto global de muchas hormigas ejecutando simultáneamente el mismo conjunto de reglas simples.

El programa de computadora (es decir, el conjunto de reglas) que está siendo buscado toma la entrada sensorial de cada hormiga como su entrada y produce la acción a realizar por cada insecto como su salida.

2.3.4 Trabajos realizados en el área de control

Dado que aquí se propone la resolución de un problema de control mediante programación genética, se considera apropiado incluir, a modo de ejemplo, dos trabajos realizados en el área específica de control, en los cuales la solución (una estrategia de control) ha sido alcanzada mediante uso de programación genética.

Aplicación militar:

El primero de estos ejemplos se refiere a un trabajo realizado por el Laboratorio de Investigación de la Marina de los Estados Unidos de América. Dicho trabajo propone una solución para la generación de un sistema de navegación autónomo utilizado en vehículos aéreos no tripulados -o UAVs *(«unmanned air vehicles»)* por sus siglas en inglés- usados en aplicaciones bélicas [11].

Dado que el campo de batalla es dinámico, los UAVs necesitan moverse reactivamente frente a locaciones de riesgos de un modo complejo y difícil de ser preprogramado. El Laboratorio de Investigación de la Marina de EE.UU. creó un nuevo enfoque para realizar esta tarea, el cual satisface las necesidades de autonomía y optimización de la navegación de los UAVs. Dicho enfoque se basa en la programación genética.

El trabajo mencionado constituye una solución a un problema de control óptimo en donde el objetivo del controlador es mover autónomamente el UAV a la zona próxima de un radar del modo más rápido y eficiente posible, para luego dar vueltas alrededor de dicho radar. La simulación asigna una posición inicial al UAV y al radar. El controlador de navegación sólo recibe dos datos como entradas: la amplitud y el ángulo de llegada de las señales de radar recibidas. El controlador luego determina los movimientos necesarios del UAV. Para fortalecer la robustez del sistema, se agrega ruido a la señal del radar. La simulación también modela un ángulo de llegada impreciso para la señal de radar recibida.

Debido a la gran cantidad de cómputos necesarios para evolucionar la estrategia de control generada por programación genética, los mismos fueron realizados por un cluster de 92 procesadores Pentium 4 de 2.4 Ghz.

Como resultado de la simulación, fue posible evolucionar los controles de navegación para el UAV, de modo tal que éste fue capaz de volar a un radar de destino usando entradas imprecisas de un sensor en un

ambiente ruidoso. Se seleccionaron parámetros reales de vuelo y se utilizaron sensores de entrada para ayudar en la transferencia de los controladores evolucionados en la simulación hacia UAVs verdaderos. El próximo paso es demostrar el uso de este método en hardware mediante la navegación autónoma de un UAV bajo el comando de una lógica de control evolucionada por programación genética.

Aplicación en robótica:

El segundo ejemplo donde se utilizó programación genética en el área de control se refiere a un problema de robótica [2]. El mismo consiste en la generación automática de una estrategia de control que permita a un robot móvil autónomo mover una caja desde el medio de una habitación de forma irregular hacia la pared (para tener noción del tamaño de la habitación, la pared norte y la pared oeste de la misma miden cada una 27,6 pies). El robot puede tener una posición inicial cualquiera dentro de la habitación. Primero debe encontrar la caja que se encuentra en el medio de ella. Luego debe empujarla hacia alguna de las paredes. Todo esto debe ser realizado dentro de un tiempo especificado.

El robot es capaz de moverse hacia adelante, rotar treinta grados hacia la derecha y rotar treinta grados hacia la izquierda. Una vez que encuentra la caja, puede moverla empujándola. Esta subtarea puede ser dificultosa dado que si el robot no aplica la fuerza del modo correcto respecto al centro de gravedad de la caja, la misma puede empezar a rotar. En ese caso, el robot perdería contacto con la caja y probablemente fallaría su propósito. El robot es considerado exitoso si cualquier parte de la caja toca cualquier pared dentro del tiempo especificado.

La aptitud de cada individuo de la población (es decir, la aptitud de cada estrategia de control candidata a solución) es computada usando cuatro casos de aptitud donde el robot comienza en diferentes posiciones iniciales. La aptitud para cada individuo es la suma de las distancias, tomadas para cada caso de aptitud, entre la pared más cercana a la caja y el punto de la caja más cercano a la misma.

Cada movimiento del robot demora un paso de tiempo. En la simulación de este problema, se definió el número 350 como la máxima cantidad de pasos de tiempo tolerables. Si, para los cuatro casos de aptitud, la caja toca la pared dentro de los 350 pasos de tiempo, entonces se considera que el individuo tiene la mejor aptitud.

El uso de programación genética permitió obtener automáticamente una estrategia de control cuasi-óptima que soluciona el problema descrito al encontrar un individuo (programa) de aptitud cero.

3. Definición del problema

En este trabajo se dará solución a un problema del área de control óptimo haciendo uso de programación genética. El problema elegido involucra desarrollar una estrategia de control (es decir, un programa de computadora) que aplicará una fuerza sobre un carro que se encuentra en movimiento sobre un camino con el fin de detenerlo por completo en el punto cero del mismo en tiempo mínimo. Dicha estrategia de control es un programa de computadora que será desarrollado sin intervención humana, mediante programación genética.

El carro puede moverse hacia la izquierda o hacia la derecha con una determinada velocidad sobre un camino unidimensional infinito sin fricción. El objetivo es centrar y dejar en reposo (velocidad cero) dicho carro en la posición cero de ese camino. Esto debe realizarse en la menor cantidad de movimientos posibles aplicando, a cada segundo, una fuerza de magnitud fija en dirección positiva o negativa, de modo tal que el carro sea acelerado hacia la derecha o hacia la izquierda, respectivamente.

En la siguiente figura vemos el carro, el cual posee un cohete en cada extremo. En este ejemplo, la posición actual *x(t)* del carro es negativa y su velocidad *v(t)* es positiva. Por lo tanto, la posición *x(t)* del carro es a la izquierda del origen (0.0) y su velocidad actual *v(t)* lo lleva hacia la dirección positiva (es decir, hacia la derecha). En consecuencia, la fuerza F es aplicada por uno de los cohetes al carro para acelerarlo en la dirección positiva.

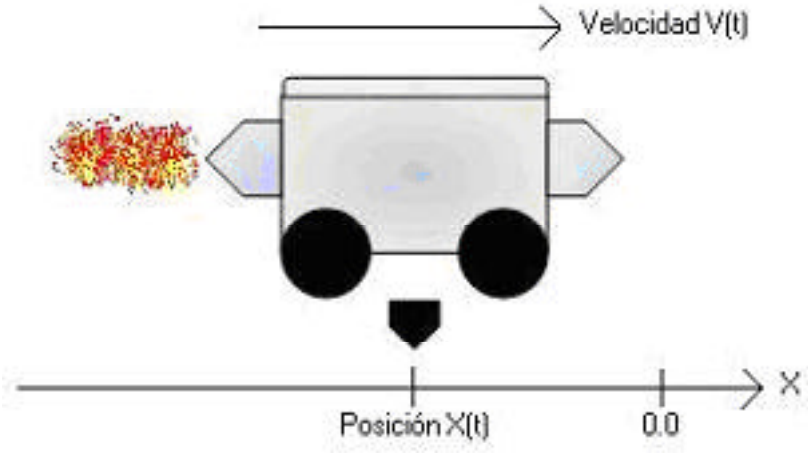

Figura 3. Problema de centrado del carro.

El problema nombrado constituye un problema de control óptimo. Este tipo de problemas involucra un sistema cuyo estado es descrito por variables (denominadas «variables de estado»). La elección de ciertas variables (denominadas «variables de control») causa un cambio en el estado del sistema. El objetivo es entonces elegir el valor de la variable de control de modo tal de provocar que el sistema se dirija hacia un estado de destino especificado con un costo óptimo. El objetivo es típicamente expresado en términos de minimizar el costo.

Existen dos variables de estado para el sistema que este trabajo propone resolver: la posición actual *x(t)* del carro sobre el camino unidimensional y su velocidad *v(t)*. Existe una variable de control para este sistema: la dirección desde la cual un cohete aplica una fuerza F al centro de masa del carro de modo tal de acelerarlo ya sea en dirección positiva o en dirección negativa a lo largo del camino. El estado de destino del sistema es aquel en el cual el carro está en reposo (es decir, con velocidad igual a cero) en la posición cero.

El objetivo del problema es entonces crear una estrategia de control que defina una secuencia de valores para la variable de control a partir de los valores de las variables de estado *x(t)* y *v(t)*, a fin de llevar el estado del sistema al estado objetivo en tiempo mínimo. Los dos valores que puede tomar la variable de control son -1 y 1 (es decir, la variable de control es binaria). Un valor de -1 indicará que el cohete que se debe activar es el de la derecha, lo cual acelerará el carro en dirección negativa (hacia la izquierda); un valor de 1 indicará que el cohete a activar debe ser el izquierdo, lo cual acelerará el carro en dirección positiva (hacia la derecha).

Para medir la eficiencia de una estrategia de control determinada, se ejecuta una simulación del sistema. En ella, la fuerza aplicada por el cohete que se active (sea cual sea) es constante. La activación de un cohete será determinada (mediante el valor binario de la variable de control) a cada segundo por la estrategia de control. La activación de un cohete imprimirá una aceleración en el carrito, el cual recorrerá cierta longitud del camino en función de su velocidad actual durante un segundo. Luego de ese segundo, un cohete será activado nuevamente (el cohete a activar será definido siempre por la estrategia de control), acelerando de nuevo el carrito y modificando así su velocidad. La simulación iterará este proceso hasta que el carro se encuentre en reposo en la posición cero.

A cada paso de tiempo, la elección de la variable de control del sistema causa un cambio en las variables de estado del mismo (las cuales ya dijimos son la posición y la velocidad del carro), trazando así una trayectoria en el espacio de estados posibles del sistema.

Cuando la fuerza *F(t)* es aplicada al carro en el paso de tiempo *t*, el móvil acelera de acuerdo a la ley de Newton del modo siguiente:

$$a(t) = \frac{F(t)}{m}$$

donde *m* es la masa del carro. Por ejemplo, si cada cohete tiene una fuerza de 1 Newton y el carro tiene una masa de 2 kg, entonces la aceleración que provocará la activación de un cohete en un instante tendrá un valor absoluto de 0,5 m/s^2. Luego, como resultado de la aceleración *a(t)*, la velocidad *v(t+?)* del carro en el siguiente paso de tiempo (el cual ocurre un pequeño espacio de tiempo *?* luego del paso de tiempo *t*) cambia a:

$$v(t+\tau) = v(t) + \tau.a(t)$$

donde ? es el tamaño del paso de tiempo.

En el paso de tiempo posterior a *t*, la posición *x(t + ?)* del carro es:

$$x(t + \tau) = x(t) + \tau.v(t)$$

Para la resolución que se dará a este problema, se considerará que = 1 segundo (es decir, uno de los dos cohetes será activado a cada segundo), *m* = 2.0 kg y *F* = 1.0 Newton (o sea, cada cohete posee una fuerza de magnitud constante de 1 Newton).

Como se puede observar, la elección del valor de la variable de control (es decir, la elección de la cantidad *u(t)* igual a un multiplicador de +1 o -1 para la magnitud |F| de la fuerza F) en el paso de tiempo *t* ocasiona un cambio en las variables de estado del sistema en el siguiente paso de tiempo.

El problema entonces se resume en encontrar una estrategia de control de tiempo óptimo que satisfaga las siguientes tres condiciones:

- La estrategia de control debe especificar cómo aplicar la fuerza para los valores de posición *x(t)* y de velocidad *v(t)* en cada paso de tiempo.
- El carro se debe detener por completo en la posición cero.
- La cantidad de movimientos realizados debe ser mínima.

4. Programación genética

A continuación, se brindarán los fundamentos teóricos sobre los cuales se basa la programación genética. Se explicará en forma detallada cómo opera la misma para generar programas de computadora en forma automática.

Este trabajo hará uso de programación genética para la obtención de un programa algorítmico (estrategia de control) que será solución al problema definido en la sección 3 «Definición del problema». Sin embargo, es necesario aclarar que la programación genética también puede ser utilizada para la creación evolutiva de programas de conocimientos. En consecuencia, esta sección será dividida en dos subsecciones:

1) Programación genética para la creación evolutiva de programas algorítmicos: aquí se abarcará la programación genética como una forma automática de generación de programas algorítmicos. Esta subsección será abordada desde un punto de vista netamente teórico. Su objetivo será informar al lector, en forma detallada, cómo opera este paradigma para generar automáticamente programas algorítmicos.

2) Programación genética para la creación evolutiva de programas de conocimientos: esta segunda subsección abarcará la programación genética como una forma de creación evolutiva de programas de conocimientos. Para explicarlo, se hará uso de un ejemplo. Dicho ejemplo es una solución alternativa al problema de centrado del carro definido en la sección 3 (es decir, una solución distinta a la que será implementada en este trabajo).

Antes de seguir, se desea repetir en forma enfática que el problema de centrado del carro será solucionado haciendo uso de programación genética para la creación evolutiva de programas algorítmicos. Por lo tanto, dicho paradigma no será utilizado en este trabajo con fines de creación evolutiva de programas de conocimientos.

En consecuencia, la solución descrita en la sección 5 «Solución propuesta» se basa exclusivamente sobre los lineamientos teóricos explicados en la subsección 4.1 «Programación genética para la creación evolutiva de programas algorítmicos». La subsección 4.2 «Programación genética para la creación evolutiva de programas de conocimientos» es entonces presentada sólo con fines de ilustrar al lector acerca de otro uso posible del paradigma sobre el que se enfoca este trabajo.

4.1 Programación genética para la creación evolutiva de programas algorítmicos

4.1.1 Introducción

La aptitud, a lo largo de un período de tiempo, causa la creación de estructura por medio de la selección natural y a través de los efectos de la recombinación sexual (cruce) y de la mutación. Es decir, la aptitud da pie a la estructura.

Los programas de computadora se encuentran entre las estructuras más complejas creadas por el hombre. Tomando como premisa que la estructura se genera a partir de la aptitud, podemos preguntarnos: ¿cómo pueden las computadoras aprender a resolver problemas sin ser explícitamente programadas? En

otras palabras, ¿cómo pueden las computadoras hacer lo que deben hacer, sin necesidad de explicarles exactamente cómo hacerlo?

Un impedimento para que las computadoras aprendan a resolver problemas sin ser explícitamente programadas es que muchos métodos de la inteligencia artificial no buscan soluciones en la forma de programas de computadora. Por lo tanto, **la única verdad es que si estamos interesados en que las computadoras resuelvan problemas sin ser explícitamente programadas, las estructuras que realmente necesitamos son programas de computadora. La computación evolutiva tradicional no nos proporciona dichas estructuras.**

Además, no deberíamos requerir especificar el tamaño, la forma y la complejidad estructural de la solución, tal como los algoritmos genéticos y la programación evolutiva lo exigen. Estos atributos deberían emerger durante el proceso de resolución del problema como resultado de sus exigencias. El tamaño, la forma y la complejidad estructural deben ser parte de la respuesta provista por la técnica de resolución, no debe ser parte de la pregunta.

En consecuencia, cuando lo que necesitamos es la flexibilidad ofrecida por los programas de computadora, el espacio de programas de computadora es aquel sobre el cual deberemos focalizarnos. Para esto se debe recurrir a una búsqueda inteligente y adaptativa sobre dicho espacio, la cual involucra comenzar con una o más estructuras pertenecientes a ese espacio, probar su performance (aptitud) para resolver el problema dado, y luego utilizar esta información de performance para modificar (y en lo posible incrementar) las estructuras actuales del espacio de búsqueda.

Solemos concebir a la inteligencia humana como la única guía exitosa para encontrar un programa que resuelva cierto problema. Cualquiera que haya escrito un programa de computadora probablemente piense que es imposible que un programa sea progresivamente modificado y mejorado de un modo mecánico e independiente del dominio sin intervención de la inteligencia humana. Y si esto llegara a ser concebido como posible, se podría presumir que sólo podría ser aplicable en algunos dominios muy limitados. Sin embargo, la evidencia empírica demuestra lo contrario [1].

4.1.2 ¿Qué es la programación genética?

Una gran variedad de problemas diferentes de distintos campos pueden ser reformulados como problemas de inducción de programas. Es decir, una variedad de problemas pueden ser solucionados mediante el descubrimiento de un programa de computadora que produzca una salida deseada ante determinadas entradas. La programación genética provee una forma de realizar inducción de programas.

La programación genética es entonces la disciplina de la computación evolutiva que provee un modo para realizar una búsqueda en el espacio de programas para encontrar aquel programa que sea más apto en la resolución de un problema dado, todo esto de modo independiente al dominio del problema. El programa (estructura) que emerja del paradigma de la programación genética es una consecuencia de la aptitud. Por lo tanto, una vez más la aptitud da pie a la estructura.

A diferencia de lo que ocurre en los algoritmos genéticos, las estructuras que sufren cambios adaptativos durante la programación genética son activas. No son codificaciones pasivas a un problema, como lo son los strings de caracteres de longitud fija de los algoritmos genéticos. Por el contrario, teniendo una computadora sobre la cual ejecutarlos, los individuos en la programación genética son estructuras activas que son capaces de ser ejecutadas en su forma actual.

4.1.3 Siete principios guía de la programación genética

Los siete principios de corrección, consistencia, justificabilidad, certeza, orden, parsimonia y decisión han jugado un papel importante en la solución exitosa de muchos problemas en la ciencia, la matemática y la ingeniería. Son por lo tanto integrales a nuestro entrenamiento y forma de pensar.

A continuación se explicarán cada uno de estos principios y se verá que la programación genética no respeta ninguno de ellos [1]:

- *Corrección:* La ciencia, la matemática y la ingeniería siempre buscan la solución correcta a un problema. Por supuesto que pueden aceptarse pequeños errores debidos a imprecisiones introducidos por la computación de datos, imprecisiones en datos observados en el mundo real, y pequeñas desviaciones causadas por simplificaciones y aproximaciones en fórmulas matemáticas. Estas desviaciones de la corrección motivadas en la práctica son aceptables no sólo porque son pequeñas numéricamente, sino también porque siempre están centradas sobre la solución correcta. Sin embargo, si el problema es, por ejemplo, resolver la ecuación cuadrática $ax^2 + bx + c = 0$, entonces una fórmula para x como por ejemplo

$$x = [-b + «(b^2\text{-}4ac)] / 2a + 0{,}000000000000001a^3bc$$

es inaceptable como solución para una raíz, a pesar de que el término incorrecto $10^{-15}a^3bc$ introduce error que es considerablemente menor a los errores debidos a imprecisiones computacionales o simplificaciones prácticas que los ingenieros y los científicos rutinariamente aceptan. El término extra $10^{-15}a^3bc$ no sólo es inaceptable, sino virtualmente inimaginable. Esta fórmula simplemente no está asociada a la solución correcta al problema. Por lo tanto es incorrecta. La programación genética trabaja sólo con soluciones admitidas como incorrectas y sólo ocasionalmente produce la solución analíticamente correcta a un problema.

- *Consistencia:* La inconsistencia no es aceptable para la mente lógica en la ciencia convencional, la matemática y la ingeniería. Una característica esencial de la programación genética es que opera simultáneamente con enfoques claramente inconsistentes y contradictorios para resolver el problema. Y aquí no se trata de permanecer con la mente abierta a nuevas soluciones hasta que se cuente con toda la evidencia, ni tampoco se trata de tolerar estos enfoques claramente inconsistentes y contradictorios. La programación genética activamente alienta, preserva y utiliza una diversidad de enfoques inconsistentes y contradictorios para resolver un problema. De hecho, una mayor diversidad permite a la programación genética arribar a una solución más rápidamente.
- *Justificabilidad:* La ciencia convencional, la matemática y la ingeniería favorecen los razonamientos en los cuales las conclusiones se generan a partir de premisas dadas mediante la aplicación de reglas lógicas de inferencia. Por ejemplo, el término extra $10^{-15}a^3bc$ mostrado en la fórmula anterior no posee justificación basada en la matemática de las ecuaciones cuadráticas. No existe secuencia lógica de razonamiento basada en premisas y reglas de inferencia para justificar este término extra. No existe una secuencia lógica basada en premisas y reglas de inferencia para justificar los resultados producidos por la programación genética.
- *Certeza:* Es dificultoso pensar para practicantes de la ciencia convencional, la matemática y la ingeniería que la solución a un problema bien definido de carácter científico, matemático o ingenieril, dependa exclusivamente de pasos ejecutados al azar. Por ejemplo, la investigación de la teoría del caos busca una explicación determinista a un fenómeno que, en superficie, parece totalmente aleatorio. En la programación genética, todo puede suceder y nada está garantizado.
- *Orden:* La gran mayoría de las técnicas de resolución de problemas y algoritmos en la ciencia convencional, la matemática y la ingeniería no sólo son deterministas, sino que también son ordenados en el sentido que proceden de un modo altamente controlado y sincronizado. Es extraño pensar en un número de procesos no coordinados, independientes y distribuidos operando asincrónicamente y en paralelo sin supervisión central. La desprolijidad y el desorden son características centrales de los procesos biológicos que operan en la naturaleza, como así también de la programación genética.
- *Parsimonia:* En el contexto de la solución a problemas de la ciencia convencional, la matemática y la ingeniería, la parsimonia se refiere a la simpleza en las explicaciones o en las soluciones dadas. En las ciencias, existe una fuerte preferencia por las explicaciones parsimoniosas. Las soluciones provistas por la programación genética no satisfacen necesariamente la condición de parsimonia, ya que para la solución provista por dicha programación, puede existir en realidad otra equivalente (que otorgue el mismo resultado), pero cuya expresión sea más simple.
- *Decisión:* La ciencia, la matemática y la ingeniería se focalizan en algoritmos que son decisivos en el sentido de que poseen puntos de terminación bien definidos en los cuales convergen a un resultado que es la solución al problema dado. Los procesos biológicos que operan en la naturaleza y la programación genética usualmente no poseen un punto de terminación claramente definido. Por el contrario, continúan y continúan. Incluso si interrumpimos estos procesos, ellos ofrecen numerosas respuestas inconsistentes y contradictorias (de todos modos, el observador externo es, obviamente, libre de focalizar su atención en la mejor respuesta obtenida).

La posibilidad de que un conjunto totalmente diferente de principios guía sea utilizado en el problema de la programación automática nace en el hecho de cómo la naturaleza crea entidades complejas que resuelven problemas mediante la evolución. A continuación veremos cómo la naturaleza tampoco respeta los principios que usualmente se conciben como fundamentales en la ciencia:

- *Corrección:* la naturaleza crea estructuras a lo largo del tiempo aplicando selección natural en base a la aptitud. Algunas estructuras son mejores que otras; sin embargo, no existe necesariamente una respuesta correcta.
- *Consistencia:* la naturaleza mantiene y cría muchos enfoques inconsistentes y contradictorios a un problema dado. De hecho, el mantenimiento de diversidad genética es un ingrediente importante de la evolución para asegurar la habilidad de adaptación a ambientes cambiantes.
- *Justificación:* en la naturaleza, la diferencia entre una estructura observada hoy y sus antecesores no es justificable en el sentido de que haya una comprobación matemática que la justifique, o en el sentido de

que haya una secuencia lógica de reglas de inferencia que haya sido aplicada a un conjunto original de premisas para producir el resultado observado.

- *Certeza:* en la naturaleza, la evolución no sigue una línea determinista. Todo puede ocurrir y nada está garantizado.
- *Orden:* el proceso evolutivo en la naturaleza es incierto y no determinista. También involucra actividad asincrónica, no coordinada, local e independiente que no es controlada en forma centralizada.
- *Parsimonia:* la aptitud, y no la parsimonia, es el factor dominante en la evolución natural. Una vez que la naturaleza encuentra una solución a un problema, comúnmente continúa respaldando esa solución. La parsimonia sólo juega un rol cuando interfiere con la aptitud (cuando el precio a pagar por una solución indirecta y compleja interfiere con la performance). Como el genoma de los seres vivos, los resultados de la programación genética son raramente la estructura mínima para desarrollar la tarea dada.
- *Decisión:* la evolución es un proceso que no se detiene, que no tiene un punto de terminación bien definido.

4.1.4 Diagrama de flujo de la programación genética

En la sección 2.3 referida al estado del arte de la programación genética, vimos que muchos problemas de campos variados pueden ser reformulados como la búsqueda de un programa que produzca una determinada salida ante ciertas entradas. Por lo tanto, el espacio de búsqueda es el espacio de todos los posibles programas de computadora compuestos por funciones y terminales apropiados al dominio del problema.

Las funciones pueden ser operaciones aritméticas, funciones matemáticas, funciones lógicas o funciones específicas del dominio. Dependiendo del problema en particular, el programa de computadora puede retornar un valor lógico, un valor entero, un valor real, un valor complejo, un vector, un valor simbólico o valores múltiples.

En primer lugar se genera una población aleatoria de individuos (programas), por medio de una búsqueda ciega al azar en el espacio de búsqueda del problema. Cada individuo (programa de computadora) es evaluado en términos de cuan bien se adapta al problema (esto constituye la aptitud del individuo). Típicamente, cada programa de computadora es ejecutado sobre un número de diferentes casos de aptitud o *«fitness cases»*, de modo tal que la aptitud sea medida como la suma o el promedio obtenido en una variedad de diferentes situaciones representativas. Estos casos de aptitud representan una prueba de diferentes valores de la variable independiente o una prueba de diferentes condiciones iniciales del sistema.

Salvo que el problema sea tan simple que pueda ser solucionado mediante una simple búsqueda aleatoria y ciega, los programas de computadora en la generación cero tendrán muy pobre aptitud. Sin embargo, es probable que algunos individuos sean más aptos que otros. Estas diferencias de aptitud serán explotadas.

El proceso genético de cruce sexual entre dos programas de computadora padres seleccionados en función de su aptitud es utilizado para crear dos programas hijos. Los programas padres son típicamente de diferentes tamaños y formas. Sus descendientes estarán compuestos de subexpresiones (subárboles, subprogramas, subrutinas o bloques de construcción) de sus padres. Los descendientes poseen típicamente diferente forma y tamaño que sus padres.

Intuitivamente, si dos programas de computadora son más o menos efectivos en resolver un problema, entonces alguna de sus partes puede tener cierto mérito. Recombinando partes elegidas aleatoriamente de programas efectivos, podemos producir nuevos programas que sean aún más aptos para resolver el problema. Eso es precisamente lo que hace el operador genético de cruce, como veremos más adelante en este trabajo .

Los programas que constituyen la población de cada generación, tenderán a ser más aptos generación tras generación. Cuando cierto criterio de parada sea alcanzado, el mejor individuo (programa de computadora) será entregado como solución.

La programación genética es un método débil (es decir, independiente del dominio), que provee una único enfoque a la complicación de encontrar un programa de computadora que resuelva un problema dado. Este proceso se puede simplificar en los siguientes tres pasos [1]:

1. Generar una población inicial de composiciones aleatorias de funciones y terminales del problema (programas de computadora).
2. Ejecutar iterativamente los siguientes subpasos hasta que el criterio de terminación sea satisfecho:
   a. Ejecutar cada programa en la población y asignarle un valor de aptitud de acuerdo a cuan bien solucione el problema.
   b. Crear una nueva población de programas de computadora aplicando las siguientes dos operaciones primarias. Las operaciones son aplicadas a los programas de computadora en base a cierto criterio de selección.

i. Copiar programas de computadora existentes en la nueva población.
ii. Crear nuevos programas de computadora recombinando genéticamente partes aleatoriamente seleccionadas de dos programas ya existentes.

3. El mejor programa de computadora que haya aparecido en cualquier generación es designado como el resultado de la programación genética. Este resultado puede ser una solución perfecta o aproximada al problema.

El siguiente es el diagrama de flujo del paradigma de la programación genética. El índice *i* se refiere a un individuo en la población de tamaño *M*. La variable *GEN* es el número de la generación actual.

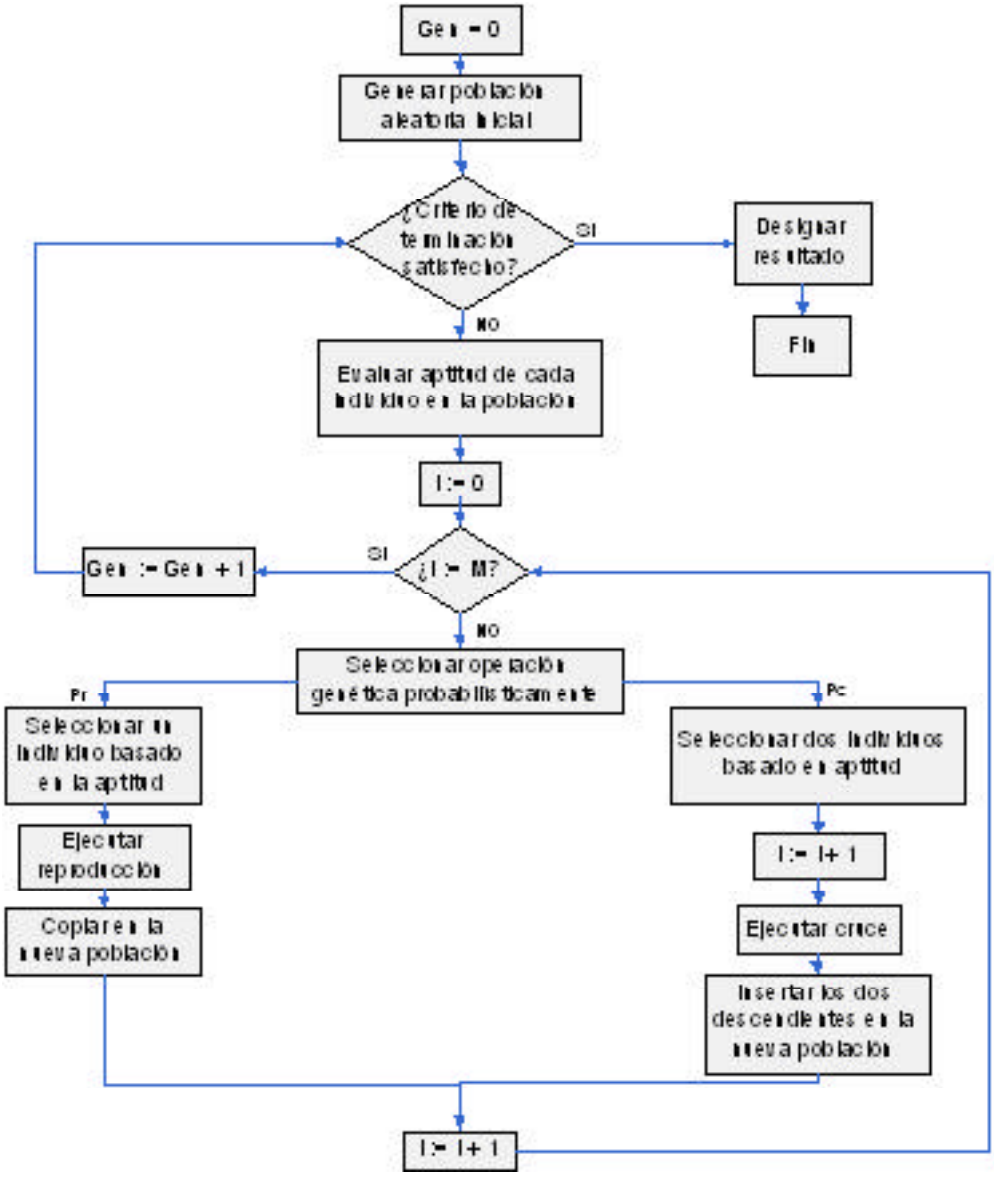

Figura 4. Diagrama de flujo de la programación genética.

4.1.5 Breve introducción al lenguaje de programación LISP

Antes de pasar a la descripción detallada de la programación genética, es conveniente brindar una breve introducción al lenguaje LISP («LISt Processing»), debido a que los programas automáticamente generados por este paradigma están codificados en lenguaje LISP.

LISP posee sólo dos tipos de entidades: átomos y listas. La constante 7 y la variable TIEMPO son ejemplos de átomos en LISP. Una lista es un conjunto ordenado de ítems dentro de un par de paréntesis, como por ejemplo (A B C D) y (+ 1 2).

Una expresión simbólica (Expresión-S) es una lista o un átomo de LISP. La Expresión-S es la única forma sintáctica en este lenguaje de programación.

Por ejemplo, (+ 1 2) es una Expresión-S de LISP. En esta Expresión-S, la función de suma aparece adentro del paréntesis de abertura de la Expresión-S. De este modo se aplica la función de adición + a dos argumentos (los átomos 1 y 2). El valor retornado por la Expresión-S es 3. Estas expresiones son ejemplos de la notación polaca (también llamada notación prefijo).

La Expresión-S (+ (* 2 3) 4) muestra cómo se puede ejecutar una composición es funciones. Aquí primero se aplica la función de multiplicación a los argumentos 2 y 3, lo cual retorna 6. Luego se aplica la función de suma a los argumentos representados por este resultado intermedio (6) y por 4. El resultado final de la Expresión-S es entonces 10.

El término «programa de computadora» por supuesto acarrea la connotación de hacer mucho más que meras operaciones aritméticas, como por ejemplo la posibilidad de ejecutar computaciones alternativas condicionadas por el valor de cálculos intermedios. LISP permite realizar estodel mismo modo, ya que siempre trata el ítem que se encuentra al lado del paréntesis izquierdo más externo como una función y luego aplica esa función a los ítems restantes de la lista (los argumentos).

Por ejemplo, en la Expresión-S (+ 1 2 (IF (> TIEMPO 10) 3 4)), primero se aplica la función > (mayor que) a los argumentos TIEMPO y 10. Esta subexpresión evaluará al valor lógico T o NIL. Si evalúa a T, retorna el primer argumento de la función IF (3); si evalúa a NILL, retorna el segundo argumento (4). Luego, se aplica la función de suma + a los argumentos 1, 2 y al argumento retornado anteriormente por la función IF, el cual ya dijimos puede ser un 3 o un 4. En consecuencia, la Expresión-S retornará 6 o 7.

El principal motivo por el cual es conveniente utilizar Expresiones-S para explicar la programación genética en forma detallada es que las Expresiones-S pueden ser representadas como un árbol que posee una raíz. Para el caso de la Expresión-S del párrafo anterior, dicho árbol es:

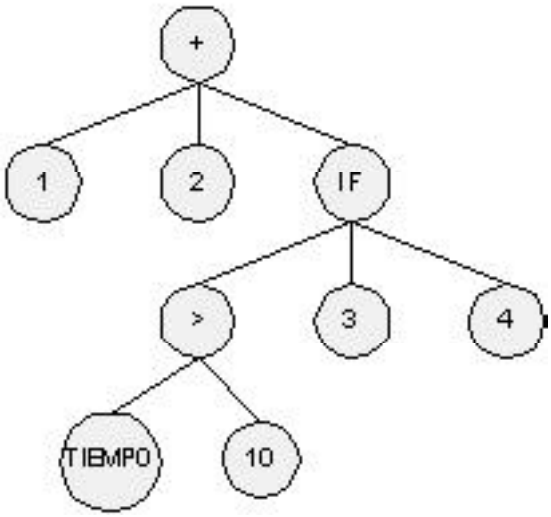

Figura 5. Representación en forma de árbol de una Expresión-S.

Puede notarse que la forma de árbol de una Expresión-S de LISP es equivalente al árbol de parse que muchos compiladores construyen internamente para representar un programa de computadora.

4.1.6 Descripción detallada de la programación genética

Hasta aquí fue visto, en forma básica, cómo actúa la programación genética. Ahora la misma será descrita en forma más detallada en términos de:

- las estructuras que sufren la adaptación,
- las estructuras iniciales,
- la aptitud,
- las operaciones que modifican las estructuras,
- el estado del sistema adaptativo,
- el criterio de terminación,
- la designación de resultado,
- los parámetros de control.

4.1.6.1 Las estructuras que sufren la adaptación

Las estructuras individuales que sufren la adaptación en la programación genética son programas de computadora. Este conjunto posible de estructuras es el conjunto de todas las posibles composiciones de funciones que pueden ser recursivamente creadas a partir del conjunto de N_{func} funciones $F = \{f_1, f_2, \ldots, f_{Nfunc}\}$ y del conjunto de N_{term} terminales $T = \{a_1, a_2, \ldots, a_{Nterm}\}$.

Las funciones en el conjunto de funciones pueden incluir:

- operaciones aritméticas (+, -, *, etc.),
- funciones matemáticas (seno, coseno, log, etc.),
- operadores boléanos (and, or, not, etc.),
- operadores condicionales (if-then-else),
- funciones que causen iteración (do-until),
- funciones que causen recursión, y,
- cualquier otro tipo de función específica del dominio que sea definida.

Los terminales son típicamente variables atómicas (representando, por ejemplo, las entradas, los sensores o las variables de estado de un sistema) o constantes atómicas (como el número 3 o la constante booleana NIL).

Afiancemos ahora estos conocimientos mediante el siguiente ejemplo.

Supongamos que tenemos el conjunto de funciones

$$F = \{AND, OR, NOT\}$$

y el conjunto de terminales

$$T = \{D0, D1\},$$

donde D0 y D1 son variables atómicas booleanas que sirven como argumentos para las funciones. Supongamos que deseamos crear un programa que implemente la función OR no exclusivo (XNOR), de modo tal que retorne verdadero si y sólo si uno de sus argumentos es verdadero y que retorne falso en otro caso. La Expresión-S de LISP que constituye este programa es

(OR (AND (NOT D0) (NOT D1)) (AND DO D1))

y puede ser representada mediante el árbol que se muestra en la siguiente figura:

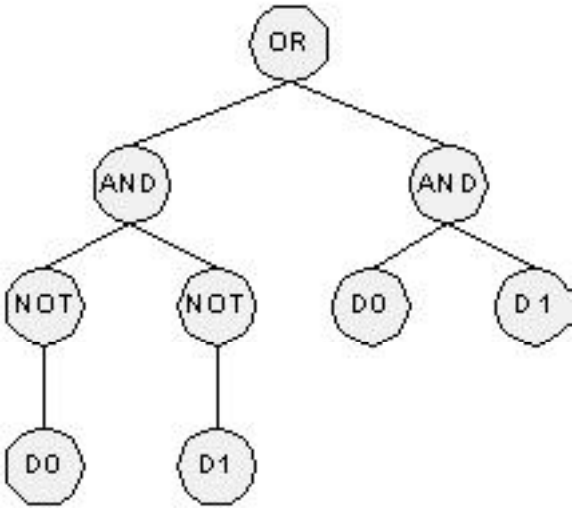

Figura 6. Expresión-S que implementa la función XNOR.

Como podemos ver, los cinco puntos internos del árbol están rotulados con las funciones (OR, AND, NOT, NOT, AND). Los cuatro puntos externos (las hojas) del árbol están rotuladas con los terminales (las variables atómicas booleanas D0, D1, D0, D1, respectivamente). La raíz del árbol está rotulada con la función que aparece más a la izquierda de la Expresión-S de LISP (OR). Este árbol es equivalente al árbol de parse que muchos compiladores construyen internamente para representar un programa de computadora.

El espacio de búsqueda de la programación genética es entonces el espacio de todas las posibles Expresiones-S de LISP que pueden ser creadas recursivamente mediante composiciones de las funciones y de los terminales disponibles en el problema. Estas son las estructuras que sufren las adaptaciones. Este espacio puede ser visto, equivalentemente, como el espacio de árboles que poseen puntos internos rotulados con las funciones disponibles y puntos externos (hojas) rotulados con los terminales disponibles.

El conjunto de funciones y terminales utilizados deben respetar las siguientes dos propiedades:

· *Cierre del conjunto de funciones y del conjunto de terminales:* la propiedad de cierre requiere que cada una de las funciones en el conjunto de funciones sea capaz de aceptar, como sus argumentos, cualquier valor y tipo de dato que sea posiblemente retornado por cualquier función en el conjunto de funciones, y que sea también capaz de aceptar, como sus argumentos, cualquier valor y tipo de dato que sea posiblemente asumido por cualquier terminal en el conjunto de terminales. Esto significa que cada función en el conjunto de funciones debe estar bien definido y cerrado para cualquier combinación de argumentos que encuentre.

En el caso simple en el que el conjunto de funciones consista en funciones booleanas como AND, OR y NOT, y el conjunto de terminales esté compuesto por variables booleanas que sólo pueden asumir T o NIL, la propiedad de cierre es satisfecha fácilmente.

Sin embargo, en programas comunes, suele ocurrir, por nombrar algunos ejemplos, que operaciones aritméticas sobre variables numéricas estén a veces indefinidas (división por 0), o que valores booleanos retornados por un operador condicional no sean aceptados como argumentos en operaciones aritméticas.

Pareciera entonces que la propiedad de cierre no puede ser satisfecha para programas comunes de computadora, o que si puede serlo, implicará una sintaxis muy restrictiva y compleja impuesta sobre los programas. Pero en realidad esto no es difícil. La propiedad de cierre puede ser alcanzada en forma sencilla para la mayoría de los problemas teniendo en cuenta un número pequeño de situaciones.

Por ejemplo, en el caso de que la operación aritmética de división pueda encontrar un 0 como segundo argumento, la propiedad de cierre no será satisfecha salvo que esta situación sea tratada de algún modo. Un enfoque posible puede ser implementar la función de división protegida %, la cual toma dos argumentos y retorna 1 cuando se intenta división por 0. Puede ser programa sencillamente del siguiente modo en LISP:

```
(defun % (numerador denominador)
    «Función de división protegida»
    (if (= 0 denominador) 1 (/ numerador denominador))).
```

De este modo, en lugar de utilizar la división común (/), utilizaríamos siempre la división protegida (%), logrando así la propiedad de cierre.

Otro ejemplo de problema y su respectiva solución en relación a la propiedad de cierre es el de un programa que posee un operador condicional que retorna (como sucede comúnmente) un valor booleano que no es aceptable. En este caso, se puede utilizar una lógica basada en números. Por ejemplo, para el caso de la función booleana «mayor que», podemos reemplazarla creando un operador comparativo que retorne valores numéricos, de modo tal que retorne 1 si el primer argumento es mayor que el segundo, o -1 en otro caso. Este operador que llamamos «gt» (greater than) puede ser programado en LISP del siguiente modo:

```
(defun gt (primer-argumento segundo-argumento)
    «La función mayor-que valuada numéricamente»
    (if (> primer-argumento segundo-argumento) 1 -1))).
```

· Suficiencia del conjunto de funciones y del conjunto de terminales: la propiedad de suficiencia requiere que el conjunto de terminales y el conjunto de funciones primitivas sean capaces de expresar una solución al problema. El usuario de la programación genética debe saber o creer que alguna composición de las funciones y los terminales que él proporciona puede llevar a la solución del problema.

Dependiendo del problema, la identificación de los terminales y de las funciones primitivas puede ser un paso obvio, un paso difícil o un paso imposible (ejemplo de esto último es predecir el resultado de elecciones).

Más allá de todos los ejemplos que puedan ser mostrados, la única verdad es que es siempre el usuario el que debe proveer el conjunto de funciones y de terminales apropiado para la resolución del problema, de acuerdo a sus conocimientos.

4.1.6.2 Las estructuras iniciales

La población inicial en la programación genética está compuesta por individuos representados por Expre-

siones-S. Cada Expresión-S es realizada mediante la generación al azar de un árbol rotulado en cada uno de sus puntos.

Se comienza seleccionando una de las funciones en el conjunto F al azar (utilizando una distribución uniforme de probabilidad). Dicha función será el rótulo de la raíz del árbol. Restringimos la selección del rótulo de la raíz al conjunto F porque no queremos generar una estructura compuesta sólo por un terminal.

La siguiente figura muestra el comienzo de la creación de un árbol aleatorio. Se seleccionó la función + del conjunto de funciones, la cual lleva dos argumentos.

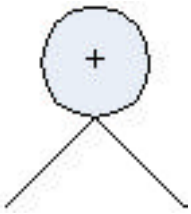

Figura 7. Nodo raíz de la Expresión-S.

Cuando un punto del árbol es rotulado con la función *f* del conjunto *F*, entonces se dibujan *z(f)* líneas irradiándose desde ese punto, donde *z(f)* es la aridad de *f*. Por cada línea, un elemento del conjunto de funciones y de terminales es seleccionado al azar para ser el punto final de esa línea.

Si se selecciona una función como rótulo para cualquiera de estos puntos finales, el proceso de generación entonces sigue recursivamente como se describió más arriba. Si se selecciona un terminal como rótulo para cualesquiera de estos puntos finales, ese punto se convierte en una hoja del árbol y el proceso de generación es terminado para ese punto.

Un ejemplo de estructura ya terminada es el siguiente árbol:

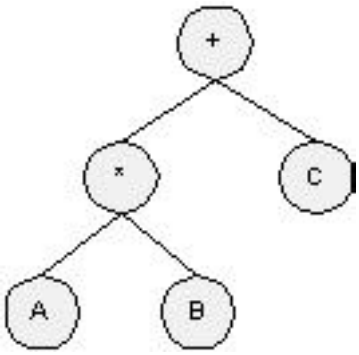

Figura 8. Ejemplo de Expresión-S ya terminada.

El proceso generativo puede implementarse de varias formas, lo cual determinará árboles iniciales de distintas formas y tamaños. A continuación se explicarán tres métodos básicos de llevar a cabo estar tarea:

- Método completo: se crean árboles para los cuales la longitud del camino desde cualquier hoja hasta la raíz es igual a la máxima profundidad especificada.
- Método creciente: se generan árboles de formas variadas, para lo cual la longitud del camino entre una hoja y la raíz es menor o igual a la máxima profundidad especificada.
- Método *«ramped half and half»*: es un método generativo mixto que incorpora tanto el método completo como el creciente. Involucra crear un número igual de árboles usando un parámetro de profundidad que varíe entre 2 y la máxima profundidad especificada. Por ejemplo, si la máxima profundidad especificada es 6, 20% de los árboles tendrán profundidad 2, 20% de los árboles tendrán profundidad 3, y así hasta la profundidad 6. Luego, para cada valor de profundidad, 50% de los árboles son creados vía el método completo y 50% de ellos son creados por el método creciente.

La duplicación de individuos en la población inicial es inservible. Reduce la diversidad genética de la población y desperdicia recursos computacionales. Por lo tanto es deseable, pero no imprescindible, evitar duplicados en la población aleatoria inicial.

4.1.6.3 Aptitud

La aptitud es un valor numérico que se asigna a cada individuo en relación directa a la calidad del mismo. En algunos casos, el desarrollo de una función bondad involucra hacer una simulación de un sistema.

La aptitud puede ser expresada de las siguientes formas:

· *Aptitud cruda («raw fitness»):* la aptitud cruda es la medida de aptitud que es expresada en la terminología natural del problema. Por ejemplo, si tenemos un problema en el cual una hormiga debe recorrer un camino tratando de comer la máxima cantidad de comida situada sobre él, entonces la aptitud cruda es la cantidad de comida tomada por el insecto.

La aptitud es, por lo general, evaluada sobre un conjunto de casos de aptitud. Estos casos de aptitud proveen un modo de evaluar una Expresión-S sobre un número de casos representativos. La definición más común de aptitud cruda de una Expresión-S es la suma de distancias, tomadas sobre todos los casos de aptitud, entre el punto retornado por la Expresión-S para el conjunto de argumentos asociados al caso de aptitud particular, y el punto correcto asociado a dicho caso. Por lo tanto, la aptitud cruda de una Expresión-S *i* en una población de tamaño *M* en el tiempo *t* es:

$$r(i,t) = \sum_{j=1}^{Nc} \left| S(i,j) - C(j) \right|$$

Como la aptitud cruda es expresada en la terminología natural del problema, el mejor valor puede ser pequeño (como cuando se trata de error) o grande (como cuando se trata de cantidad de comida obtenida, beneficio alcanzado, etc.).

· *Aptitud estandarizada («standardized fitness»):* la aptitud estandarizada *s(i,t)* expresa la bondad de un individuo de modo tal que siempre un valor numérico menor sea un mejor valor.

Si para un problema particular, un menor valor de aptitud cruda es mejor, entonces la aptitud estandarizada es igual a la cruda. Es decir:

$$s(i,t) = r(i,t)$$

Es deseable hacer que el mejor valor de aptitud estandarizada sea igual a 0. Si este no es el caso, conviene sumar o restar una constante.

Si para un problema particular, un mayor valor de aptitud cruda es mejor, entonces la aptitud estandarizada se computa a partir de la cruda del siguiente modo, de modo tal que el mejor valor posible de aptitud estandarizada sea cero:

$$s(i,t) = r_{max} - r(i,t)$$

Si no se conoce un límite r_{max} y un número mayor de aptitud cruda es mejor, o si un número menor de aptitud cruda es mejor y no existe un límite inferior, entonces la aptitud estandarizada no puede ser calculada.

Aptitud ajustada («adjusted fitness»): la aptitud ajustada *a(i,t)* se calcula a partir de la estandarizada del siguiente modo:

$$a(i,t) = \frac{1}{1 + s(i,t)}$$

Donde *s(i,t)* es la aptitud estandarizada para el individuo *i* en tiempo *t*.

La aptitud ajustada toma valores entre 0 y 1. Es mayor para mejores individuos. Esta forma de expresar la aptitud tiene el beneficio de que exagera la importancia de pequeñas diferencias en el valor de la aptitud estandarizada a medida que dicha aptitud se acerca a 0. Sin embargo, su uso no es obligatorio en la programación genética.

· *Aptitud normalizada:* la aptitud normalizada *n(i,t)* se computa a partir de la aptitud ajustada *a(i,t)* del siguiente modo:

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)}$$

La aptitud normalizada tiene las siguientes tres características deseables:
1. Varía entre 0 y 1.
2. Es mayor para mejores individuos en la población.
3. La suma de las aptitudes normalizadas de todos los individuos de una población es 1.

4.1.6.4 Operaciones para modificar las estructuras

Aquí se describirán las operaciones de reproducción darwiniana, de cruce y de mutación.

· Reproducción

Esta operación, la cual es asexual porque actúa sobre un solo individuo a la vez, consiste en dos pasos muy simples. Primero, una única Expresión-S es seleccionada de la población utilizando algún método de selección (en caso que el lector no conozca métodos de selección, se propone la lectura del apéndice «Métodos de selección»). Luego, el individuo seleccionado es copiado, sin alteración, desde la población actual hacia la nueva población (la nueva generación).

· Cruce

El operador de cruce comienza con dos Expresiones-S padres y produce dos Expresiones-S descendientes. Es una operación sexual por lo tanto. Los padres son seleccionados según algún método de selección.

La operación comienza seleccionando (utilizando una distribución uniforme de probabilidad) un punto de cruce para cada padre por separado. El fragmento de cruce para un padre en particular es el árbol que posee su raíz en el punto de cruce de ese padre y que consiste del subárbol entero que se encuentra por debajo del punto de cruce. Visto en términos de Expresiones-S de LISP, el fragmento de cruce es la sublista que comienza en el punto de cruce. Dicho subárbol puede consistir en un solo terminal (si el punto de cruce es un terminal) o incluso en el árbol completo que representa a la Expresión-S (si el punto de cruce es la raíz de dicho árbol).

La primera Expresión-S descendiente se producirá borrando el fragmento de cruce del primer padre en el primer padre e ingresando el fragmento de cruce del segundo padre en el punto de cruce del primer padre. El segundo descendiente se obtiene de modo simétrico.

Por ejemplo, supongamos que tenemos los dos padres representados por los siguientes árboles:

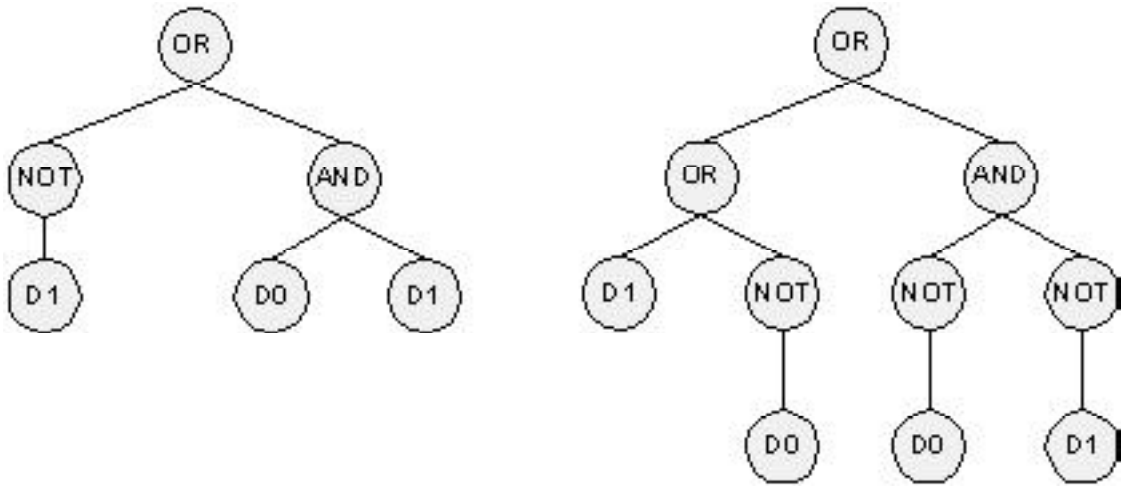

Figura 9. Expresiones-S a ser cruzadas.

En términos de Expresiones-S, estos dos padres son:

(OR (NOT D1) (AND D0 D1))

y

(OR (OR D1 (NOT D0)) (AND (NOT D0) (NOT D1)))

Asumimos que los nodos de ambos árboles están numerados de izquierda a derecha de modo *«depth first»*. Supongamos que el punto de cruce para el primer padre es el 2 (NOT), mientras que el punto de cruce para el segundo padre es el 6 (AND). Por lo tanto, los dos fragmentos de cruce son los siguientes:

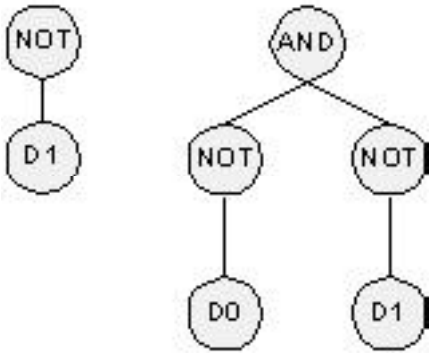

Figura 10. Fragmentos de cruce.

Luego del cruce, los dos descendientes son los siguientes árboles:

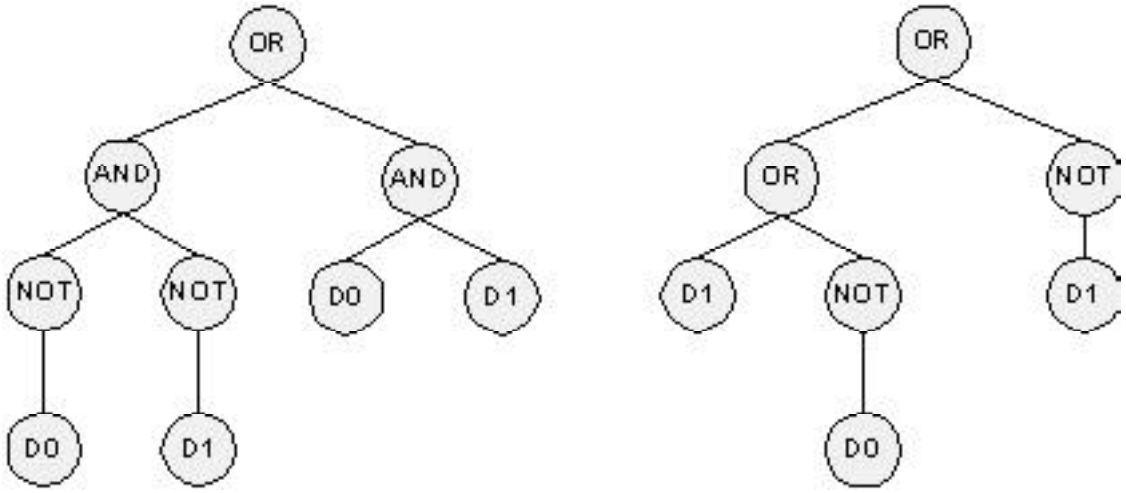

Figura 11. Expresiones-S descendientes luego de la operación de cruce.

En términos de Expresiones-S de LISP, los dos descendientes son:

(OR (AND (NOT D0) (NOT D1)) (AND DO D1))
y
(OR (OR D1 (NOT D0) (NOT D1))

Como consecuencia de la propiedad de cierre, los árboles resultantes siempre son Expresiones-S de LISP sintácticamente legales.

Cabe destacar que tanto la raíz de un árbol como uno de sus terminales pueden ser elegidos como puntos de cruce.

· Mutación

En el caso de los algoritmos genéticos, la operación de mutación tiende favorecer la diversidad, evitando así la convergencia. Sin embargo, como veremos más adelante en la sección 4.1.7 «Convergencia de la población en la programación genética» de este trabajo, es improbable que ocurra dicha convergencia en la programación genética.

La mutación es asexual y opera sobre una única Expresión-S, la cual es seleccionada en base a una probabilidad proporcional a su aptitud.

La operación de mutación comienza seleccionando un punto aleatorio dentro de la Expresión-S. Dicho punto puede ser interno o externo. La operación de mutación remueve lo que esté seleccionado en el punto elegido como así también todo aquello que esté por debajo de ese punto, y luego inserta un árbol creado aleatoriamente en ese punto. Esta operación es controlada por un parámetro que especifica el máximo tamaño (medido por profundidad) para el nuevo subárbol creado que será insertado.

Por ejemplo, en la siguiente figura, el punto 3 (D0) de la Expresión-S «Antes» es elegido como punto de mutación. La subexpresión (NOT D1) es generada al azar e insertada en ese punto para producir la Expresión-S que se muestra en el árbol «Después».

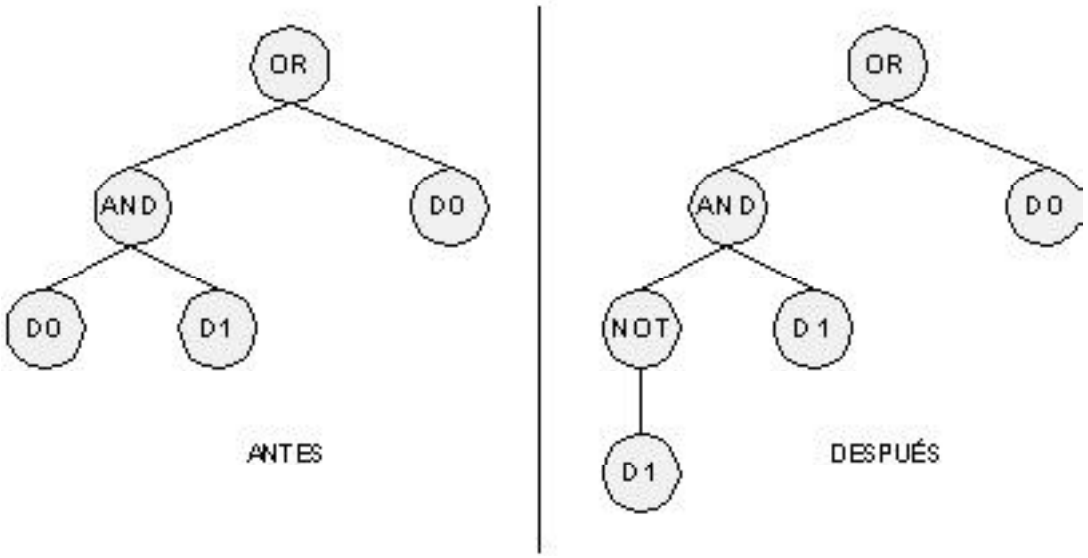

Figura 12. Ejemplo de operación de mutación.

4.1.6.5 Estado del sistema adaptativo

En la programación genética, el estado del sistema en cualquier momento durante el proceso consiste sólo de la población actual de individuos.

En una implementación computacional de este paradigma, es necesario colocar en memoria los parámetros de control, el conjunto de terminales, el conjunto de funciones, y el mejor individuo hasta el momento (si es que se utiliza ese método de designación de resultado).

4.1.6.6 Criterio de terminación

El criterio de terminación para la programación genética puede estar dado cuando se alcance un cierto número máximo *G* de generaciones en una ejecución, o cuando algún predicado de éxito específico del problema sea alcanzado (como por ejemplo, encontrar una solución 100% correcta).

4.1.6.7 Designación de resultado

Un método posible de designación de resultado consiste en designar como tal al mejor individuo que haya aparecido en cualquier generación, para lo cual es necesario colocarlo en memoria durante la ejecución.

Otro método alternativo es el de designar como resultado al mejor individuo de la población al momento de terminar la ejecución. Obviamente, no es necesario colocar al individuo en memoria en este último caso.

Por lo general, el mejor individuo hasta el momento se encontrará en la población al momento de la terminación, en cuyo caso ambos métodos designarían el mismo resultado.

4.1.6.8 Parámetros de control

Los siguientes parámetros son utilizados para controlar la ejecución de la programación genética:

- Tamaño M de la población.
- Máximo número G de generaciones.
- Probabilidad de cruce P_c.
- Probabilidad de reproducción P_r.
- Para seleccionar los puntos de cruce, se puede usar una distribución de probabilidad que asigne P_{ip} probabilidades de que el punto de cruce sea un punto interno (función). Conviene que P_{ip} sea alta, de modo tal que se promueva la recombinación de grandes estructuras en lugar de un mero intercambio de terminales de árbol a árbol.
- Profundidad máxima $D_{created}$ para las Expresiones-S creadas por la operación de cruce.
- Profundidad máxima $D_{initial}$ para los individuos creados al azar en la población inicial.
- Probabilidad de mutación P_m.

4.1.7 Convergencia de la población en la programación genética

En el algoritmo genético convencional, cuando un individuo se cruza incestuosamente consigo mismo, los dos descendientes resultantes serán idénticos. Este hecho fortalece la tendencia hacia la convergencia en el algoritmo genético. La convergencia se denomina prematura si la población converge hacia un resultado globalmente subóptimo. La convergencia prematura puede ocurrir cunado un individuo subóptimo mediocre posee extraordinariamente buena aptitud en relación a otros individuos en la población en ese momento. En esta situación (usualmente llamada «supervivencia del mediocre»), el algoritmo genético falla en encontrar el óptimo global. Una vez que la población converge en el algoritmo genético convencional, la única forma de cambiar la situación es mediante mutación. La mutación puede, en principio, llevar a cualquier parte; sin embargo, en la práctica, la población rápidamente reconverge.

En contraste, en la programación genética, cuando un individuo se cruza consigo mismo, los dos descendientes resultantes serán, en general, distintos (excepto en el caso infrecuente en el que los puntos de cruce para ambos individuos sean los mismos). Como antes, la operación Darwiniana de reproducción crea una tendencia hacia la convergencia; sin embargo, en la programación genética, la operación de cruce genera una presión contrapuesta que aleja de la convergencia [1]. Por lo tanto, se concluye que la convergencia de la población es improbable en la programación genética. Es por eso que la operación de mutación es aquí considerada como secundaria.

4.2 Programación genética para la creación evolutiva de programas de conocimientos

En la subsección anterior se describió detalladamente el modo de operación de la programación genética para la obtención automática de programas de computadora de tipo algorítmico. Ese es precisamente el uso que se le dará a este paradigma para la resolución del problema planteado en este trabajo.

Ahora bien, **la programación genética también puede ser utilizada para la creación evolutiva de programas de conocimientos. Estos conocimientos son expresados en forma de reglas, las cuales constituyen una base de reglas.**

Para explicar cómo opera la programación genética con este fin, haremos uso de un ejemplo a lo largo de esta subsección. Dicho ejemplo, como ya dijimos anteriormente, constituye un modo de resolución alternativo (respecto al propuesto en la sección 5) al problema de centrado del carro definido en la sección 3. Este método de resolución alternativo hace uso de reglas de lógica borrosa evolucionadas mediante programación genética para resolver el problema nombrado.

Se pretende entonces en esta subsección cumplir dos objetivos:

a) Demostrar cómo la programación genética es utilizada no sólo para la generación automática de programas algorítmicos, sino también para la generación automática de bases de reglas que representan programas de conocimientos.
b) Mostrar una solución alternativa a la que será desarrollada en la sección 5 de este trabajo para el problema definido en la sección 3.

4.2.1 Diseño evolutivo de controladores basados en lógica borrosa

Un enfoque evolutivo basado en programación genética puede ser utilizado en el diseño de controladores basados en lógica borrosa. El objetivo es la producción de bases de reglas de lógica borrosa, las cuales internamente son representadas como árboles sintácticos.

Este modelo ha sido aplicado específicamente al problema de centrado del carro en un trabajo realizado en la Universidad de Málaga en España [3]. El mismo fue expuesto en el Simposio Internacional de Control Inteligente de la IEEE en 1996. Los resultados obtenidos en dicho trabajo demuestran que una buena parametrización del algoritmo y una función de evaluación apropiada llevan a soluciones cuasi-óptimas.

El trabajo es novedoso en cuanto a que presenta un enfoque en el cual los individuos (árboles) no representan programas computacionales tradicionales, sino bases de reglas de lógica borrosa cuyo objetivo es la resolución de un problema de control (el problema de centrado del carro, como ya dijimos anteriormente).

Antes de explicar cómo se ha utilizado el paradigma de programación genética para llevar a cabo esta resolución del problema mencionado, es primero necesario explicar brevemente qué son los controladores de lógica borrosa. Luego se brindará un resumen del trabajo realizado por el equipo de la Universidad de Málaga.

4.2.2 Controladores de lógica borrosa

Los FLCs («*Fuzzy Logic Controllers*», por sus siglas en inglés) son sistemas basados en reglas que incorporan exitosamente la flexibilidad de la capacidad de decisión humana por medio del uso de la teoría de conjuntos borrosos *(«fuzzy sets»)*. Expresiones del lenguaje natural (por ejemplo: «la presión es baja») son

utilizadas en las reglas borrosas. La ambigüedad de las mismas es modelada por funciones miembro, las cuales procuran representar la concepción de un experto humano respecto a los términos lingüísticos. Una función miembro brinda una medida de la confianza con la cual un valor numérico preciso es descrito por una etiqueta lingüística.

Las reglas tienen la forma de IF [*condiciones*] THEN [*acciones*], donde las condiciones y las acciones son términos lingüísticos que describen los valores de las variables de entrada y de salida, respectivamente (por ejemplo: IF *presión* IS *baja* THEN *apertura-válvula* IS *small*). La base de reglas borrosas del FLC está constituida por un número de dichas reglas. Esta base es utilizada para producir valores de salida precisos de acuerdo a valores precisos de entrada. Este proceso de control es dividido en tres etapas (utilizamos la terminología inglesa):

a) *«Fuzzification»:* se calcula la entrada borrosa, es decir, se evalúan las variables de entrada (constituidas por valores numéricos precisos) con respecto a sus expresiones lingüísticas en el lado de la condición de la regla.

b) *«Fuzzy inference»:* se calcula la salida borrosa, es decir, se evalúa la fortaleza de la activación de cada regla y se combinan los respectivos lados de acción de cada una de ellas.

c) *«Defuzzification»:* se calcula la salida final, es decir, se convierte la salida borrosa a un valor numérico preciso.

Este proceso es ilustrado en la siguiente figura:

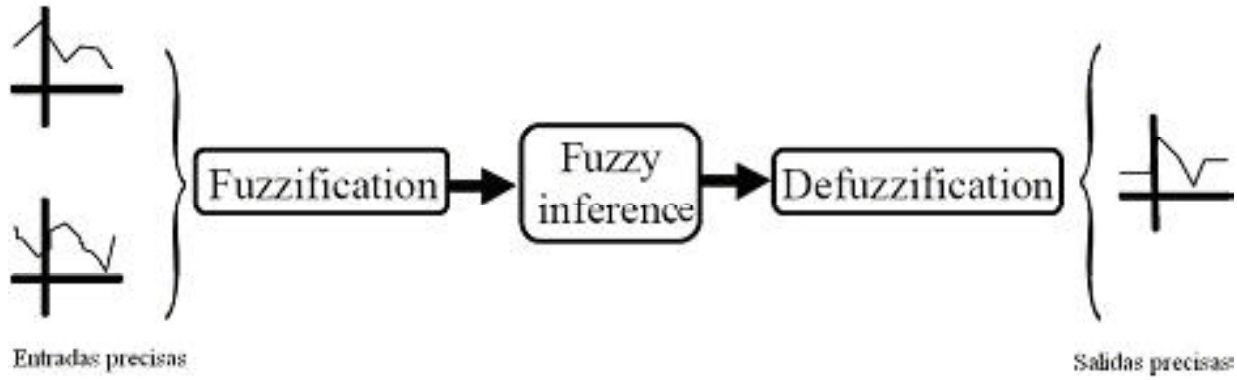

Figura 13. Proceso de control borroso.

4.2.3 Descripción del problema a ser solucionado evolutivamente mediante control borroso

El problema que propone resolver el trabajo presentado en el Simposio Internacional de Control Inteligente de la IEEE de 1996 es exactamente el mismo descrito en la sección 3 «Definición del problema» de este trabajo. Lo único que varía es la selección que el autor de este trabajo y que el grupo de trabajo en la Universidad de Málaga han hecho para el valor de ?(segundos que dura cada paso de tiempo) y para el valor de *F* (magnitud de la fuerza a aplicar sobre el carro). El valor de la masa del carro es el mismo en ambos casos: 2 kg.

El valor elegido por el grupo de trabajo de la Universidad de Málaga para la cantidad de segundos que dura cada paso de tiempo es = 0.02 segundos. Para el caso de la fuerza *F* a aplicar sobre el carro, mientras que en la resolución dada para este problema en este trabajo se utiliza una fuerza de valor absoluto constante igual a 1.0 Newtons, en el trabajo realizado en la Universidad de Málaga se considera el uso de una fuerza de magnitud variable con un valor máximo de *F* = 2.5 Newtons.

La solución óptima para este problema es conocida y consiste en aplicar una fuerza en el sentido positivo (es decir, que empuje el carro hacia la derecha) si se cumple que

$$-x(t) > \frac{v(t)^2 \, Signo(v(t))}{2|F|/m}$$

, y aplicar una fuerza en el sentido contrario (es decir, que empuje el carro hacia la izquierda) en caso contrario.

4.2.4 Diseño de un controlador de lógica borrosa

Dado que éste se trata de un problema de control óptimo, el problema de centrado del carro es apto de ser solucionado mediante control borroso. El desarrollo de un FLC para este sistema involucra los siguientes tres pasos:

a) Determinar las variables de condición (es decir, de entrada) y especificar los conjuntos borrosos *(«fuzzy sets»)* para describirlas.
b) Determinar las variables de acción (es decir, de salida) y especificar los conjuntos borrosos *(«fuzzy sets»)* para describirlas.
c) Producir la base de reglas.

Es claro que la posición *x(t)* y la velocidad *v(t)* del carro son las variables de entrada, y que la fuerza *F(t)* aplicada al carro es la única variable de salida en este problema. Los conjuntos borrosos que describirán (mediante expresiones lingüísticas) estas variables son los siguientes cinco: negativamente grande (NG), negativamente pequeño (NP), cero (CE), positivamente pequeño (PP) y positivamente grande (PG).

El tercer paso (producción de la base de reglas) puede ser realizado por un experto humano. El uso de expresiones lingüísticas borrosas permite una incorporación más fácil del conocimiento humano. Por ejemplo, parece intuitivo en este problema que la fuerza aplicada debe ser casi siempre dirigida hacia el origen (dado que queremos ubicar el carro en la posición cero) y, una vez que estamos cerca de ese punto, se debe revertir la fuerza para frenar el carro. Las siguientes reglas borrosas expresan este conocimiento:

IF *pos* **IS** PG **THEN** *fuer* **IS** NG
IF *pos* **IS** PP **THEN** *fuer* **IS** NG
IF *pos* **IS** CE **AND** *vel* **IS** PG **THEN** *fuer* **IS** NG
IF *pos* **IS** CE **AND** *vel* **IS** PP **THEN** *fuer* **IS** NG
IF *pos* **IS** CE **AND** *vel* **IS** NP **THEN** *fuer* **IS** PG
IF *pos* **IS** CE **AND** *vel* **IS** NG **THEN** *fuer* **IS** PG
IF *pos* **IS** NP **THEN** *fuer* **IS** PG
IF *pos* **IS** NG **THEN** *fuer* **IS** PG

donde *pos*, *vel* y *fuer* se refieren posición, velocidad y fuerza respectivamente.

El FLC descrito aquí no es el óptimo. Mejoras en la base de reglas o en la interpretación de los términos lingüísticos pueden ser realizados mediante algoritmos genéticos. El trabajo realizado en la Universidad de Málaga para dar solución a este problema hace uso de programación genética para generar bases de reglas que pueden ser usadas como un punto de partida para mejoras posteriores por parte de un experto humano o incluso pueden utilizarse como una solución final.

4.2.5 Enfoque evolucionario

Aquí se brindará un resumen de cómo el equipo de trabajo en la Universidad de Málaga ha utilizado el paradigma de programación genética para resolver el problema dado haciendo uso de reglas borrosas.

4.2.5.1 Dos modos de codificación de una base de reglas

Existen dos enfoques alternativos cuando se debe codificar una base de reglas: el enfoque de Pitt y el enfoque de Michigan. En el primero, cada individuo en la población representa una base entera de reglas que compite con las otras. En el segundo, una única regla es contenida en cada individuo, por lo que la conducta del FLC es determinada por la cooperación entre todos los individuos. Debe tenerse en cuenta que el hecho de seleccionar uno de estos dos enfoques condiciona otros aspectos del algoritmo (por ejemplo: la asignación de aptitud).

En la solución propuesta por el equipo de trabajo en la Universidad de Málaga, se eligió el enfoque de Pitt.

4.2.5.2 Codificación de bases de reglas

Dado que cada individuo contiene una lista de reglas, son necesarios una codificación de reglas individuales y un mecanismo para unirlas.

Una regla individual puede ser fácilmente representada mediante un árbol binario: la raíz es un nodo IF, y las ramas izquierda y derecha representan las condiciones y las acciones, respectivamente. Al mismo

tiempo, tanto las condiciones como las acciones pueden ser expresadas como árboles. Por un lado, una variable unida a un conjunto borroso (es decir, una variable descrita por una expresión lingüística) puede ser representada como un árbol con un nodo raíz IS: el nombre de la variable estará en la rama izquierda y el conjunto borroso estará en la rama derecha. Por otro lado, una conjunción de estas expresiones puede ser representada como un árbol con un nodo raíz AND y dos ramas representando conjunciones anidadas o parejas (variable, conjunto borroso). La siguiente figura muestra un ejemplo:

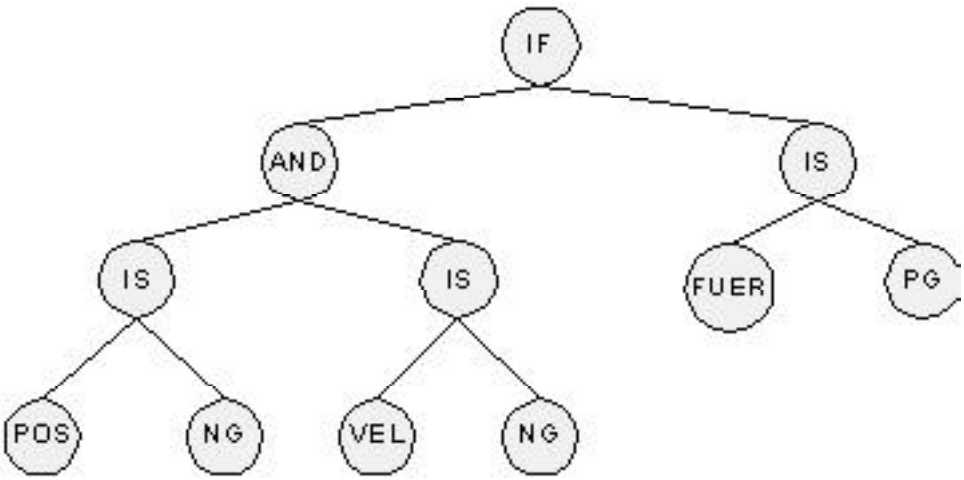

Figura 14. Representación de una regla individual.
IF *pos* **IS** NG **AND** *vel* **IS** NG **THEN** *fuer* **IS** PG

Una lista de reglas puede ser representada como una lista de árboles. Sin embargo, es más apropiado utilizar una representación en un único árbol dado que esto permite a las reglas estar más firmemente conectadas, facilitándose así la propagación de buenos bloques funcionales. Para lograr esto, se requieren dos símbolos adicionales. El símbolo EL *(«empty list»)* representa una lista vacía. El símbolo RLIST combina reglas del mismo modo que el símbolo AND une expresiones. Por lo tanto, una lista de reglas es un árbol con un nodo RLIST y dos ramas representando reglas individuales o listas análogas de reglas.

Ésta constituye una representación flexible y poderosa de las bases de reglas de un FLC. Administra bien el hecho de que un FLC puede poseer reglas con estructuras muy diferentes entre sí, como así también que puede tener un número variable de reglas. Este aspecto contrasta con enfoques basados en algoritmos genéticos en donde la estructura o el número de reglas en el FLC es arbitrariamente prefijado.

4.2.5.3 Necesidad de un sistema de tipos

La flexibilidad de esta representación mediante árboles hacen de la operación genética de cruce una herramienta poderosa dado que permite realizar intercambios en varios niveles: reglas, expresiones, e incluso nombres de variables. Sin embargo, el operador tradicional de cruce que fue descrito en la sección 4.1.6.4 «Operaciones para modificar las estructuras» de este trabajo, puede producir reglas sin sentido. Por ejemplo, consideremos el caso en el cual un nombre de variable es sustituido por una regla completa. Existen tres soluciones posibles para este problema:

a) Definir funciones cerradas, de modo tal que las reglas mal definidas sean reinterpretadas de algún modo predefinido.
b) Reparar el árbol, borrando los subárboles incorrectos y agregando nuevos subárboles si es necesario.
c) Seleccionar puntos de cruce de modo tal que los árboles resultantes sean correctos.

El equipo de la Universidad de Málaga que ha realizado este trabajo, seleccionó la tercera opción con el objetivo de evitar la creación de un alto número de árboles inútiles, como así también de evitar cambios en individuos descendientes debido a mecanismos de reparación.

A cada símbolo se le asigna un atributo «tipo», determinado por las particiones definidas por la relación de equivalencia «es intercambiable con». Por ejemplo, dado que un nodo EL puede ser intercambiado con un nodo RLIST, ambos tienen el mismo tipo. La siguiente figura muestra todos los tipos:

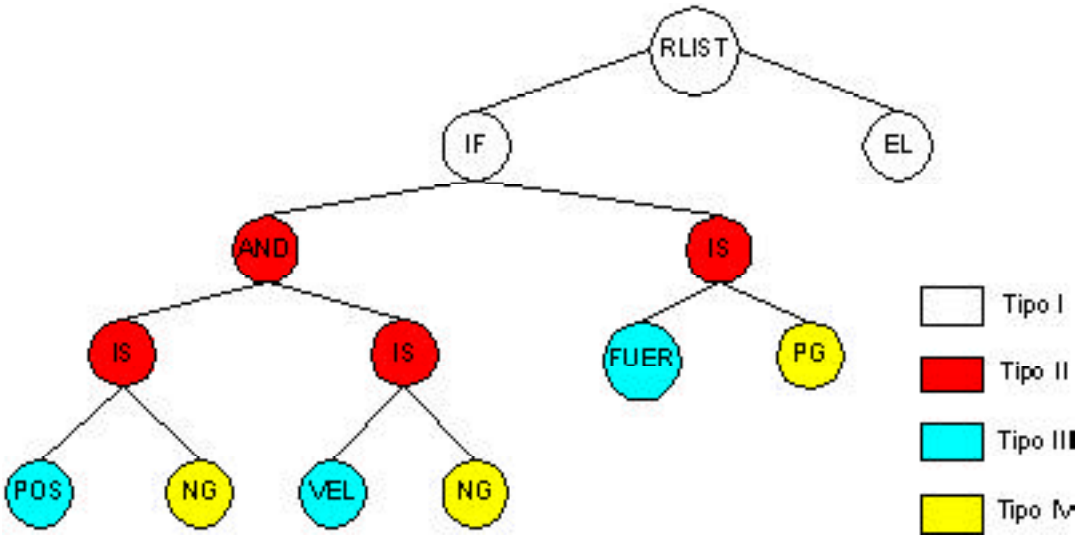

Figura 15. Atributo «tipo» asignado a cada nodo.

La operación de cruce primero selecciona un nodo aleatorio en un padre; luego, un nodo del mismo tipo es aleatoriamente seleccionado en el otro padre y los subárboles correspondientes son intercambiados, generándose así dos descendientes válidos. Sin embargo, si se consideran sólo los tipos mostrados en la figura de arriba, se puede llegar a la creación de árboles sintácticamente correctos pero semánticamente incorrectos, con variables de salida ubicadas en el lado de la condición de la regla y/o variables de entrada ubicadas en el lado de la acción (por ejemplo: **IF** *fuer* **IS** NG **THEN** *pos* **IS** CE). Estas situaciones deben ser evitadas a fin de hacer un buen uso de los recursos computacionales. Para lograrlo, dos subtipos para los tipos II y III son definidos.

Cada subtipo puede ser visto como la variedad del tipo correspondiente que aparece en el lado de la condición de la regla o en el lado de la acción. Por ejemplo, para el tipo VBLE (tipo III), se definen los subtipos VBLEIN y VBLEOUT, los cuales representan las variables de entrada y de salida respectivamente. La siguiente figura muestra la totalidad de los tipos y subtipos para la solución dada a este problema:

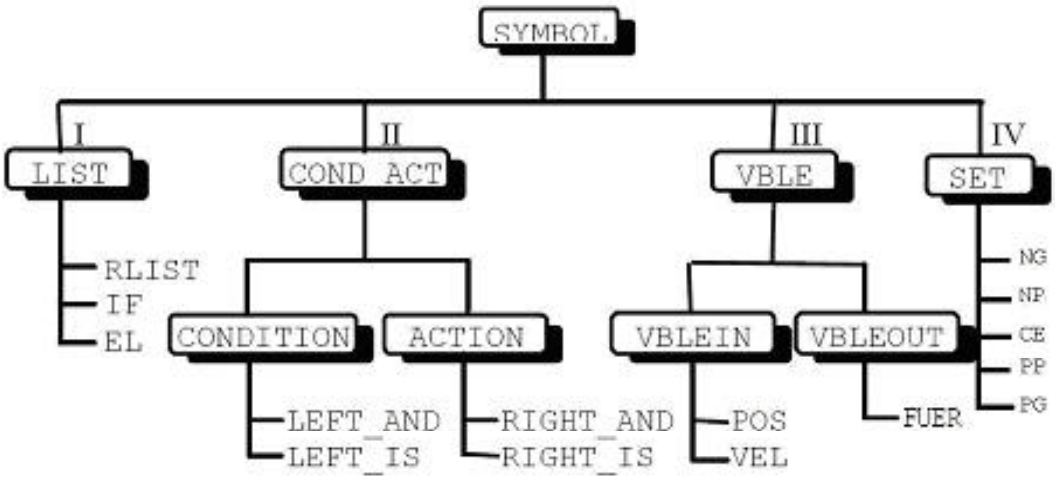

Figura 16. Totalidad de tipos y subtipos utilizados.

4.2.5.4 Generación de bases de reglas

Se define que los árboles pueden poseer una profundidad máxima igual a seis en la generación inicial e igual a quince en las generaciones sucesivas. Inicialmente, la población es generada utilizando el método «ramped half and half» descrito en la sección 4.1.6.2 «Las estructuras iniciales». Se sigue la gramática mostrada a continuación en modo postorden para producir árboles correctos. Como el lector de este trabajo podrá observar, sólo las últimas dos reglas (que describen variables de entrada y de salida) son dependientes del problema. El resto de las reglas puede ser usado en cualquier otro problema borroso.

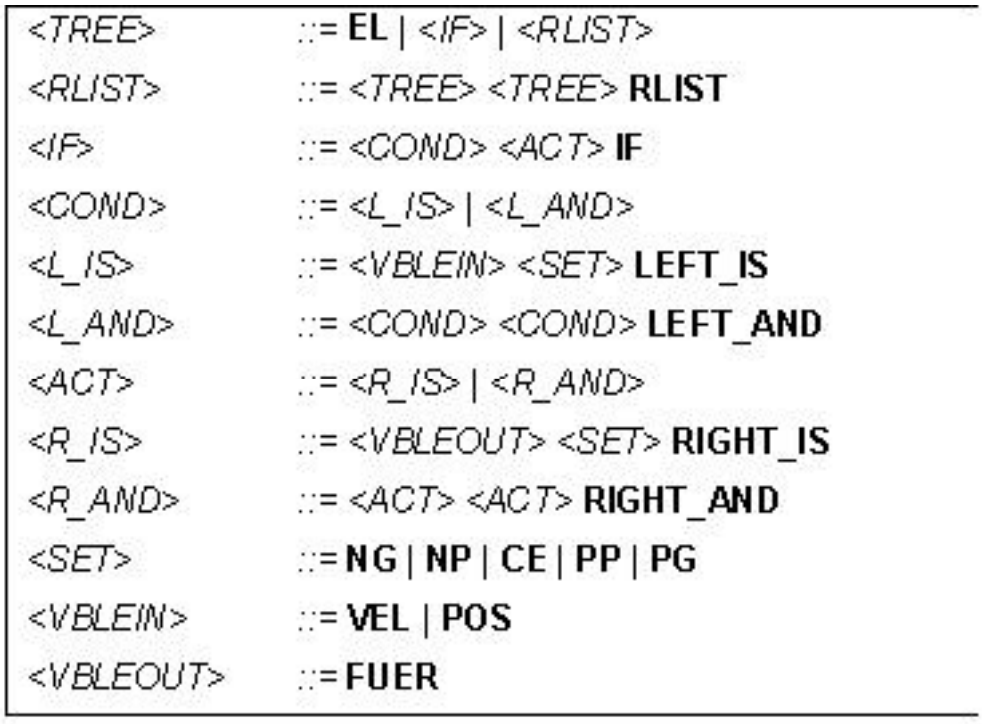

Figura 17. Gramática utilizada para la producción de árboles.

Cada árbol es generado de modo top-down, es decir, eligiendo un nodo raíz y luego produciendo subárboles apropiados de un modo similar. Para generar un nodo, un tipo aleatorio es seleccionado y un símbolo elegido al azar perteneciente a ese tipo es asignado a ese nodo. La selección de este símbolo aleatorio depende del nivel del nodo (por ejemplo, un nodo VBLEIN no puede seleccionarse como raíz de un árbol que representa a un individuo).

4.2.5.5 Evaluación de bases de reglas

Los individuos son entrenados a través de 16 simulaciones del problema (cada una con unos valores iniciales determinados). La aptitud es asignada de acuerdo a la performance obtenida en estos casos de aptitud.

Las simulaciones son ejecutadas para un número máximo de 500 pasos de tiempo cada una. Si el carro no puede ser centrado para una determinada simulación, el número máximo de pasos de tiempo (500) es asignado. El tiempo promedio de todas las simulaciones sobre un individuo es restado a 500 para obtener un valor maximizado de aptitud para dicho individuo.

4.2.6 Resultados

A continuación, el se mostrarán los resultados obtenidos por el grupo de trabajo de la Universidad de Málaga al hacer uso de la solución hasta aquí explicada. Primero se mostrará la performance obtenida haciendo uso de la solución óptima vista en la sección 4.2.3 «Descripción del problema a ser solucionado evolutivamente mediante control borroso» y la obtenida haciendo uso de la solución intuitiva mostrada en la sección 4.2.4 «Diseño de un controlador de lógica borrosa».

4.2.6.1 Solución óptima y solución intuitiva

Para medir la performance obtenida al utilizar un FLC que hace uso de la solución óptima y al utilizar otro FLC que utiliza la solución intuitiva, se utiliza la cantidad promedio de pasos de tiempo demorados para centrar el carro en 100 simulaciones del problema, comenzando cada una en puntos iniciales aleatorios en el dominio [-2.5 ; 2.5] (es decir, el carro comienza cada simulación estando ubicado sobre un punto aleatorio de ese intervalo). Se excluyen las situaciones en las cuales el estado inicial es también una solución. Como se dijo en la sección 4.2.3 «Descripción del problema a ser solucionado evolutivamente mediante control borroso», los parámetros de la simulación son ?? = 0.02 segundos, m = 2.0 kg. La fuerza F posee una magnitud de valor máximo 2.5 Newtons. Las simulaciones son ejecutadas para un número máximo de 500

pasos de tiempo. El carro es considerado centrado cuando max (|pos|, |vel|) < 0.5.

Los resultados obtenidos son los siguientes: la solución óptima promedió 129 pasos de tiempo para centrar el carro, mientras que la solución intuitiva promedió 249 pasos de tiempo, excediendo en 14 ocasiones el número máximo de pasos de tiempo permitidos.

4.2.6.2 Solución dada por programación genética

10 ejecuciones del sistema de programación genética fueron llevadas a cabo. Se seleccionó un tamaño de población de 500 individuos para un número máximo de 51 generaciones (es decir, la generación inicial más 50 adicionales). La selección de progenitores fue en un 50% proporcional a la aptitud (haciendo uso del método de selección elitista descrito en el apéndice «Métodos de selección») y en un 50% al azar. La siguiente figura muestra la mejor aptitud y la aptitud promedio para cada generación, promediadas para las 10 ejecuciones.

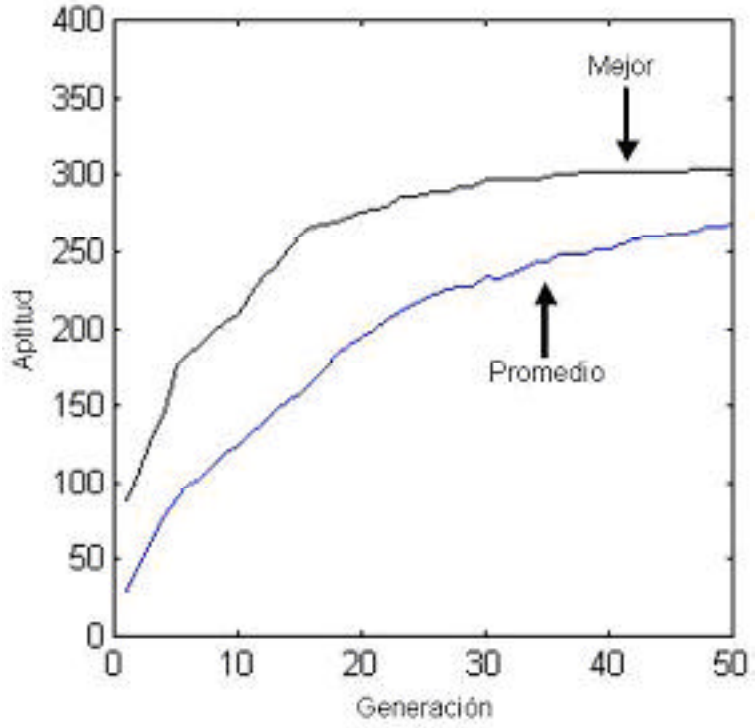

Figura 18. Promedio (para 10 ejecuciones) de mejor aptitud y de aptitud promedio en cada generación.

El mejor individuo generado obtuvo un valor de aptitud de 321 (promediando 158 pasos de tiempo para centrar el carro). Dicho individuo corresponde al siguiente FLC de 9 reglas:

1) **IF** *pos* **IS** PG **THEN** *fuer* **IS** NG
2) **IF** *pos* **IS** PP **THEN** *fuer* **IS** NG
3) **IF** *pos* **IS** NG **THEN** *fuer* **IS** PG
4) **IF** *vel* **IS** PG **THEN** *fuer* **IS** NG
5) **IF** *vel* **IS** NP **THEN** *fuer* **IS** PG
6) **IF** *vel* **IS** NG **THEN** *fuer* **IS** PG
7) **IF** *pos* **IS** CE **AND** *vel* **IS** PP **THEN** *fuer* **IS** NG
8) **IF** *pos* **IS** NP **AND** *vel* **IS** CE **THEN** *fuer* **IS** PG
9) **IF** *pos* **IS** PP **AND** *pos* **IS** PG **AND** *vel* **IS** CE **AND** *vel* **IS** NP **THEN** *fuer* **IS** NG **AND** *fuer* **IS** NP **AND** *fuer* **IS** PG

La base de reglas aquí mostrada fue obtenida luego de borrar reglas duplicadas. Las reglas 2, 3 y 6 aparecieron dos veces y las reglas 1 y 4 estaban repetidas tres veces. Esto indica que son bloques funcionales importantes para el FLC. El mejor individuo también contenía dos reglas sin valor alguno cuyos lados de condición eran equivalentes a un conjunto borroso vacío, por lo que no tenían influencia en la salida.

5. Solución propuesta

La estrategia de control puede ser vista como un programa algorítmico de computadora que, en función de las entradas representadas por valores cualesquiera de las variables de estado *x(t)* y *v(t)*, determine el valor de la variable de control (es decir, determine qué cohete activar) con el fin de llevar el sistema a su estado objetivo. Se propone crear dicho programa sin intervención humana, mediante programación genética aplicada a la creación evolutiva de programas.

Un denominado «motor genético» será el encargado de implementar el paradigma de la programación genética para generar dicho programa algorítmico. El motor genético fue creado por el autor de este trabajo en el lenguaje C de programación, con el objetivo de dar solución al problema anteriormente definido. Este motor será el encargado de generar una población inicial de individuos (programas), medir sus aptitudes respectivas, crear nuevas generaciones de individuos evolucionados (programas de mayor aptitud) mediante operaciones genéticas, y designar el resultado final de la ejecución.

A continuación veremos los lineamientos sobre los cuales se basa la ejecución del motor genético programado para este trabajo, en términos de las siguientes características:

1. Conjunto de funciones y de terminales.
2. «*Wrapper*» (envolvedor) utilizado.
3. Medida de aptitud *(fitness)*.
4. Parámetros para controlar la ejecución del motor genético.
5. Generación de población inicial.
6. Cálculo de aptitud cruda de individuos.
7. Operadores genéticos utilizados.
8. Métodos de selección utilizados.
9. Criterios para terminar la ejecución.
10. Método para designar un resultado.
11. Modo de retorno del individuo solución.

5.1 Conjunto de funciones y de terminales

Como ya se dijo anteriormente en este trabajo, la solución de tiempo óptimo para este problema es conocida y consiste en, dada cualquier posición *x(t)* y cualquier velocidad *v(t)*, aplicar la fuerza *F* para acelerar el carro en la dirección positiva si

$$-x(t) > \frac{v(t)^2 \, Signo(v(t))}{2|F|/m}$$

o, en caso contrario, aplicar la fuerza *F* para acelerarlo en la dirección negativa. La función signo retorna +1 si su argumento es positivo y -1 en otro caso.

En la sección 3 «Definición del problema» establecimos que la masa del carro es igual a 2 kg y que la fuerza de cada cohete posee una magnitud fija de 1 Newton. Como podemos observar, en virtud de estos valores, el denominador *2|F|/m* es igual a 1.0 y puede así ser ignorado.

La estrategia de control óptima anteriormente mencionada puede ser entonces codificada en pseudocódigo del siguiente modo:

```
si (-1.0 * x > v * ABS(v))
    entonces
controlador = 1.0
    sino
controlador = -1.0,
```

donde ABS es la función valor absoluto.

Ahora bien, el motor genético simula los procesos de evolución en la naturaleza sobre una población de programas (individuos) que son Expresiones-S de LISP (es decir, programas codificados en lenguaje LISP). Más allá del lenguaje elegido y de sus características particulares, debemos ser concientes de que un programa de computadora es simplemente una composición de varias funciones actuando sobre varios argumentos. Para el caso de la estrategia de control óptima anteriormente presentada, su Expresión-S en LISP es la siguiente:

(> (* -1 X) (*V (ABS V)))

La función ABS es una función de un solo argumento que retorna el valor absoluto del mismo.

En esta Expresión-S, la función mayor que «>» es una función numérica de dos argumentos que retorna +1 si su primer argumento es mayor que el segundo y que retorna 0 en caso contrario.

En la siguiente figura se muestra el árbol correspondiente a esta Expresión-S:

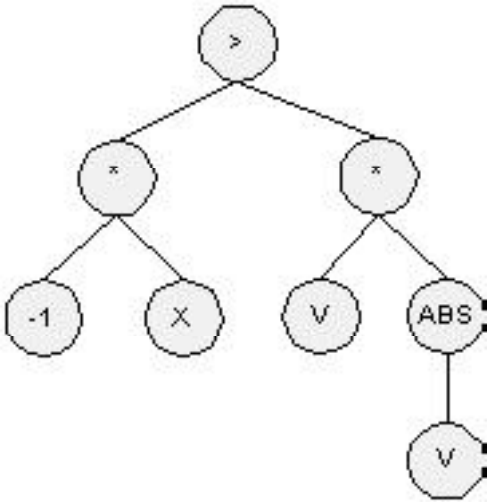

Figura 19. Expresión-S de la estrategia óptima.

Como podemos observar, el conjunto de terminales requeridos para este programa LISP que constituye la solución óptima es representado por el conjunto T = {X, V, -1}. A su vez, el conjunto de funciones está representado por F = {>, *, ABS}.

En consecuencia, el motor que ejecutará la programación genética podría cargarse sólo con este conjunto de terminales y de funciones. Hacer esto no sólo sería correcto, sino también sería la elección más eficiente, ya que los elementos incluidos en ambos conjuntos son los únicos necesarios para resolver óptimamente el problema.

Ahora bien, si acotáramos el conjunto de funciones sólo a las funciones nombradas anteriormente, estaríamos obviamente desvirtuando la prueba práctica que este trabajo propone realizar para dar un sustento empírico a lo dicho en forma teórica acerca del paradigma de la programación genética. En dicho caso, la prueba se desvirtuaría debido a que estaríamos haciendo un uso excesivo de los conocimientos que nos brinda conocer la solución. De ese modo, la prueba sería sólo una comprobación de que la programación genética opera correctamente en este problema. La situación ideal para demostrar la potencia de la programación genética sería cargar el conjunto de funciones con todas las funciones existentes, a fin de lograr la mayor abstracción de dicho paradigma respecto al problema puntual que pretende resolver. Sin embargo, el lector podrá darse cuenta por sí mismo que hacer esto sería un enfoque totalmente ineficiente.

Por lo tanto, se considera que lo más apropiado es encontrar un punto medio entre ambos extremos. Esto se traduce en colocar dentro del conjunto de funciones a aquellas que ya sabemos son parte de la solución óptima, como así también otras que no tienen utilidad alguna en este problema. Siguiendo el lineamiento expuesto, el conjunto de terminales con los cuales será ejecutado el motor genético será T = {X, V, -1}. En cuanto al conjunto de funciones, se realizarán distintas pruebas con el motor genético para analizar el comportamiento del mismo haciendo uso de tres diferentes conjuntos de funciones. Estos son:

1) El conjunto de funciones formado por las únicas funciones que conforman la solución óptima al problema. Son en total 3 funciones:

   F = {*, >, abs}

2) El conjunto de funciones formado por las únicas funciones que conforman la solución óptima al problema más otras 6 que no tienen utilidad en este problema (operadores aritméticos de suma, de resta, de división; operadores relacionales «menor que», «menor o igual que», «mayor o igual que»). Son en total 9 funciones:

F = {+, -, *, /, <, >, <=, >=, abs}

3) El conjunto de funciones formado por las anteriores funciones más otras 6 que no tienen utilidad en este problema (operadores lógicos *and*, *or*; operadores relacionales «igual que», «distinto que»; seno, coseno). Son en total 15 funciones:

F = {+, -, *, /, <, >, <=, >=, abs, and, or, ==, !=, sin, cos}

Los resultados obtenidos haciendo uso de los distintos conjuntos de funciones, como así también el análisis de los mismos, se encuentran en la sección 6 «Resultados experimentales».

El motor genético redefine algunas de estas funciones de modo tal de satisfacer la propiedad de cierre. Recordemos que la propiedad de cierre se refiere a que cada una de las funciones en el conjunto de funciones sea capaz de aceptar, como sus argumentos, cualquier valor y tipo de dato que sea posiblemente retornado por cualquier función en el conjunto de funciones, y que sea también capaz de aceptar, como sus argumentos, cualquier valor y tipo de dato que sea posiblemente asumido por cualquier terminal en el conjunto de terminales. Esto significa que cada función en el conjunto de funciones debe estar definida y cerrada para cualquier combinación de argumentos que encuentre.

El motor genético define los operadores relacionales siguiendo una lógica basada en números, de modo tal que el valor devuelto por los mismos sea 1 (verdadero) o 0 (falso), dependiendo de su valor de verdad. Asimismo, el operador aritmético de división ha sido redefinido en el motor genético de modo tal que retorne el número 1 en caso de un intento de división por 0. Mediante estas definiciones, se logra la satisfacción de la propiedad de cierre, la cual es indispensable para el funcionamiento del motor genético.

5.2 *«Wrapper»* (envolvedor) utilizado

La Expresión-S que constituye la solución óptima para este problema retornará siempre 1 o 0, según el valor de verdad de la función «mayor que», como se explicó anteriormente. Ahora bien, otras Expresiones-S pueden retornar cualquier valor (por ejemplo: la división de dos números, el producto de dos números, etc).

La variable de control del problema del carro es binaria. Es decir, puede tomar sólo 1 o -1 como valores, lo que indicará si el cohete a activar es el de la izquierda o el de la derecha, respectivamente. En consecuencia, es necesario definir un modo por el cual se asocie cualquier valor retornado por una Expresión-S a sólo uno de los valores 1 o -1.

Para lograr esto se utiliza un denominado *«wrapper»* o «envolvedor». Su funcionamiento es muy sencillo: convierte cualquier valor positivo retornado por una Expresión-S en 1, y convierte todos los demás valores (negativos o cero) en -1.

Por ejemplo, para el caso de la Expresión-S óptima anteriormente comentada, si la misma evalúa a 1, el *«wrapper»* definirá que la variable de control debe poseer valor 1, activando así el cohete izquierdo; si dicha Expresión-S evalúa a 0, el *«wrapper»* asignará el valor -1 a la variable de control, activando así el cohete derecho del carro.

5.3 Medida de aptitud *(fitness)*

La aptitud cruda o *«raw fitness»* es aquella que expresa la aptitud de un individuo en la terminología natural del problema. Para el caso de nuestro problema de centrado del carro, un individuo (programa) será más apto cuantos menos movimientos realice para ubicar y dejar en reposo al carro en el punto cero. Por lo tanto, la aptitud de un individuo se define en el motor genético como la sumatoria de la cantidad de movimientos que la simulación del individuo (la estrategia de control) realiza sobre el carro para llevarlo al estado objetivo para cada caso de aptitud definido por el usuario.

Muchos individuos (sobre todo en las poblaciones más tempranas) serán incapaces de llevar el sistema a su estado objetivo. Para evitar que esto impida el correcto funcionamiento del motor genético, el usuario del motor debe definir la cantidad máxima de movimientos por caso de aptitud que se permite realizar a cada individuo (es decir, a cada estrategia de control) sobre el carro para llevar el sistema a su objetivo.

Para cada individuo y para cada caso de aptitud, el motor genético realiza una simulación de cómo dicho individuo (estrategia de control) mueve el carro sobre su camino unidimensional (partiendo de las condiciones iniciales –valores específicos de *x(t)* y *v(t)*- correspondientes al caso de aptitud ejecutado), y computa como aptitud cruda la cantidad de movimientos que realiza la estrategia de control (individuo) seleccionada sobre el carro para llevarlo a su estado objetivo, si es que lo logra. Si no lo logra, se le asigna como aptitud cruda la cantidad de movimientos límite de ejecución definido por el usuario más uno. ¿Por qué se le adiciona un uno a dicha cantidad? El motivo es sencillo. Supongamos que la cantidad de movimientos máxima permitida para centrar el carro en un caso de aptitud sea 1000. Si la estrategia de control de la cual

se pretende obtener su aptitud centra el carro en reposo en el movimiento número 1000, su aptitud será obviamente 1000. Si otra estrategia no logra centrar el carro en 1000 movimientos, entonces se le asignará 1001 como aptitud, a fin de indicar que la misma no logró su propósito. La aptitud cruda final de cada individuo será la sumatoria de sus aptitudes crudas para cada caso de aptitud.

Obviamente, cuanto más cercana a cero sea la aptitud de un individuo, más apto será el mismo.

5.4 Parámetros para controlar la ejecución del motor genético

- La profundidad máxima permitida para los individuos de la población inicial se define en 6.
- La profundidad máxima de los individuos descendientes en cada generación (mediante la operación genética de cruce) se define en 17. Este límite impide el uso de grandes cantidades de tiempo de computación en algunos pocos individuos extremadamente grandes.
- A la hora de aplicar un operador genético, la probabilidad de reproducción es del 10% y la probabilidad de cruce es del 90%.
- La cantidad máxima de individuos por generación es ingresada por el usuario.
- La cantidad máxima de generaciones a procesar es ingresada por el usuario.
- La cantidad de casos de aptitud es ingresada por el usuario, junto con sus respectivos valores iniciales (posición *x(t)* y velocidad *v(t)* del carro) para cada una.
- La cantidad máxima de movimientos que puede realizar cada individuo (estrategia de control) para llevar el sistema al estado objetivo es, como ya se dijo anteriormente, ingresada por el usuario.
- La cantidad máxima de generaciones sucesivas en las que puede repetirse el mejor individuo es ingresada por el usuario.

5.5 Generación de población inicial

La población inicial se genera creando una cantidad de individuos especificada por el usuario, donde cada individuo está representado por un árbol (el cual es una forma de representación de una Expresión-S de LISP).

Estos árboles son creados con formas variadas y con una profundidad máxima igual a 6, como ya se dijo anteriormente. Esto significa que la longitud del camino entre la raíz del árbol y una hoja (terminal) cualquiera no es mayor a 6. Las formas variadas se logran haciendo la selección aleatoria de etiquetas desde el conjunto combinado C = F U T (el cual consiste en la unión de los conjuntos de funciones y de terminales) para nodos en profundidades menores a la máxima permitida, al mismo tiempo que se restringe la selección aleatoria de etiquetas desde el conjunto de terminales para nodos en la máxima profundidad permitida.

Los individuos duplicados en la generación aleatoria inicial son improductivos. Desperdician recursos computacionales y reducen la diversidad genética de la población [1]. En consecuencia, el motor genético desarrollado para este trabajo impide duplicados en la generación inicial. Si en la población inicial se crea un individuo idéntico a uno anteriormente generado, el mismo es eliminado y se repite el proceso hasta crear un individuo distinto a todos los ya creados.

5.6 Cálculo de aptitud cruda de individuos

Una vez generada la población inicial, se calcula la aptitud cruda de cada individuo de la cual se compone mediante la simulación anteriormente explicada en la subsección 5.3 «Medida de aptitud (fitness)». Una vez que ésta es obtenida, se determina si se satisface alguno de los dos criterios para terminar la ejecución de la programación genética (dichos criterios son explicados en la subsección 5.9 «Criterios para terminar la ejecución»). Si ninguno de estos criterios son satisfechos, comienza el ciclo evolutivo mediante la producción de una población descendiente (la siguiente generación).

5.7 Operadores genéticos utilizados

Para generar la población descendiente se hace uso de dos operaciones genéticas: la operación de reproducción Darwiniana y la operación de cruce.

Recordemos los siguientes conceptos que ya vimos anteriormente en este trabajo. La operación de reproducción es asexual ya que opera sobre un único individuo y produce sólo un descendiente cada vez que es ejecutada. La misma consiste en dos pasos. Primero, una Expresión-S es elegida de la población de acuerdo a un determinado criterio de selección. Luego, el individuo seleccionado es copiado, sin alteración alguna, desde la población actual a la nueva población. Para el caso de la operación de cruce, recordemos que la misma crea variación genética en la población a través de la producción de dos nuevos individuos descendientes a partir de partes tomadas de cada uno de sus dos padres.

A fin de maximizar la diversidad genética, el motor genético ejecuta la operación de reproducción con una probabilidad del 10%, mientras que la probabilidad de la operación de cruce es del 90%.

Un tamaño máximo permitido (expresado en términos de la profundidad del árbol) es establecido para cada descendiente creado mediante la operación de cruce. Como ya se dijo anteriormente, este valor es igual a 17. Para respetar este lineamiento, el motor genético opera del siguiente modo. Si el cruce entre dos padres crea un descendiente de tamaño no permitido, la operación de cruce es abortada para dicho descendiente y el primero de sus padres es seleccionado arbitrariamente para ser reproducido en la nueva población. En este caso, el otro descendiente producto del cruce es ingresado normalmente en la nueva población. Ahora bien, si los dos descendientes poseen un tamaño no permitido, el motor genético aborta la operación de cruce para ambos descendientes y ambos padres son reproducidos en la nueva población.

El motivo por el cual no se hace uso de mutación en el motor genético creado para este trabajo deriva de lo expresado en la sección 4.1.7 «Convergencia de la población en la programación genética». Allí habíamos establecido que, en el algoritmo genético convencional, cuando un individuo se reproduce incestuosamente consigo mismo, los dos descendientes resultantes son idénticos, lo cual fortalece la tendencia hacia la convergencia en el algoritmo genético. Es allí donde es útil la operación de mutación, la cual permite desarrollar una búsqueda exploradora (a lo ancho) que se encarga de explorar nuevos dominios en busca de mejores soluciones, proporcionando así una garantía de accesibilidad para todos los puntos del espacio de búsqueda. La mutación puede, en principio, llevar a cualquier parte; sin embargo, en la práctica, la población rápidamente reconverge. Lo dicho hasta aquí es válido para el caso de los algoritmos genéticos. En contraste, en el caso de la programación genética mientras que la operación Darwiniana de reproducción crea una tendencia hacia la convergencia, la operación de cruce genera una presión contrapuesta que aleja de la misma. Esto se debe a que cuando un individuo (programa) se reproduce consigo mismo, los dos descendientes resultantes serán, en general, distintos (excepto en el caso infrecuente en el que los puntos de cruce para ambos individuos sean los mismos). Es por esto que se concluye que la convergencia de la población es improbable en la programación genética, y es este el motivo por el cual el motor genético no hace uso de la operación de mutación.

5.8 Métodos de selección utilizados

Tanto para el caso de la reproducción como así también para el del cruce, el motor genético creado para este trabajo selecciona los progenitores haciendo un uso combinado del método de selección elitista y del método de selección por ruleta por bondad (véase el apéndice *Métodos de selección*) del modo que se explica en el siguiente párrafo.

Cuando se crea una nueva generación, el mejor individuo presente en la generación actual es automáticamente reproducido (es decir, copiado sin cambios) en la nueva población (es aquí donde se aplica elitismo), a fin de no correr el riesgo de perderlo. Para el resto de los descendientes, sus progenitores son seleccionados mediante el método de selección por ruleta por bondad. Recordemos que esta última es una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. Por lo tanto, los valores de la función bondad se utilizan como probabilidades de selección. Ahora bien, el lector atento puede preguntarse cómo puede ser esto posible si anteriormente habíamos establecido que un individuo es más apto cuanto más cercana a cero se encuentre su aptitud. De acuerdo a este razonamiento, un individuo más apto (de aptitud más cercana a cero) tendría menor probabilidad de ser elegido como progenitor. Obviamente este no es el caso, ya que lo que se utiliza como probabilidad de selección no es la aptitud cruda *(raw fitness)*, sino la aptitud normalizada *(normalizad fitness)*. Por lo tanto, el motor genético, además de calcular la aptitud cruda de cada individuo, calcula su aptitud normalizada (del modo descrito en la sección 4.1.6.3 «Aptitud»), a fin de poder utilizar el método de selección por ruleta por bondad. La aptitud normalizada constituye la probabilidad de que el individuo sea seleccionado como progenitor en las operaciones de reproducción o de cruce. Recordemos que la aptitud normalizada varía entre 0 y 1 para cada individuo, es mayor para mejores individuos en la población, y que la suma de todas las aptitudes normalizadas de todos los individuos de una población cualquiera es 1.

Una vez finalizada la producción de la nueva generación, la aptitud cruda de los individuos que pertenecen a la misma es calculada. Si no se satisface algunos de los dos criterios para terminar la ejecución (explicados en la siguiente subsección), el proceso hasta aquí descrito itera hasta satisfacer una de ellas.

5.9 Criterios para terminar la ejecución

El motor genético creado para este trabajo implementa dos criterios para la terminación de la ejecución. La satisfacción de uno de estos dos criterios finalizará la ejecución:

1. cuando se alcance el número máximo de generaciones (el cual es ingresado por el usuario al iniciar el motor genético),
2. cuando el mejor individuo en cada generación sea sucesivamente el mismo (no haya cambiado) a lo largo

de una determinada cantidad de generaciones. Mediante este criterio, se impide que el motor siga funcionando cuando es aparente que ha convergido a una solución.

5.10 Método para designar un resultado

Si la terminación de la ejecución es generada por el alcance del número máximo de generaciones, se retornará el mejor individuo presente en la última generación.

Si el criterio de terminación es alcanzado porque el mejor individuo fue el mismo a lo largo de una determinada cantidad de generaciones sucesivas, dicho individuo será retornado como resultado.

5.11 Modo de retorno del individuo solución

Cabe aquí aclarar el modo en el cual el motor genético devuelve el individuo resultado. El motor imprime por pantalla la Expresión-S que representa a dicho individuo y, al mismo tiempo, genera un archivo de simulación de extensión *.sim* para cada uno de los casos de aptitud que dicho individuo logra resolver. Estos archivos de simulación pueden ser abiertos luego mediante el programa de simulación creado por el autor de este trabajo, el cual puede ser accedido desde la interfaz gráfica del motor genético. A través de este programa, el usuario podrá observar gráficamente una simulación de cómo la estrategia de control resultante del motor genético mueve el carro (desde las condiciones iniciales de posición y velocidad definidas en el caso de aptitud correspondiente) hasta dejarlo en el estado objetivo.

6. Resultados experimentales

En la sección 5.1 «Conjunto de funciones y de terminales» se estableció que se realizarían pruebas para analizar el funcionamiento del motor genético haciendo uso de tres conjuntos de funciones distintos: uno de ellos es el formado sólo por las funciones que participan en la solución óptima del problema; los otros dos conjuntos incluyen funciones que no tienen utilidad en la misma.

A continuación veremos los resultados obtenidos en cada una de estas pruebas.

6.1 Resultado experimental número uno

6.1.1 Conjunto de funciones utilizado

En esta primera prueba, se hará uso del conjunto de funciones F = {+, -, *, /, <, >, <=, >=, abs}. Es decir, a las tres funciones que forman parte de la solución óptima (*, >, abs), se le agregan 6 funciones que no tienen utilidad en este problema.

6.1.2 Parámetros ingresados en el motor genético

Se cargó la interfaz del motor genético con los siguientes parámetros de entrada:

1. Cantidad de individuos en cada generación: 50.
2. Cantidad de generaciones: 50.
3. Cantidad de casos de aptitud a simular para cada individuo: 4.
4. Cantidad máxima de movimientos que se permite demorar a cada estrategia de control para que la misma centre y deje en reposo al carro para cada caso de aptitud: 120.000.
5. Cantidad máxima de generaciones en las cuales puede repetirse el mejor individuo: 15.

Cada caso de aptitud se compone de las siguientes condiciones iniciales:

	Posición inicial del carro	*Velocidad inicial del carro*
Caso de aptitud nº 1	2.000 m	100 m/s
Caso de aptitud nº 2	-1.903 m	-102 m/s
Caso de aptitud nº 3	1.850 m	-87 m/s
Caso de aptitud nº 4	-1.744 m	76 m/s

Tabla 2. Casos de aptitud.

Como se puede observar, se seleccionaron casos de aptitud de modo tal que se cubriesen los siguientes posibles casos: que el carro se encuentre en el eje positivo con velocidad positiva, que se encuentre en el eje positivo con velocidad negativa, que se encuentre en el eje negativo con velocidad positiva, que se encuentre en el eje negativo con velocidad negativa.

Aquí tenemos la interfaz del motor genético ya cargada con los valores anteriormente comentados:

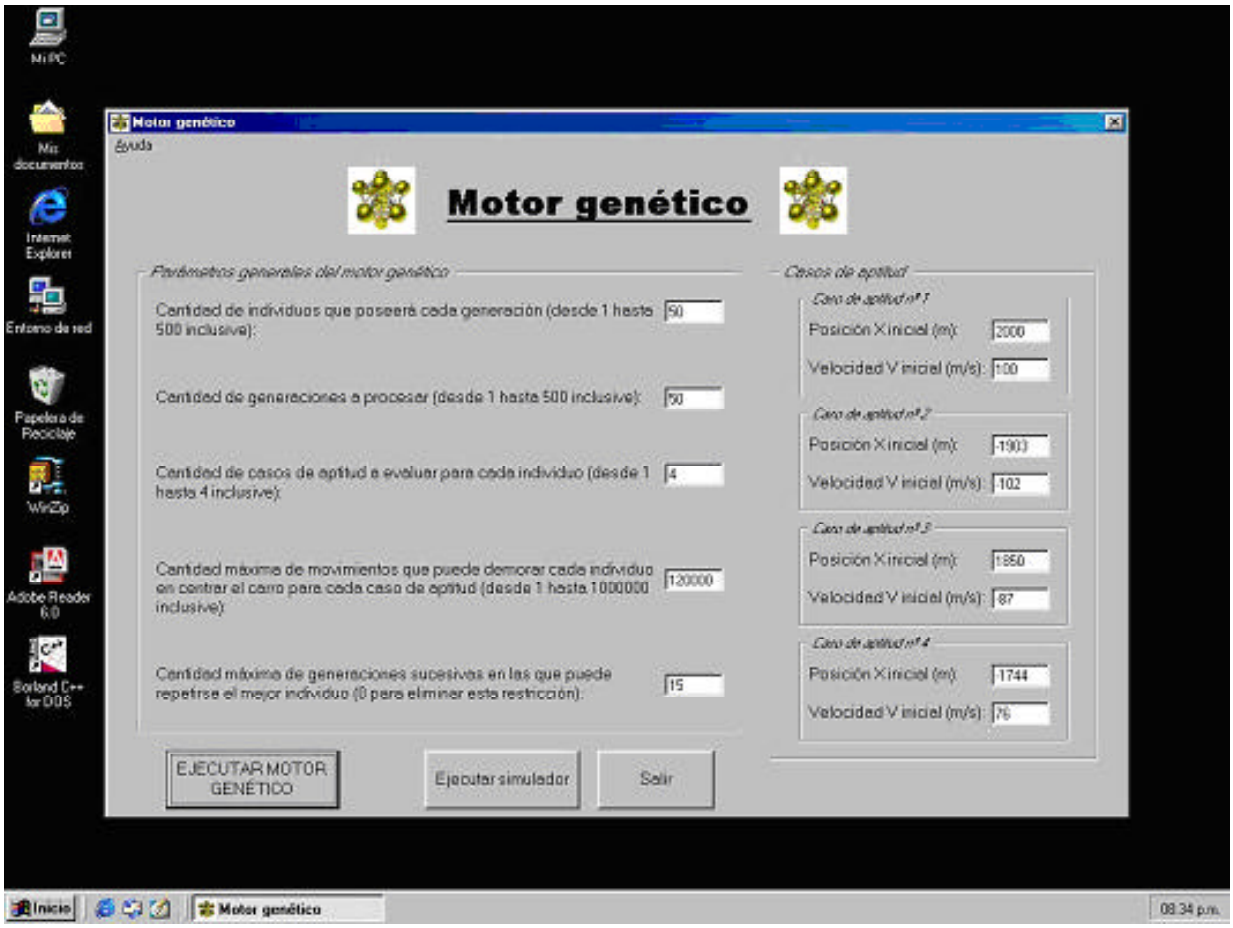

Figura 20. Interfaz del motor genético.

6.1.3 Resultado final emitido por el motor

Luego de aproximadamente dos horas de procesamiento (en una computadora con procesador AMD Athlon XP 2600+ de 2,08 Ghz), el motor genético finalizó su ejecución y emitió el resultado que se ve en la siguiente pantalla, terminando así su ejecución:

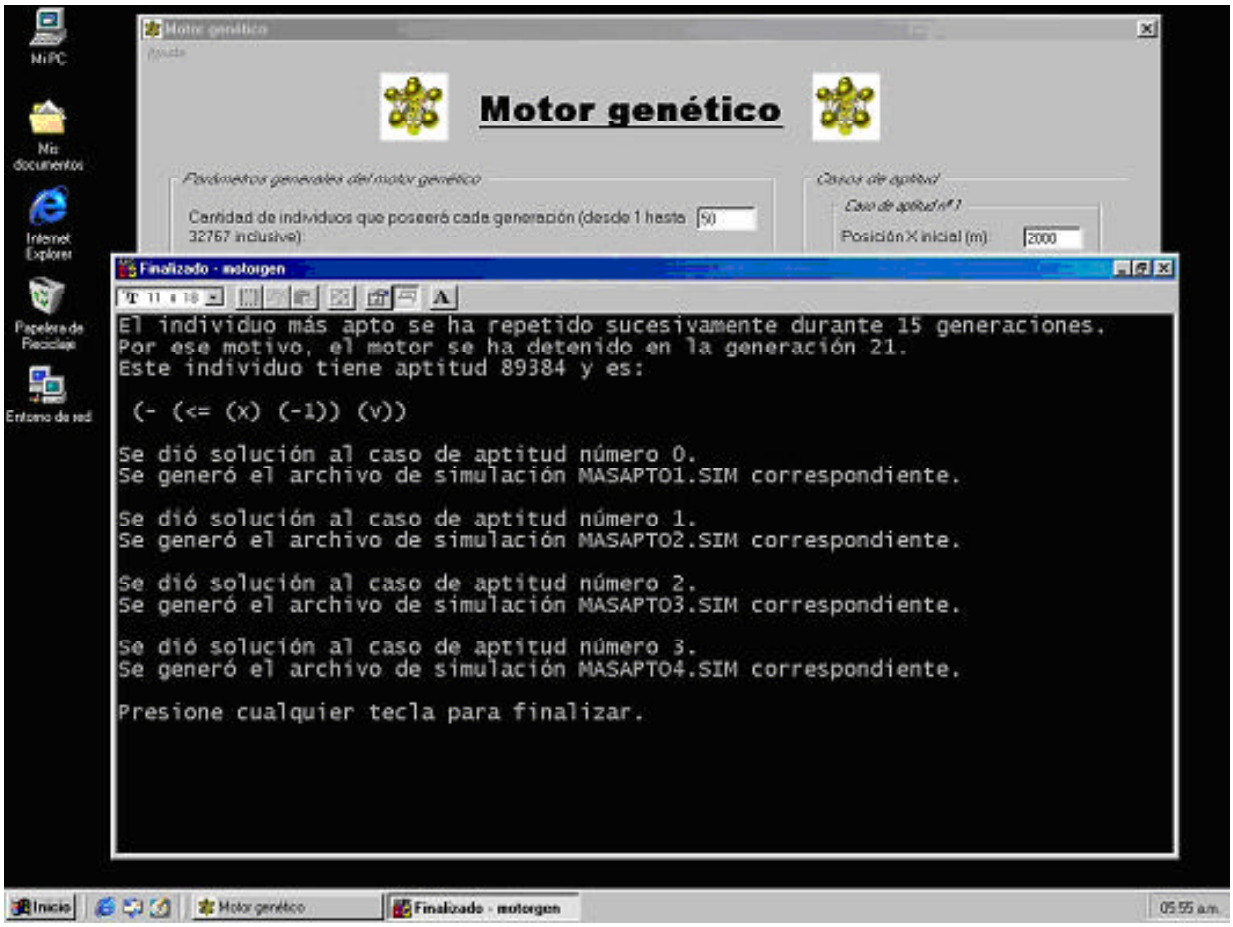

Figura 21. Resultado final emitido por el motor.

A continuación se analizarán los resultados mostrados en esta pantalla. El motor genético actuó sobre un total de 22 generaciones (la generación inicial más 21 adicionales). Se detuvo debido a la existencia de un individuo que se repitió como el mejor de su generación a lo largo de 15 generaciones consecutivas (recordemos que éste es uno de los criterios de parada del motor genético).

La aptitud del individuo (estrategia de control) resultante es igual a 89.384. Tal como informa la pantalla, esta estrategia de control logra llevar el carro a su estado objetivo (es decir, logra situar el carro en x=0 con v=0) en todos los casos de aptitud que se han definido. Esto significa que el individuo logra llevar el carro al estado objetivo para todas las duplas (posición inicial; velocidad inicial) definidas en cada caso de aptitud. Que su aptitud sea igual a 89.384 significa que la estrategia de control resultante demora, para la sumatoria de movimientos realizados en cada uno de los cuatro casos de aptitud anteriormente establecidos, 89.384 movimientos para dejar el carro centrado (x=0) en reposo (v=0). Esto nos indica que esta estrategia de control mueve el carro un promedio de 89.384 / 4 = 22.346 veces en cada caso de aptitud (para llevar el carro a su estado objetivo).

La Expresión-S que representa a este individuo es (- (<= (x) (-1)) (v)). La misma puede ser representada mediante la siguiente estructura de árbol (de hecho, este individuo, como así también todos los otros individuos manipulados por el motor genético, son representados en el mismo como árboles):

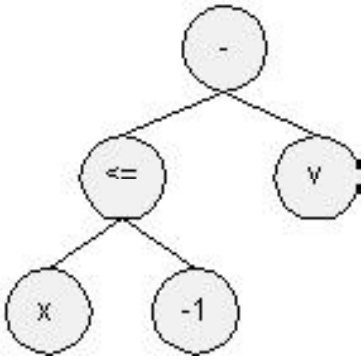

Figura 22. Expresión-S retornada por el motor genético.

6.1.4 Evolución de aptitud a lo largo de las generaciones

A continuación, se hará un análisis que resulta muy interesante para evaluar el funcionamiento del paradigma de la programación genética implementada en este motor genético: se analizará cómo han evolucionado los individuos a lo largo de las distintas generaciones. Para satisfacer este objetivo, se mostrará, para cada generación, la aptitud del individuo menos apto de la población, la aptitud del individuo más apto de la población, y la aptitud promedio de la misma. Así podrá observarse cómo evolucionaron las aptitudes de los individuos a lo largo de las distintas generaciones.

A continuación presentamos estos datos en forma de tabla:

Número de generación	*Aptitud del individuo menos apto*	*Aptitud del individuo más apto*	*Promedio de aptitud de todos los individuos*
0	480.004	304.834	476.500,59
1	480.004	304.834	465.990,41
2	480.004	304.834	465.990,41
3	480.004	304.834	458.983,59
4	480.004	89.400	454.674,91
5	480.004	89.400	450.366,25
6	480.004	89.384	438.245,16
7	480.004	89.384	404.458,31
8	480.004	89.384	404.458,97
9	480.004	89.384	364.751,91
10	480.004	89.384	373.208,72
11	480.004	89.384	303.863,59
12	480.004	89.384	337.811,00
13	480.004	89.384	313.408,59
14	480.004	89.384	310.225,69
15	480.004	89.384	314.534,38
16	480.004	89.384	253.966,84
17	480.004	89.384	266.532,59
18	480.004	89.384	257.754,69
19	480.004	89.384	195.254,84
20	480.004	89.384	159.696,88
21	480.004	89.384	183.133,77

Tabla 3. Evolución de aptitud a lo largo de las generaciones.

Un valor de aptitud de 480.004 indica que el individuo no logra centrar el carro en ninguno de los casos de aptitud establecidos (dicho número proviene de realizar la sumatoria de los 120.001 movimientos realizados sobre el carro -sin éxito- para cada uno de los cuatro casos de aptitud definidos anteriormente). En consecuencia, podemos observar que en todas las generaciones hubo individuos que no lograban centrar el carro para ninguno de los cuatro casos de aptitud.

Como podemos observar, el mejor individuo desde la generación 0 (la población inicial) hasta la generación 3 inclusive, fue el mismo (su aptitud era 304.834 – recordemos que este número es la suma de los movimientos realizados en cada caso de aptitud hasta dejar el carro en su estado objetivo). En la generación 4, el mejor individuo pasó a ser otro, el cual se mantuvo como el mejor de su población hasta la generación 5 inclusive (su aptitud era 89.400). Luego, en la generación 6, el mejor individuo pasó a ser otro. Este individuo se mantuvo como el mejor de su población hasta la generación 21 inclusive. Dado que dicho programa (individuo) se repitió como el mejor durante 15 generaciones sucesivas (desde la 7 hasta la 21 inclusive), el motor genético se detuvo y retornó dicha estrategia de control como solución.

Los datos de la tabla anterior son representados en el siguiente gráfico para una mejor visualización de los mismos:

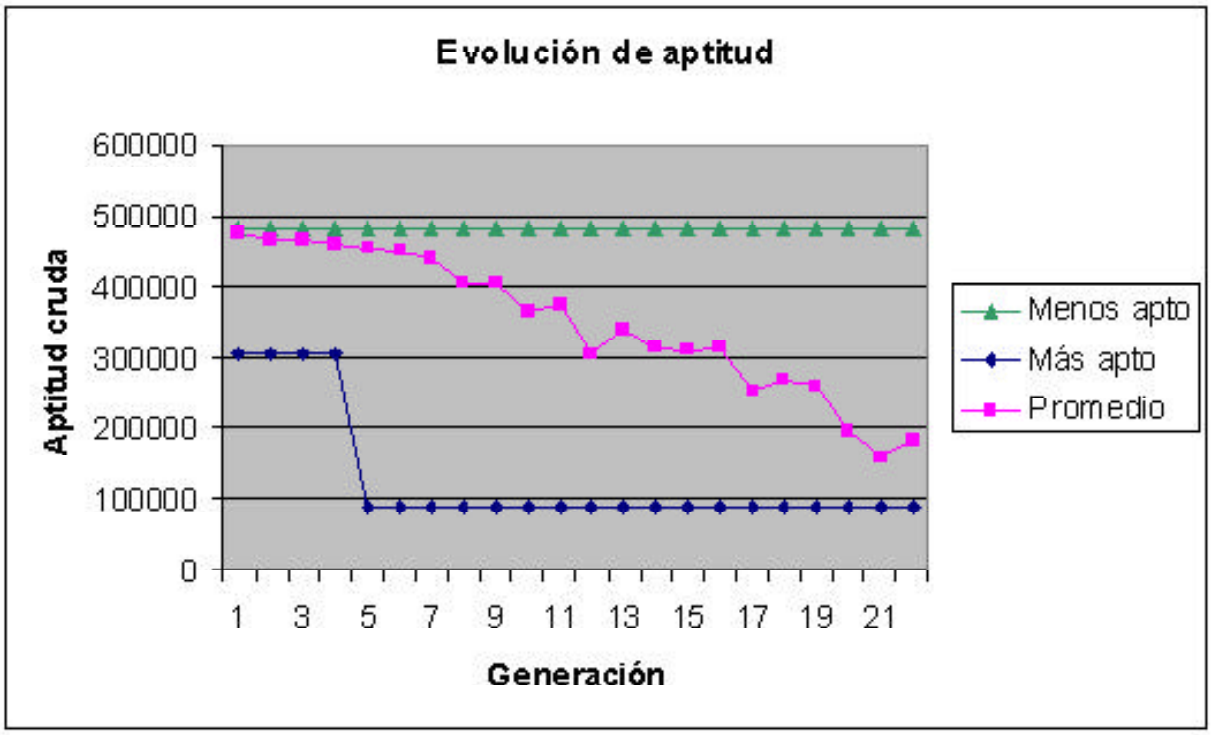

Figura 23. Gráfico de evolución de aptitud a lo largo de las generaciones.

Como el lector podrá apreciar, este gráfico es de suma utilidad para comprender en forma sencilla cómo evolucionó la aptitud de los individuos a lo largo de las distintas generaciones. Es interesante en particular observar los cambios que sufrió la aptitud promedio de la población generación tras generación. Puede notarse que la aptitud promedio no mejoró en todas las generaciones. Sin embargo, la misma sigue una tendencia que la aproxima a la aptitud del mejor individuo.

6.1.5 Comparación con la solución óptima

Resulta interesante hacer una comparación entre la eficiencia alcanzada por la estrategia de control automáticamente generada en esta ejecución del motor genético y por la estrategia de control que representa la solución óptima al problema de centrado del carro. Una estrategia será más eficiente que la otra en la medida que ejecute menos movimientos sobre el carro para llevarlo al estado objetivo.

El lector recordará que en la sección 5.1 se informó que existe una solución óptima conocida para el problema del centrado del carro. La misma es representada por la Expresión-S (> (* -1 X) (*V (ABS V))).

A continuación se muestra, para cada uno de los cuatro casos de aptitud anteriormente definidos, la cantidad de movimientos que realizan sobre el carro la estrategia de control óptima y la estrategia de control otorgada automáticamente por el motor genético (a la cual llamaremos «estrategia subóptima» de ahora en más), respectivamente, para centrarlo (x=0) y dejarlo en reposo (v=0).

	Posición inicial	*Velocidad inicial*	*Estrategia óptima - cantidad de movimientos*	*Estrategia subóptima retomada por el motor genético - cantidad de movimientos*
Caso de aptitud nº 1	2.000 m	100 m/s	No logra situar el carro en x=0 con v=0	48.400
Caso de aptitud nº 2	-1.903 m	-102 m/s	No logra situar el carro en x=0 con v=0	16.688
Caso de aptitud nº 3	1.850 m	-87 m/s	No logra situar el carro en x=0 con v=0	7.864
Caso de aptitud nº 4	-1.744 m	76 m/s	No logra situar el carro en x=0 con v=0	16.432

Tabla 4. Comparación entre la estrategia óptima y la estrategia subóptima para ubicar el carro en x=0 con v=0.

El lector puede verse sorprendido por los datos contenidos en esta tabla. A decir verdad, el autor de este trabajo también se sorprendió al obtener estos resultados en la experimentación. La pregunta que al lector seguramente le surgirá es la siguiente: ¿cómo es posible que la estrategia óptima para centrar el carro no logre ubicarlo en x=0 con v=0 en estos casos de aptitud, mientras que la estrategia subóptima retornada por el motor genético sí lo logra? ¿Es entonces la denominada «estrategia de control óptima» realmente óptima? La respuesta a estos interrogantes es sencilla y será explicada a continuación.

La estrategia de control conocida como óptima para este problema logra ubicar el carro, para *cualesquiera* valores iniciales de posición y de velocidad, en una posición *cercana* al origen (eventualmente en x=0), con una velocidad *cercana* a cero (eventualmente con v=0), haciendo uso de la *menor* cantidad de movimientos posibles. Por lo tanto, la estrategia denominada «óptima» es óptima en el sentido de que utiliza la menor cantidad de movimientos posibles para centrar el carro cerca del origen con una velocidad cercana a cero, pero no necesariamente ubica el carro en x=0 con v=0 (aunque en ciertas ocasiones sí puede lograrlo).

La estrategia subóptima retornada por el motor genético, en cambio, centra el carro en la posición x=0 con velocidad v=0 para las condiciones iniciales de los cuatro casos de aptitud anteriormente definidos, moviendo el carro una cantidad de veces mucho mayor en comparación a la cantidad de veces que el carro es movido por la estrategia óptima para ubicarlo cercano al origen con velocidad cercana a 0.

En la siguiente tabla veremos, para los mismos casos de aptitud anteriormente vistos, cuántos movimientos realiza la estrategia óptima para ubicar el carro cercano al origen con velocidad cercana a nula. Se especificará en dicha tabla los valores de x y v más pequeños que es capaz de alcanzar dicha estrategia en cada uno de estos casos de aptitud:

	Posición inicial	*Velocidad inicial*	*Posición final*	*Velocidad final*	*Cantidad de movimientos realizados*
Caso de aptitud nº 1	2.000 m	100 m/s	0,5 m	1,0 m/s	558
Caso de aptitud nº 2	-1.903 m	-102 m/s	1 m	1,5 m/s	565
Caso de aptitud nº 3	1.850 m	-87 m/s	0,5 m	0 m/s	438
Caso de aptitud nº 4	-1.744 m	76 m/s	-1 m	-1,5 m/s	369

Tabla 5. Cantidad de movimientos realizados por la estrategia óptima para ubicar el carro cercano al origen con velocidad cercana a cero.

Como podemos observar a través de la experimentación, para las cuatro duplas de valores iniciales de *x* y *v*, la cantidad de movimientos realizados para centrar el carro por la estrategia óptima es notablemente menor a la cantidad de movimientos realizados por la estrategia subóptima:

	Posición inicial	*Velocidad inicial*	*Estrategia óptima - cantidad de movimientos*	*Estrategia subóptima - cantidad de movimientos*
Caso de aptitud nº 1	2.000 m	100 m/s	558	48.400
Caso de aptitud nº 2	-1.903 m	-102 m/s	565	16.688
Caso de aptitud nº 3	1.850 m	-87 m/s	438	7.864
Caso de aptitud nº 4	-1.744 m	76 m/s	369	16.432

Tabla 6. Comparación entre la estrategia óptima y la estrategia subóptima.

Obviamente, en esta diferencia debemos tener en cuenta que la estrategia subóptima logra ubicar el carro en *x*=0 con *v*=0 para cada caso de aptitud, mientras que la estrategia óptima no lo hace para ninguna. De todos, la diferencia relativa de movimientos realizados por una y otra estrategia es significativa.

6.1.6 Utilidad de la estrategia de control retornada por el motor genético

Para el caso de la estrategia óptima, puede asegurarse que la misma logra situar el carro cercano al origen y con velocidad cercana a cero para *cualesquiera* valores iniciales de posición y velocidad. Sin embargo, para el caso de la estrategia subóptima retornada por el motor genético, esto no se cumple. Es decir, existen duplas de valores iniciales de posición y de velocidad para los cuales la estrategia subóptima no logra ubicar el carro en *x*=0 con *v*=0 y para los cuales tampoco logra ubicarlo cercano al origen con velocidad cercana a 0. Para aseverar esto mediante experimentación, se define el número 5.000.000 como la máxima cantidad de movimientos permitidos; si la estrategia subóptima supera ese número de movimientos para centrar el carro, se considera que no logra su objetivo. Algunas de estas duplas de valores iniciales para las cuales la estrategia subóptima no logra centrar el carro ni ubicarlo cercano al origen con velocidad cercana a cero son, por ejemplo:

(Posición inicial ; velocidad inicial)
(-8.300 m; 3.421 m/s)
(9.312 m; 3.316 m/s)
(7.900 m; -4.415 m/s)

Tabla 7. Ejemplos de duplas iniciales no soportadas por la estrategia subóptima.

La pregunta que cabe aquí preguntarnos es entonces la siguiente: ¿qué utilidad tiene la estrategia subóptima para centrar el carro en *x*=0 con *v*=0, si sabemos que no logra ese objetivo para ciertas duplas de valores iniciales (*x*;*v*)?

Debido a la infinidad de duplas iniciales (*x*;*v*) que pueden ser formadas, es imposible determinar la totalidad de duplas iniciales en las que la estrategia subóptima es capaz de centrar el carro. Sin embargo, podemos acotar el dominio de valores iniciales permitidos para *x* y *v* a cierto intervalo, y realizar simulaciones de centrado del carro partiendo de todas las duplas (*x*;*v*) que pueden formarse con valores enteros de ese intervalo.

Por ejemplo, tomemos el intervalo de números enteros [-250; 250]. La cantidad de números enteros en este intervalo es 501. Si para formar las duplas de valores iniciales de posición y velocidad tomáramos sólo números pertenecientes a ese intervalo, podríamos formar 501 * 501 = 251.001 duplas distintas. Podríamos realizar una simulación de cómo la estrategia subóptima intenta centrar el carro en *x*=0 y *v*=0 para cada una de las 251.001 duplas de valores iniciales. Si el resultado fuese exitoso para todas las duplas, sabríamos

que el carro puede ser centrado partiendo de cualquier dupla formada por números enteros en el intervalo [-250; 250]. Luego podríamos extender dicho intervalo mediante un procedimiento de prueba y error.

Se realizaron las 251.001 simulaciones para determinar si la estrategia subóptima logra centrar el carro en x=0 con v=0 para todas las duplas iniciales posibles que pueden formarse con números enteros del intervalo anteriormente nombrado. El resultado fue exitoso: la estrategia subóptima logró centrar el carro para todas las 251.001 duplas (x;v).

Sería de sumo interés realizar simulaciones para extender este intervalo y así determinar con seguridad un intervalo aún más grande de duplas iniciales (x;v) para las cuales la estrategia subóptima logra centrar el carro. Esto no será realizado en este trabajo debido a que dichas simulaciones consumen una gran cantidad de tiempo. De hecho, para realizar las 251.001 simulaciones anteriormente nombradas, se utilizaron dos computadoras: una de ellas con procesador Pentium III de 450 Mhz; la otra, con procesador AMD Athlon XP 2600+ de 2,08 Ghz. Las simulaciones fueron divididas en dos partes y cada parte fue asignada a cada computadora. Con las dos computadoras operando en paralelo, el proceso demoró un total de 16 horas aproximadamente. El lector podrá imaginar entonces que, haciendo uso de los recursos computacionales poseídos por el autor de este trabajo, una simulación para todas las duplas formables, por ejemplo, en el intervalo [-2000; 2000], requeriría días de procesamiento ininterrumpido.

Se presume que la estrategia de control subóptima logra centrar el carro incluso para duplas formadas por números enteros en el intervalo [-1000; 1000]. Esta presunción está basada simplemente en que para aproximadamente 53.100 duplas formadas por valores elegidos al azar en dicho intervalo, la estrategia subóptima logró centrar el carro siempre. Por ejemplo:

Posición inicial	*Velocidad inicial*	*Cantidad de movimientos realizados por la estrategia subóptima para ubicar carro en x=0 con v=0*
1.000	1.000	4.007.980
1.000	-1.000	1.334.661
-1.000	1.000	3.999.980
-1.000	-1.000	1.337.328

Tabla 8. Ejemplos de duplas iniciales para las cuales la estrategia subóptima centra el carro (x=0) con velocidad nula (v=0).

De todos modos, en este caso no se realizó una simulación estricta para todas las duplas formables como la realizada para el intervalo [-250; 250], por lo que no puede asegurarse que el carro sea centrado para *todas* las duplas formadas por números enteros en el intervalo [-1.000; 1.000].

6.1.7 Simulación del individuo generado

Luego de esta ejecución del motor genético, y tal como la pantalla de finalización de ejecución del mismo avisó, se crearon cuatro archivos MASAPTO1.SIM, MASAPTO2.SIM, MASAPTO3.SIM y MASAPTO4.SIM. Los mismos pueden ser abiertos por el programa de simulación creado para este trabajo, el cual puede ser ejecutado desde la misma interfaz del motor genético.

Cada archivo, al ser abierto por dicho programa, permite observar una simulación gráfica de cómo la estrategia de control resultante centra el carro partiendo, respectivamente, de las condiciones iniciales de posición y de velocidad especificados para cada caso de aptitud.

Obviamente, mostrar dicha simulación aquí es imposible. De todos modos, mostramos aquí una imagen de la ejecución de dicho simulador para que el lector pueda apreciarlo.

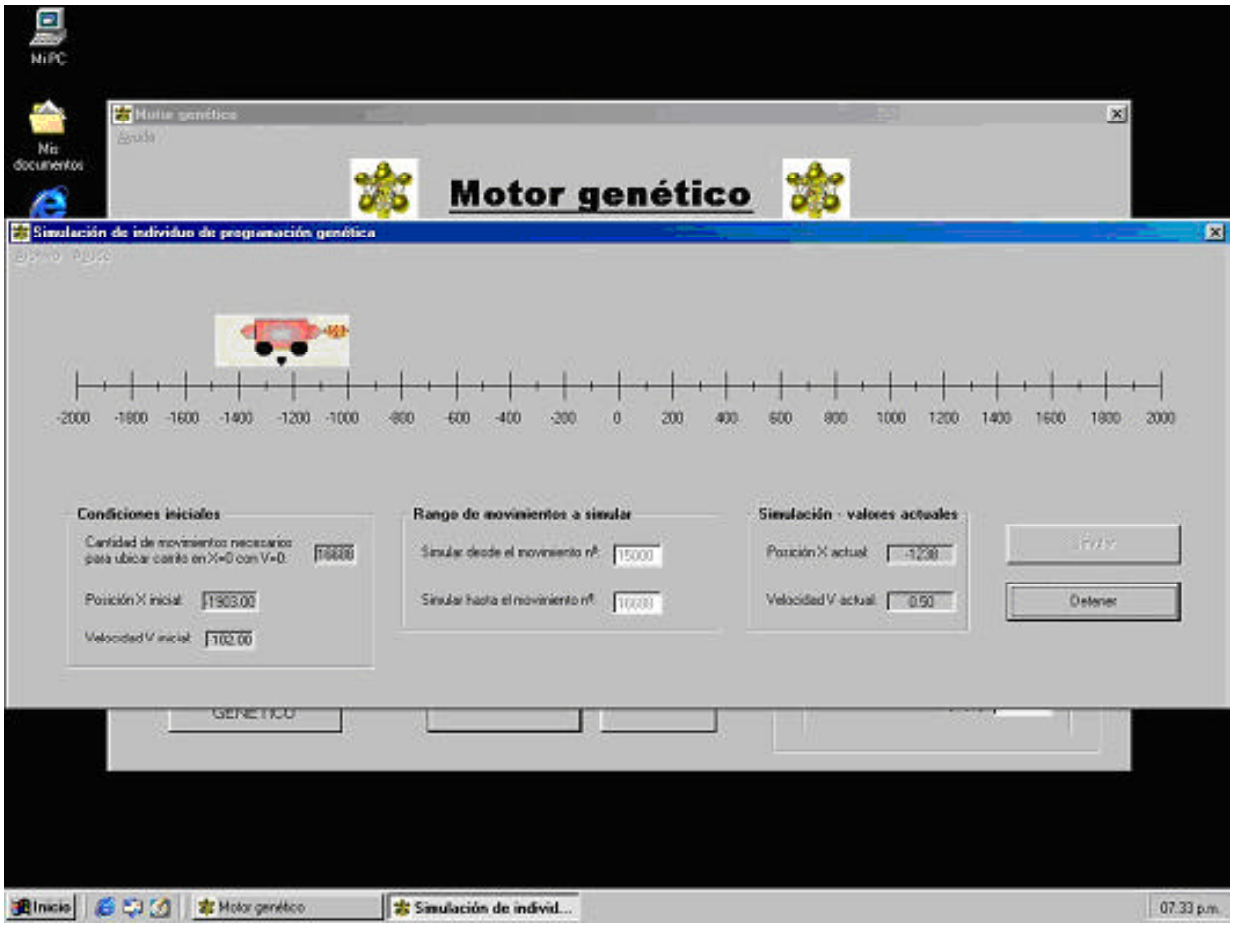

Figura 24. Simulador de individuo retornado por el motor genético.

6.2 Resultado experimental número dos

La prueba anterior nos permitió tener un vistazo rápido de cómo evoluciona la aptitud de los individuos en las distintas generaciones bajo la acción de este motor genético. Sin embargo, dicha prueba fue extremadamente sencilla, dado que se ingresó un número muy pequeño de individuos (50) y de generaciones (50) a ser procesadas, como así también un número pequeño de máxima cantidad de generaciones sucesivas en las que el mejor individuo podía repetirse (15).

A través del gráfico de evolución de aptitud, se apreció que el mejor individuo de la población cambió sólo en dos oportunidades. Además, al comparar la aptitud de la estrategia subóptima generada y la aptitud de la estrategia óptima conocida para este problema, pudimos apreciar que la primera era significativamente peor que la segunda, si bien la primera lograba siempre centrar el carro en x=0 con v=0.

Cabe aquí preguntarnos entonces si hubiese sido posible encontrar una estrategia subóptima más apta (es decir, de aptitud más cercana a la de la óptima) si hubiésemos permitido la evolución de los individuos a lo largo de un número mayor de generaciones, permitiendo también que el mejor individuo en una población dada se repitiese sucesivamente durante un número mayor de generaciones. ¿Sería así posible encontrar una estrategia subóptima mejor a la generada en la prueba anterior?

Para dar respuesta a este interrogante, se realizó la prueba que será descrita a continuación.

6.2.1 Conjunto de funciones utilizado

En esta prueba se hizo más dificultosa y más realista la generación de una estrategia subóptima, dado que se incluyeron en el conjunto de funciones aún más funciones que no forman parte de la solución óptima del problema.

Se hizo uso del conjunto de funciones F = {+, -, *, /, <, >, <=, >=, abs, and, or, ==, !=, sin, cos}. Es decir, a las tres funciones que forman parte de la solución óptima (*, >, abs), se le agregaron 12 funciones que no tienen utilidad en este problema.

6.2.2 Parámetros ingresados en el motor genético

El objetivo de esta prueba fue observar cómo evolucionaría la aptitud de los individuos si se estableciera un número mayor de individuos y un número mayor de generaciones a ser procesadas, como así también un

número mayor de generaciones sucesivas en las que se permitiese la repetición del mejor individuo. En consecuencia, se cargó la interfaz del motor genético con los siguientes parámetros de entrada:

1. Cantidad de individuos en cada generación: 120.
2. Cantidad de generaciones: 500.
3. Cantidad de casos de aptitud a simular para cada individuo: 4.
4. Cantidad máxima de movimientos que se permite demorar a cada estrategia de control para que la misma centre y deje en reposo al carro para cada caso de aptitud: 160.000.
5. Cantidad máxima de generaciones en las cuales puede repetirse el mejor individuo: 215.

Cada caso de aptitud se compone de las mismas condiciones iniciales definidas en la prueba anterior:

	Posición inicial del carro	*Velocidad inicial del carro*
Caso de aptitud nº 1	2.000 m	100 m/s
Caso de aptitud nº 2	-1.903 m	-102 m/s
Caso de aptitud nº 3	1.850 m	-87 m/s
Caso de aptitud nº 4	-1.744 m	76 m/s

Tabla 9. Casos de aptitud

6.2.3 Resultado final emitido por el motor

Luego de aproximadamente 27 horas de procesamiento (en una computadora con procesador AMD Athlon XP 2600+ de 2,08 Ghz), el motor genético finalizó su ejecución y emitió el resultado que se ve en la siguiente pantalla, terminando así su ejecución:

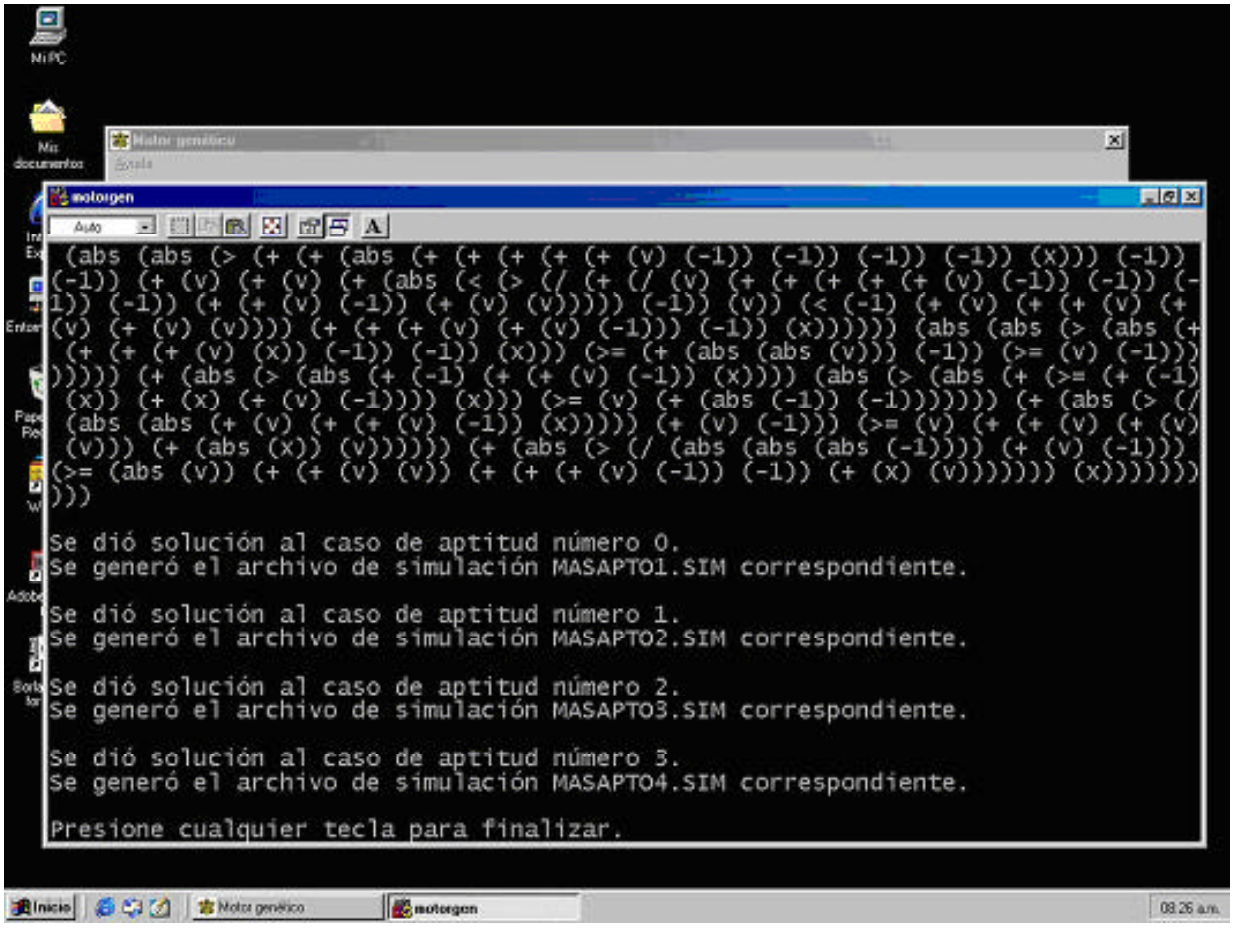

Figura 25. Resultado final emitido por el motor.

El motor se detuvo por el alcance del número máximo de generaciones. Es decir, no hubo un individuo que se repitiera como el mejor de su población durante 215 generaciones.

La estrategia subóptima retornada como resultado en esta ejecución del motor genético es la que puede observarse en la anterior imagen. Como se puede apreciar, su extensión es notablemente superior a la de la obtenida en la prueba anterior. Su aptitud es 5.916. Logra llevar el carro a su estado objetivo en los cuatro casos de aptitud definidos. Recordemos que el hecho de que su aptitud sea 5.916 significa que, para llevar el carro a su estado objetivo en los cuatro casos de aptitud, la estrategia realiza, para la sumatoria de los cuatro casos, 5.916 movimientos sobre el carro.

Puede observarse que haber aumentado el número de individuos por población, el número de generaciones a procesar y el número de generaciones en las que puede repetirse sucesivamente el mejor individuo, incrementó drásticamente la aptitud de la estrategia de control resultante. Mientras que en la prueba anterior se obtuvo un individuo de aptitud 89.384, en esta prueba se obtuvo un individuo mucho más apto de aptitud 5.916. Es decir, se obtuvo una estrategia de control 15 veces mejor.

6.2.4 Evolución de aptitud a lo largo de las generaciones

A continuación se mostrará, en forma gráfica, cómo evolucionó la aptitud de los individuos en cada generación:

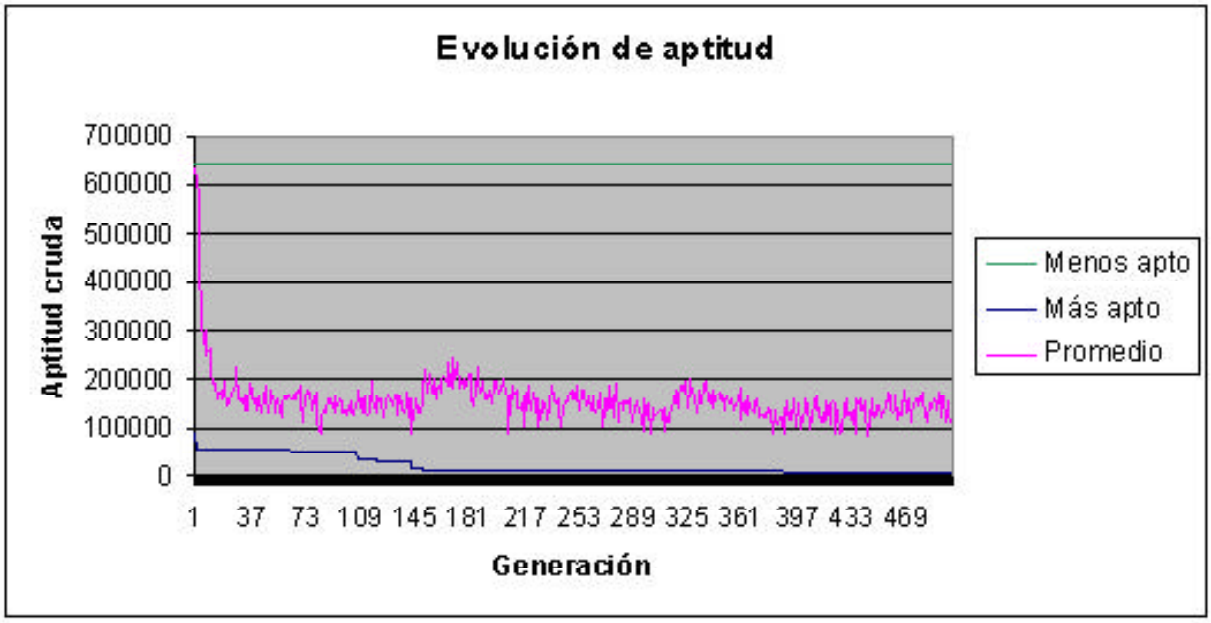

Figura 26. Gráfico de evolución de aptitud a lo largo de las generaciones.

Puede observarse que el mejor individuo de la población cambió en numerosas ocasiones, aproximándose cada vez más a cero. Esto nos demuestra la utilidad que tiene el hecho de aumentar el número máximo de generaciones a ser procesadas.

Como también puede observarse, el peor individuo se mantuvo en todas las generaciones. Su aptitud es de 640.004, número que nace de sumar 4 veces el número 160.001. Es decir, el número 640.004 representa la sumatoria de movimientos realizados por la estrategia de control para centrar el carro –sin éxito- en los cuatro casos de aptitud definidos por el usuario.

6.2.5 Comparación con la solución óptima

A continuación se mostrará una tabla en donde figuran la cantidad de movimientos realizados sobre el carro para llevarlo a su estado objetivo por la estrategia óptima y por la estrategia subóptima obtenida en esta prueba, respectivamente, para cada uno de los cuatro casos de aptitud definidos.

	Posición inicial	*Velocidad inicial*	*Estrategia óptima - cantidad de movimientos*	*Estrategia subóptima - cantidad de movimientos*
Caso de aptitud n° 1	2.000 m	100 m/s	558	1.516
Caso de aptitud n° 2	-1.903 m	-102 m/s	565	2.080
Caso de aptitud n° 3	1.850 m	-87 m/s	438	1.232
Caso de aptitud n° 4	-1.744 m	76 m/s	369	1.088

Tabla 10. Comparación entre la estrategia óptima y la estrategia subóptima.

Mediante esta tabla puede apreciarse cuánto más cercana a la aptitud de la estrategia óptima es la aptitud de la estrategia subóptima generada en esta prueba, en relación a la estrategia subóptima de la prueba anterior. Puede notarse entonces la utilidad de haber aumentado el número de individuos por generación, el número de generaciones a procesar y el número máximo de repeticiones permitidas del mejor individuo en generaciones sucesivas.

6.3 Resultado experimental número tres

Se realizó una tercera prueba exactamente idéntica a la anteriormente explicada en la sección 6.2. Es decir, se utilizó el mismo conjunto de funciones, los mismos parámetros de entrada y los mismos casos de aptitud.

En esta prueba que demoró aproximadamente 22 horas, el motor genético volvió a detenerse por el alcance del número máximo de generaciones. La estrategia subóptima generada dio solución a los cuatro casos de aptitud definidos. Su aptitud (16488) fue levemente peor que la de la estrategia obtenida en la prueba anterior (5916).

Para esta prueba sólo se desea mostrar al lector el gráfico de evolución de aptitudes, para que el mismo pueda apreciar que, más allá del diferente resultado obtenido, la evolución de los individuos en el motor genético se produce de un modo muy semejante en distintas ejecuciones del mismo para pruebas idénticas:

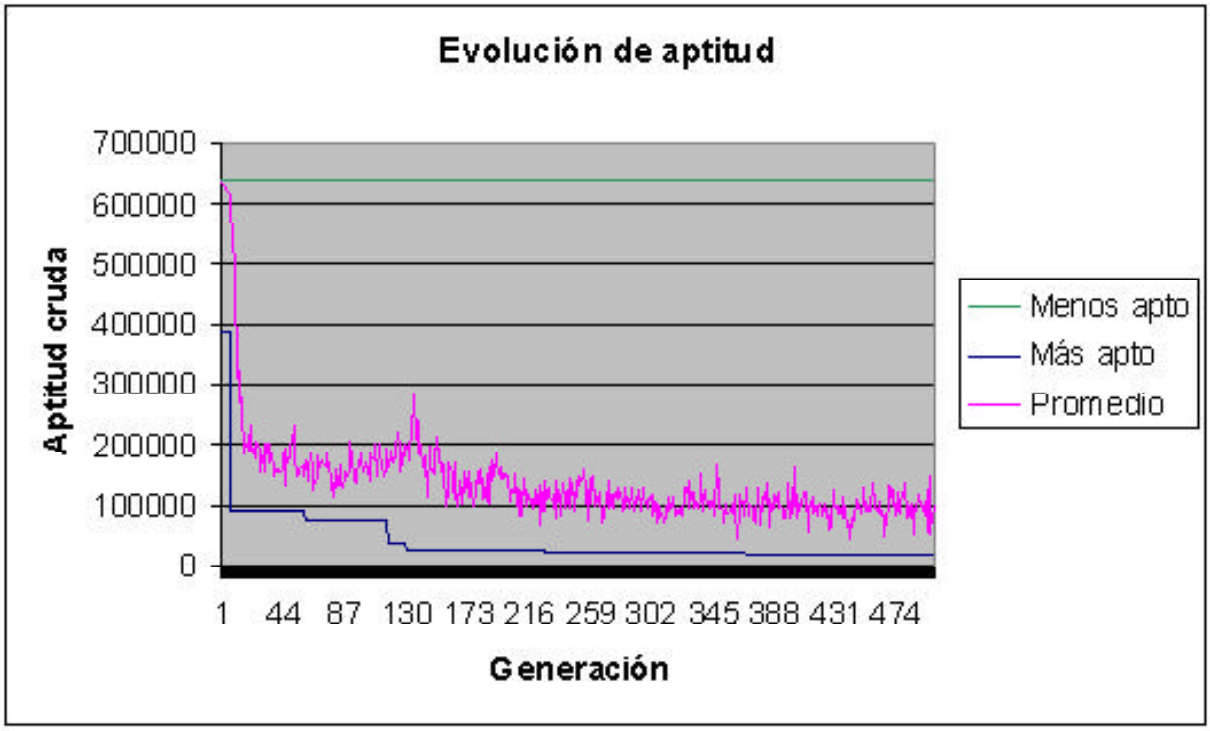

Figura 27. Gráfico de evolución de aptitud a lo largo de las generaciones.

Como podrá notar el lector, este gráfico es semejante al obtenido en la prueba anterior.

6.4 Resultado experimental número cuatro

6.4.1 Conjunto de funciones utilizado

Se analizó el comportamiento del motor cuando el conjunto de funciones está formado sólo por las funciones que forman parte de la solución óptima conocida. Por lo tanto, tenemos que F = {*, >, abs}.

6.4.2 Parámetros ingresados en el motor genético

Se cargó la interfaz del motor genético con los siguientes parámetros de entrada:

1. Cantidad de individuos en cada generación: 120.
2. Cantidad de generaciones: 500.
3. Cantidad de casos de aptitud a simular para cada individuo: 4.
4. Cantidad máxima de movimientos que se permite demorar a cada estrategia de control para que la misma centre y deje en reposo al carro para cada caso de aptitud: 130.000.
5. Cantidad máxima de generaciones en las cuales puede repetirse el mejor individuo: 175.

Los casos de aptitud son los mismos de las tres pruebas anteriores:

	Posición inicial del carro	*Velocidad inicial del carro*
Caso de aptitud n° 1	2.000 m	100 m/s
Caso de aptitud n° 2	-1.903 m	-102 m/s
Caso de aptitud n° 3	1.850 m	-87 m/s
Caso de aptitud n° 4	-1.744 m	76 m/s

Tabla 11. Casos de aptitud.

6.4.3 Resultado final emitido por el motor

Luego de aproximadamente 18 horas de procesamiento, el motor finalizó su ejecución y mostró la siguiente pantalla:

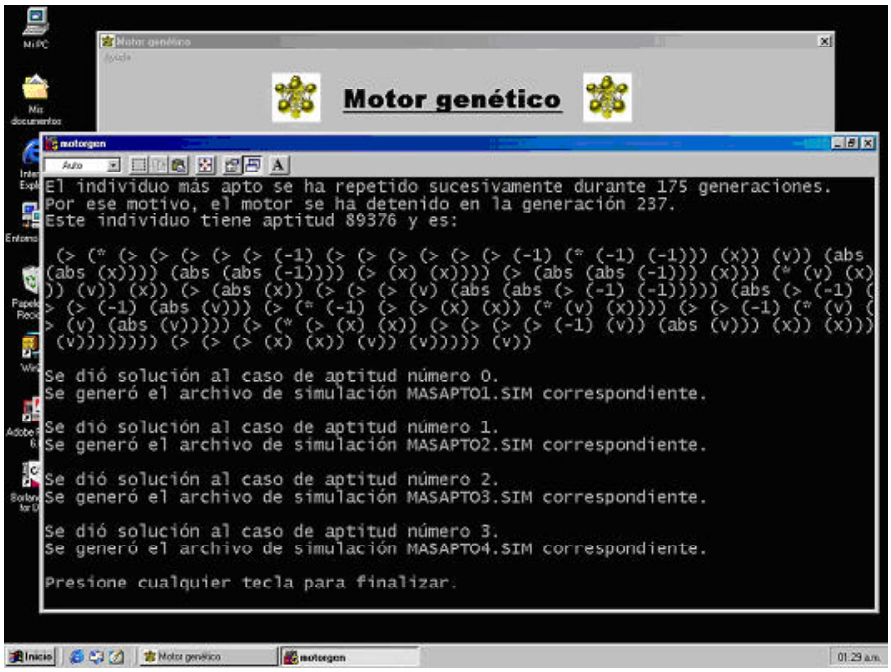

Figura 28. Resultado final emitido por el motor.

Como puede observarse, el motor se detuvo en la generación 237 porque el mejor individuo se repitió durante 175 generaciones sucesivas. Dicha estrategia de control fue retornada entonces como resultado. Su aptitud es 89376 y da solución a los cuatro casos de aptitud ingresados.

6.4.4 Evolución de aptitud a lo largo de las generaciones

A continuación se muestra el gráfico correspondiente:

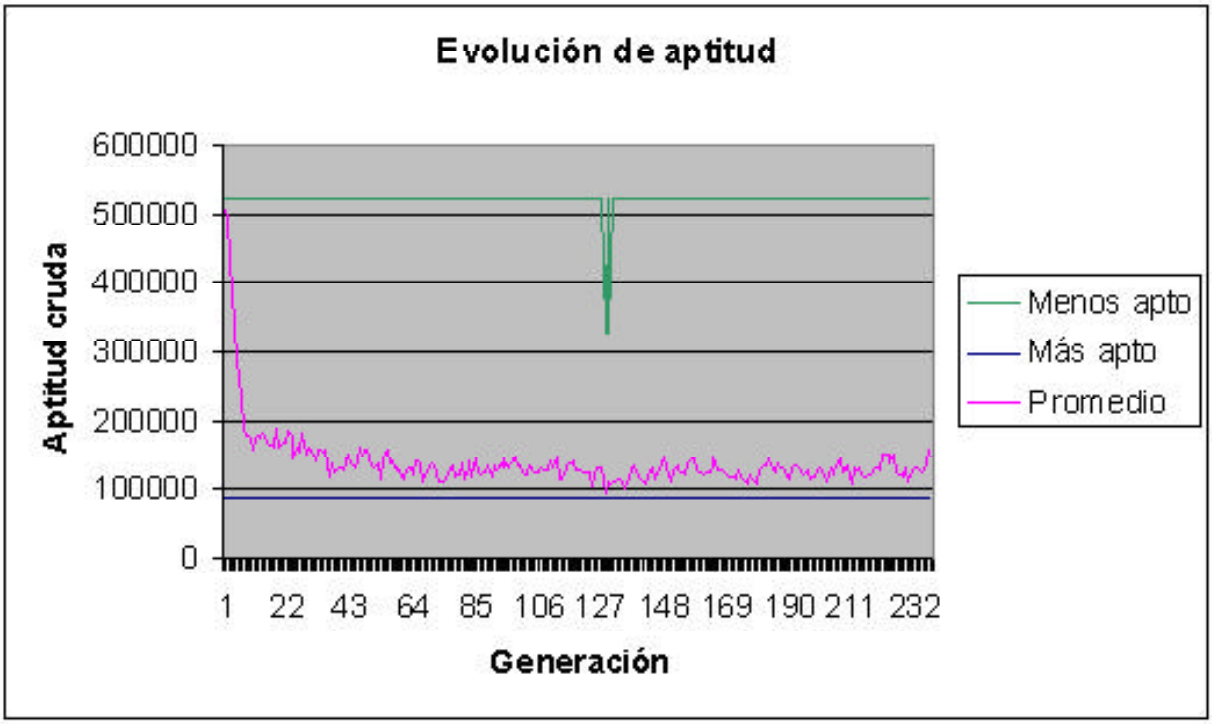

Figura 29. Gráfico de evolución de aptitud a lo largo de las generaciones.

El mejor individuo en la población sólo cambió una vez. En la generación inicial, el mejor individuo tuvo aptitud 89.384. Dicho individuo se mantuvo como el mejor de su población hasta la generación 61 inclusive. Luego, en la generación 62, nació un individuo levísimamente superior de aptitud 89.376. Dicho individuo se mantuvo como el mejor de su población durante 175 generaciones sucesivas. Por ese motivo, el motor se detuvo en la generación 237.

Durante todas las generaciones, el peor individuo tuvo aptitud 520.004, a excepción de las generaciones 127 y 129, en las cuales el peor individuo tuvo aptitud 324.834.

6.4.5 Comparación con la solución óptima

A continuación se mostrará una tabla en donde figuran la cantidad de movimientos realizados sobre el carro para llevarlo a su estado objetivo por la estrategia óptima y por la estrategia subóptima obtenida en esta prueba, respectivamente, para cada uno de los cuatro casos de aptitud definidos.

	Posición inicial	*Velocidad inicial*	*Estrategia óptima - cantidad de movimientos*	*Estrategia subóptima - cantidad de movimientos*
Caso de aptitud nº 1	2.000 m	100 m/s	558	48.400
Caso de aptitud nº 2	-1.903 m	-102 m/s	565	16.684
Caso de aptitud nº 3	1.850 m	-87 m/s	438	7860
Caso de aptitud nº 4	-1.744 m	76 m/s	369	16.432

Tabla 12. Comparación entre la estrategia óptima y la estrategia subóptima.

Recordemos que en estas diferencias se debe tener en cuenta que la estrategia subóptima lleva el carro exactamente a x=0 y v=0, mientras que la estrategia óptima lo lleva a valores de x y v cercanos a 0. De todos modos, la estrategia subóptima aquí obtenida es notablemente peor que la óptima.

Podemos notar que la estrategia subóptima (de aptitud 89.376) obtenida en esta prueba es también notablemente peor respecto a la estrategia subóptima generada en la prueba experimental número dos, cuya aptitud había sido 5.916. ¿Cuáles pueden ser los motivos?:

- En primer lugar, en esta prueba se definió como 175 la cantidad máxima de generaciones sucesivas en las que el mejor individuo podía repetirse, mientras que en la prueba experimental número 2, ese número se definió como 215. En consecuencia, en este caso se incrementó la probabilidad de que el motor se detuviera antes de alcanzar la máxima cantidad de generaciones. De hecho, eso fue lo que ocurrió. En esta prueba experimental, el motor procesó sólo 238 generaciones (la inicial más 237 adicionales); en la prueba número dos, el motor había procesado 500 generaciones, permitiendo así una mayor evolución de individuos en aquella prueba.
- En segundo lugar, en la generación 237 de la prueba experimental número dos, el individuo más apto tenía aptitud 12.632, mientras que en esta prueba, el individuo retornado en dicha generación tuvo aptitud 89.376. Por lo tanto, se puede observar que más allá de lo expresado en el anterior párrafo, la evolución de aptitud en esta prueba fue inferior a la de la prueba número dos. Esto podría estar motivado por el hecho de haber usado un conjunto de funciones reducido en esta prueba. Si bien las funciones aquí utilizadas fueron las que forman parte de la solución óptima, el hecho de que la programación genética trabaje sobre soluciones probablemente incorrectas al problema dado, puede hacer que una mayor diversidad de funciones incremente la probabilidad de obtención de estrategias subóptimas más aptas.

6.5 Comparación de resultados obtenidos

En la siguiente tabla se repiten los resultados obtenidos en las anteriores pruebas. Se muestra la cantidad de movimientos que cada estrategia subóptima obtenida hasta aquí realiza sobre el carro para centrarlo en x=0 con v=0 en cada uno de los cuatro casos de aptitud que fueron definidos. Excluimos en esta tabla al resultado experimental número uno, debido a que en éste, a diferencia del resto, se utilizó un menor número de individuos por población y una menor cantidad de generaciones.

	Caso de aptitud nº 1	*Caso de aptitud nº 2*	*Caso de aptitud nº 3*	*Caso de aptitud nº 4*
Estrategia óptima	**558**	**565**	**438**	**369**
Resultado nº 2	**1.516**	**2.080**	**1.232**	**1.088**
Resultado nº 3	5.728	5.884	2.840	2.036
Resultado nº 4	48.400	16.684	7860	16.432

Tabla 13. Comparación entre la estrategia óptima y los resultados obtenidos hasta el momento.

Como puede verse, el mejor resultado obtenido fue el correspondiente a la prueba experimental número dos, en la cual se obtuvo una estrategia subóptima de aptitud 5916. Fue en esta prueba donde se usó el conjunto de funciones más extenso utilizado hasta aquí, compuesto por las 3 funciones que forman parte de la solución óptima al problema, más otras 12 que no tienen utilidad aquí.

Si bien la estrategia óptima es mejor que la subóptima del resultado número dos, ambas realmente no son totalmente comparables dado que la óptima ubica el carro en una posición en el intervalo [-1; 1] con una velocidad en el intervalo [-1.5; 1.5] para estos cuatro casos de aptitud, mientras que la estrategia subóptima lo hace siempre en la posición 0 con velocidad 0.

Sería muy interesante, por lo tanto, observar cómo se desempeña el paradigma de la programación genética si no exigimos que la estrategia subóptima que automáticamente genere detenga el carro exactamente en x=0 con v=0, sino que lo ubique con un valor de x en el intervalo [-1; 1] y con un valor de v en el intervalo [-1.5; 1.5], tal como lo hace la estrategia óptima en los cuatro casos de aptitud hasta aquí tratados. **De ese modo, la comparación entre la estrategia óptima y otra estrategia subóptima retornada como resultado del motor genético, sería totalmente legítima.**

Esta prueba fue realizada. A continuación se mostrarán sus resultados en el resultado experimental número cinco.

6.6 Resultado experimental número cinco

6.6.1 Conjunto de funciones utilizado

Se utilizó el conjunto de funciones más extenso utilizado en las anteriores pruebas (que como ya sabemos, es aquel para el cual se obtuvo la estrategia subóptima de mejor aptitud hasta el momento). Por lo tanto:

F = {+, -, *, /, <, >, <=, >=, abs, and, or, ==, !=, sin, cos}

6.6.2 Parámetros ingresados en el motor genético

1. Cantidad de individuos en cada generación: 120.
2. Cantidad de generaciones: 500.
3. Cantidad de casos de aptitud a simular para cada individuo: 4.
4. Cantidad máxima de movimientos que se permite demorar a cada estrategia de control para que la misma centre y deje en reposo al carro para cada caso de aptitud: 160.000.
5. Cantidad máxima de generaciones en las cuales puede repetirse el mejor individuo: 215.

Cada caso de aptitud se compone de las mismas condiciones iniciales definidas en las pruebas anteriores:

	Posición inicial del carro	*Velocidad inicial del carro*
Caso de aptitud n° 1	2.000 m	100 m/s
Caso de aptitud n° 2	-1.903 m	-102 m/s
Caso de aptitud n° 3	1.850 m	-87 m/s
Caso de aptitud n° 4	-1.744 m	76 m/s

Tabla 14. Casos de aptitud.

6.6.3 Resultado final emitido por el motor

Luego de 24 horas de procesamiento, el motor genético emitió el siguiente resultado:

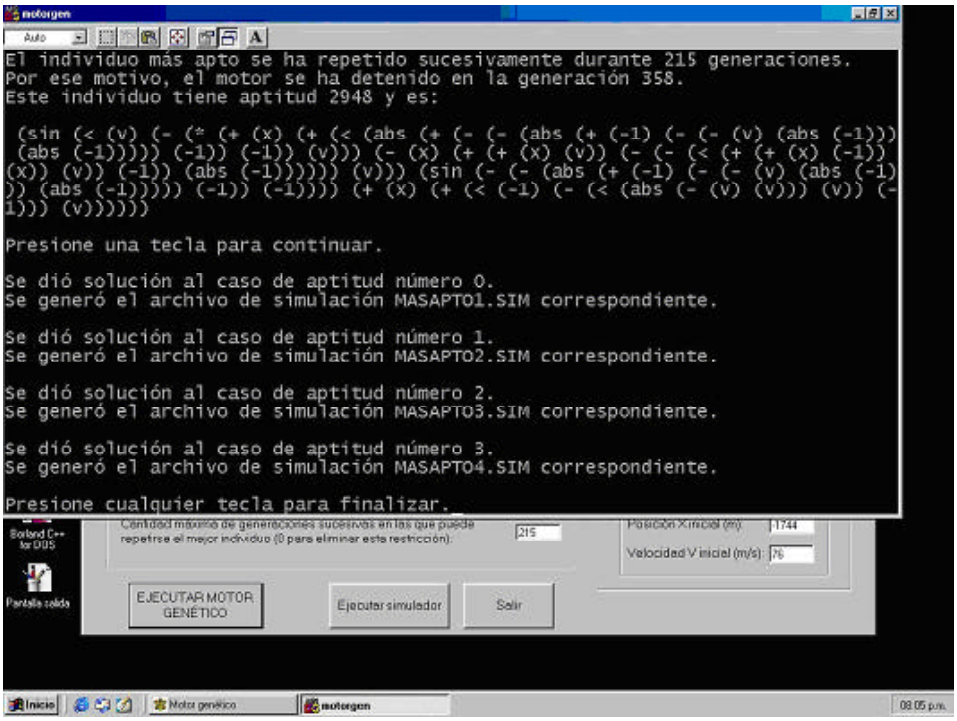

Figura 30. Resultado final emitido por el motor.

Como podemos ver, el motor se detuvo en la generación 358 debido a la repetición del mejor individuo durante 215 generaciones sucesivas. El individuo (estrategia de control) resultante dio solución a los cuatro casos de aptitud que fueron definidos. Posee una aptitud igual a 2948, **la cual es aproximadamente dos veces mejor a la aptitud de la mejor estrategia obtenida en las pruebas anteriores (cuya aptitud fue 5916).**

6.6.4 Evolución de aptitud a lo largo de las generaciones

A continuación se muestra el gráfico correspondiente:

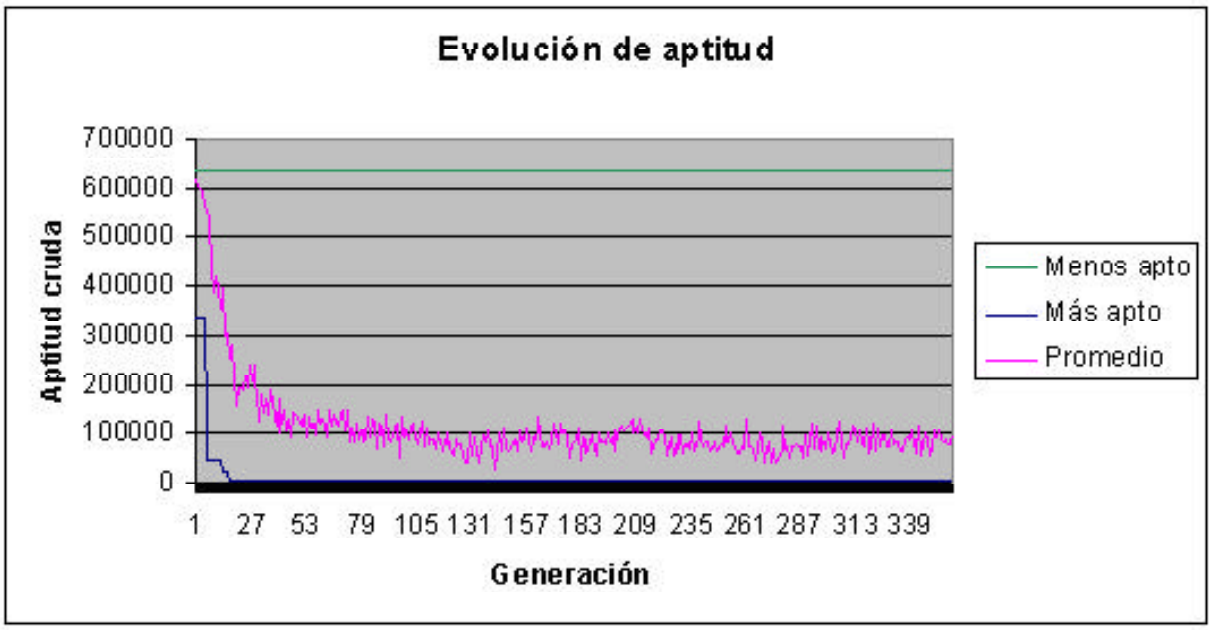

Figura 31. Gráfico de evolución de aptitud a lo largo de las generaciones.

Como puede verse, la evolución de aptitud siguió una tendencia semejante respecto a los ejemplos anteriores. En este caso, tanto la aptitud promedio como la aptitud del individuo más apto, fueron notablemente mejores respecto a las anteriores pruebas. Esto se debe, obviamente, a que la cantidad de movimientos necesarios para ubicar el carro en el intervalo [-1; 1] con velocidad en el intervalo [-1.5; 1.5] es menor que la cantidad de movimientos necesarios para ubicarlo en x=0 con v=0 en los cuatro casos de aptitud que se definieron.

6.6.5 Comparación con la solución óptima

A continuación se muestra la cantidad de movimientos realizados sobre el carro por la estrategia óptima y por la estrategia subóptima obtenida en esta prueba, respectivamente, para ubicarlo en una posición del intervalo [-1; 1] con velocidad en el intervalo [-1.5; 1.5]:

	Posición inicial	*Velocidad inicial*	*Estrategia óptima - cantidad de movimientos*	*Estrategia subóptima - cantidad de movimientos*
Caso de aptitud nº 1	2.000 m	100 m/s	558	835
Caso de aptitud nº 2	-1.903 m	-102 m/s	565	830
Caso de aptitud nº 3	1.850 m	-87 m/s	438	694
Caso de aptitud nº 4	-1.744 m	76 m/s	369	589

Tabla 15. Comparación entre la estrategia óptima y la estrategia subóptima.

Puede notarse que la aptitud conseguida en este caso es cercana a la de la estrategia óptima. Mediante este ejemplo puede apreciarse el poderío que tiene la programación genética para la generación automática de programas sin intervención de inteligencia humana.

7. Conclusiones y futuras líneas de investigación

Ingresando parámetros de entrada adecuados en el motor genético, la estrategia de control subóptima, que es automáticamente generada y retornada como resultado por el mismo, lleva el carro a su estado objetivo partiendo de cada una de las cuatro duplas iniciales de posición y de velocidad, que son definidas en cuatro casos de aptitud ingresados por el usuario al comienzo de la ejecución del motor.

Para el caso en el que se exigió al motor obtener una estrategia que llevara el carro a x=0 y v=0 en todos los casos de aptitud que se ingresaran, fue posible la obtención de una estrategia subóptima de aptitud aproximadamente 2 veces inferior a la aptitud de la estrategia óptima (para los casos de aptitud utilizados). Para el caso en el que se permitió al motor genético retornar una estrategia que ubicara el carro cercano al origen con velocidad cercana a nula (tal como lo hace la estrategia óptima), se obtuvo una estrategia subóptima de aptitud sólo 0.5 veces inferior a la de la óptima (para los casos de aptitud utilizados).

Estos resultados fueron obtenidos utilizando el conjunto de funciones más extenso utilizado en este trabajo, compuesto por las 3 funciones que forman parte de la solución óptima conocida al problema, más otras 12 que no tienen utilidad aquí.

Además de las cuatro duplas de posición y de velocidad ingresadas en los casos de aptitud, la estrategia de control, que es otorgada por el motor genético, permite llevar el carro a su estado objetivo partiendo de distintas duplas iniciales. Sin embargo, es probable que existan otras duplas iniciales de posición y de velocidad para las cuales la estrategia generada no logre este objetivo. Por lo tanto, para cualesquiera otros valores que no sean los ingresados en los casos de aptitud, se deberá realizar una simulación para cada dupla de valores de posición y de velocidad iniciales a fin de determinar para cuáles de ellas la estrategia de control resultante permite llevar el carro al estado deseado.

A fin de extender la cantidad de duplas iniciales para las cuales una determinada estrategia subóptima permita centrar el carro, se propone como futura línea de investigación analizar si esto puede ser logrado haciendo uso de más casos de aptitud durante la ejecución del motor.

A fin de mejorar la aptitud de las estrategias subóptimas generadas por el motor genético, se propone realizar ejecuciones del mismo durante una cantidad de generaciones mayor que la máxima cantidad seleccionada (500) para realizar las pruebas en este trabajo, a fin de determinar si es posible la obtención de estrategias de control de aptitud más cercana a la óptima a medida que la cantidad de generaciones crece. Se propone también agregar el uso del operador genético de mutación a fin de analizar si el mismo puede ocasionar un cambio positivo en la evolución de los individuos.

Como última línea de futura investigación, se propone investigar la potencia de la programación genética en la creación automática de programas de conocimientos.

Referencias

[1] John R. Koza. Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.

[2] Automatic Programming of Robots using Genetic Programming. John R. Koza, James P. Rice. Stanford University, EE.UU., 1992. http://www.cse.unr.edu/~monica/Courses/CS493-790/PapersPDF/Koza_AutomaticProgramming.pdf

[3] Evolutionary Design of Fuzzy Logic Controllers. Carlos Troya, Enrique Alba, José Troya. Departamento de lenguajes y ciencias de la computación. Universidad de Málaga, España, 1996. http://www.lcc.uma.es/~ccottap/papers/isic96flc.pdf

[4] Seminario: algoritmos genéticos. Jorge Martín Martín, Diego García Morate. Departamento de Informática. Universidad de Valladolid, España, 2004. http://www.infor.uva.es/~calonso/IAI/TrabajoAlumnos/memoriaAG.pdf

[5] Stochastic Modelling and Genetic Algorithm-Based Optimal Control of Air Conditioning Systems. H. N. Lam. Departamento de Ingeniería Mecánica. Universidad de Hong Kong,

China.
http://www.ibpsa.org/proceedings/bs93/papers/BS93%20435-441.PDF
[6] Sitios oficiales de John R. Koza.
http://www.genetic-programming.com
http://www.genetic-programming.org
[7] Algoritmos genéticos y computación evolutiva. Adam Marczyk. 2004.
http://the-geek.org/docs/algen/
[8] Nils J. Nilsson. Inteligencia Artificial: una nueva síntesis. Mc Graw Hill, España, 1998.
[9] Timetabling experiments using genetic algorithms. Liviu Lalescu, Costin Badica.
Departamento de Ingeniería de Software. Universidad de Craiova, Rumania.
http://software.ucv.ro/~badica_costin/cercetare/papers/tainn03_p2.pdf
[10] Procesamiento de Imágenes Digitales. Universidad de Sevilla, España, 2003.
http://www.us.es/gtocoma/pid/t0304/t0304.htm
[11] Autonomous Navigation Control of UAVs Using Genetic Programming. Sitio oficial de la Armada de los Estados Unidos de América.
http://www.nrl.navy.mil/content.php?P=04REVIEW197
[12] Cómputo evolutivo aplicado a la planificación de tareas de visión. Enrique Dunn, Gustavo Olague. Departamento de Ciencias de la Computación, División de Física Aplicada. Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California, Méjico, 2001.
http://www.minas.unalmed.edu.co/facultad/publicaciones/dyna/133/computo.pdf
[13] 36 Human-Competitive Results Produced by Genetic Programming.
http://www.genetic-programming.com/humancompetitive.html

Apéndice A - Algoritmos genéticos

1. Introducción

Los algoritmos genéticos en particular constituyen el paradigma más tradicional de la computación evolutiva y han hecho de ésta un área de investigación muy activa en la actualidad y que genera un interés creciente por parte de investigadores de muchas otras áreas. Este interés viene motivado en parte por el gran éxito que estas técnicas han tenido en la resolución de problemas reales. La aplicación de los algoritmos genéticos sigue creciendo en gran cantidad de áreas de muy diversa naturaleza.

En la naturaleza, el proceso evolucionario ocurre cuando las siguientes cuatro condiciones son satisfechas:

- Una entidad posee la habilidad de reproducirse.
- Existe una población de dichas entidades.
- Existe alguna variedad entre estas entidades.
- La variedad se asocia con alguna diferencia en la habilidad para sobrevivir en el ambiente.

En la naturaleza, la variedad se manifiesta como la variación de cromosomas en las entidades de la población. La variedad se traduce en una variación tanto en la estructura como así también en la conducta de las entidades del ambiente. Estas variaciones, a su vez, son reflejadas por las diferencias en las tasas de supervivencia y reproducción. Las entidades que son más capaces de desarrollar tareas en su ambiente (es decir, individuos más aptos) sobreviven y se reproducen en una tasa más alta; las entidades menos aptas, si sobreviven y se reproducen, lo hacen en una tasa menor. Este es el concepto detrás de la supervivencia de los más aptos y de la selección natural descrita por Charles Darwin en su obra *«On the Origin of Species by Means of Natural Selection»*, en 1859. Luego de un período de tiempo y varias generaciones, la población contiene más individuos cuyos cromosomas se traducen en estructuras y conductas que permiten a esos individuos desarrollar sus tareas mejor en su ambiente, como así también sobrevivir y reproducirse. Por lo tanto, a lo largo del tiempo, la estructura de los individuos en la población cambia como consecuencia de la selección natural. Al ver estas diferencias visibles y mensurables en la estructura que nacen de las diferencias de aptitud, podemos decir que la población ha evolucionado. En este proceso, la estructura surge de la aptitud.

John Holland, en su libro «Adaptation in Natural and Artificial Systems» (1975) mostró cómo el proceso evolucionario puede ser aplicado a sistemas artificiales mediante algoritmos genéticos.

El algoritmo genético simula los procesos evolucionarios descritos por Darwin, como así también las operaciones que naturalmente ocurren sobre los cromosomas. En la naturaleza, los cromosomas son strings

de caracteres codificados en el alfabeto de cuatro letras de la naturaleza. Las cuatro bases nucleótidas que aparecen a lo largo de una molécula de ADN son la adenina (A), la citosina (C), la guanina (G) y la timina (T). Esta secuencia de bases nucleótidas constituyen el string del cromosoma o el genoma de un individuo biológico. Por ejemplo, el genoma humano contiene alrededor de 2.870.000.000 bases nucleótidas.

Las moléculas de ADN son capaces de reproducirse por sí mismas. Además, los substrings que contienen miles de bases nucleótidas de la molécula de ADN son traducidas, utilizando el denominado código genético, en las proteínas y las enzimas que crean la estructura y que controlan la conducta en la células biológicas. Estas estructuras y conductas que se crean permiten a un individuo desarrollar tareas en su ambiente, sobrevivir y reproducirse en diferentes tasas. Los cromosomas de la generación descendente contiene strings de bases nucleótidas de sus padres de modo tal que dichos strings que lleven a una aptitud superior sean transferidos a futuras generaciones de la población en proporciones mayores. Ocasionalmente pueden ocurrir mutaciones en los cromosomas.

2 ¿Qué son los algoritmos genéticos?

Un algoritmo genético es un algoritmo matemático paralelo que transforma un conjunto (población) de objetos matemáticos individuales (típicamente strings de caracteres de longitud fija modelados en base a los strings de cromosomas), cada uno de los cuales posee un valor de aptitud (*«fitness»*) asociado, en una nueva población (es decir, la próxima generación) utilizando operaciones modeladas en base al principio Darwiniano de reproducción y supervivencia de los más aptos y en base a las operaciones genéticas que ocurren en la naturaleza.

3. Terminología genética

Es necesario comprender el vocabulario utilizado en los algoritmos genéticos para evitar confusiones:

- El conjunto de posibles soluciones constituye una *población de individuos*.
- Cada individuo está representado por un *cromosoma*.
- Cada cromosoma está conformado por una secuencia de *parámetros* o *genes*.
- Cada gen se localiza en una posición específica del cromosoma, llamada *locus*.
- Cada valor que puede tomar un gen constituye un *allelo*.
- El conjunto de genes contenido en un cromosoma constituye un *genotipo*, el cual posee la información necesaria para construir un *individuo* o *fenotipo*.
- Cada ciclo del procedimiento evolutivo constituye una *generación*.

4. Ejemplo de utilización del algoritmo genético

Antes de establecer el algoritmo básico utilizado por los algoritmos genéticos, y con el fin de afianzar la comprensión de este tema y de comprender cómo actúan los operadores genéticos de cruce y de mutación en estos algoritmos, el autor de esta trabajo considera apropiado ilustrar el uso del algoritmo genético con un ejemplo muy simple consistente en un problema de optimización: encontrar la mejor estrategia de negocio para una cadena de cuatro restaurantes de hamburguesas. Para los objetivos de este ejemplo simple, la estrategia para estos restaurantes consistirá en realizar tres decisiones binarias:

- ***Precio:*** ¿el precio de las hamburguesas debe ser de 50 centavos o de 10 pesos?
- ***Bebida:*** ¿se tiene que servir bebida cola o vino con la hamburguesa?
- ***Velocidad de servicio:*** ¿el restaurante debe proveer un servicio sin prisa con mozos vestidos con esmoquin o debe utilizar un servicio rápido compuesto por mozos vestidos con uniformes blancos?

4.1 Esquema de representación del problema

El objetivo es encontrar la combinación de estas tres variables de decisión que lleven al mayor beneficio. Como hay tres de dichas variables, cada una de las cuales puede asumir dos valores posibles, sería muy natural para este problema representar cada estrategia posible de negocio con un string de caracteres de longitud L = 3 sobre un alfabeto de tamaño K = 2. Para cada variable de decisión, un valor de 0 o 1 es asignado a cada una de las dos posibles decisiones. El espacio de búsqueda de este problema consiste entonces en $2^3 = 8$ estrategias de negocio posibles. Queda así constituido el esquema de representación para este problema. La identificación de un esquema de representación apropiado es el primer paso hacia la resolución del problema.

La siguiente tabla muestra cuatro de las ocho posibles estrategias de negocios expresadas en el esquema de representación recién descrito:

Número de restaurante	Precio	Bebida	Velocidad de servicio	Representación binaria
1	Alto	Cola	Rápido	011
2	Alto	Vino	Rápido	001
3	Bajo	Cola	Lento	110
4	Alto	Cola	Lento	010

Tabla 1. Cuatro estrategias posibles de negocios.

Las decisiones gerenciales acerca de los cuatro restaurantes son realizadas por un hombre que sin esperarlo heredó los restaurantes de un tío rico, el cual no le proveyó la información necesaria para saber qué estrategia de negocio produce el mayor beneficio.

En particular, el heredero no sabe cuál de las tres variables es la más importante. Desconoce la magnitud del máximo beneficio que puede percibir si llega a las decisiones óptimas o la magnitud de las pérdidas en las que puede incurrir si lleva a cabo las opciones incorrectas. Desconoce cuál variable es capaz de producir el mayor cambio en el beneficio. De hecho, no sabe siquiera si alguna de estas tres variables es relevante.

El nuevo administrador no sabe si puede acercarse al óptimo global siguiendo un procedimiento paso a paso de cambiar una variable por vez, escoger el mejor resultado, luego cambiar de modo similar una segunda variable para volver a escoger el mejor resultado. Esto implica que no sabe si las variables pueden ser optimizadas separadamente o si están interrelacionadas de un modo no lineal. Tal vez las variables están interrelacionadas de un modo tal que se puede alcanzar un óptimo global sólo si identifica primero una combinación particular de dos variables y luego varía la variable restante.

El heredero enfrenta el obstáculo adicional de recibir información acerca del ambiente sólo en la forma del beneficio alcanzado por cada restaurante cada semana. Es decir, los clientes obviamente no escriben cartas explicatorias detalladas identificando los factores precisos que afectaron su decisión de elegir ese restaurante. En otras palabras, la performance observada de los restaurantes durante la operación real es la única respuesta recibida por el administrador.

Además, el heredero no está seguro de que el ambiente operacional permanezca igual semana tras semana. Los gustos del público son inconstantes y las reglas del juego pueden cambiar abruptamente. Un esquema de operación que trabaja bien una semana, puede producir menos beneficio la semana siguiente. Los cambios en el ambiente no sólo pueden ser abruptos, sino que tampoco son anunciados con anticipación. De hecho, ni son anunciados, sino que simplemente ocurren.

Como si todo esto fuera poco, el administrador enfrenta la obligación de tomar una decisión inmediata sobre cómo empezar a operar los restaurantes la mañana siguiente. No puede tomarse el lujo de usar un procedimiento de decisión que converja a un resultado en un tiempo lejano en el futuro. No hay tiempo para un período separado de entrenamiento o de experimentación. Además, para que sea útil, el procedimiento de decisión deben empezar a producir en lo inmediato un flujo de decisiones intermedias que mantengan al sistema por encima del mínimo nivel requerido para la supervivencia, empezando esto la primera semana y continuando durante todas las semanas siguientes.

Este problema definido en forma tan desprolija es diferente a la mayoría de los problemas que encontraríamos en un libro, pero se asemeja mucho a problemas prácticos de decisión del mundo real. Es también muy semejante a los problemas de adaptación en la naturaleza.

Como el administrador no sabe nada acerca del ambiente que está enfrentado, él puede razonablemente probar una estrategia aleatoria para cada uno de los cuatro restaurantes durante la primera semana. El heredero puede esperar que este enfoque aleatorio alcance un beneficio aproximadamente igual al beneficio promedio del espacio de búsqueda en su totalidad. Favorecer la diversidad en las estrategias de negocio (individuos) seleccionadas hace que se maximice la probabilidad de obtener una performance cercana al promedio del espacio de búsqueda en su totalidad y, además, posee el beneficio de maximizar la cantidad de información que puede ser aprendida durante la primera semana de operaciones. En este ejemplo, se utilizarán las cuatros estrategias expresadas en la tabla anteriormente mostrada como la población aleatoria inicial.

De hecho, el administrador está procediendo del mismo modo que el algoritmo genético. La ejecución de dicho algoritmo comienza con un esfuerzo por aprender algo acerca del ambiente, para lo cual prueba un número de puntos aleatoriamente seleccionados del espacio de búsqueda. En particular, el algoritmo gené-

tico comienza, en la generación 0 (la generación aleatoria inicial), con una población formada por individuos creados en forma aleatoria. En este ejemplo, el tamaño *M* de la población es 4.

Para cada generación para la cual se corre el algoritmo genético, cada individuo en la población es probado contra el ambiente desconocido con el objetivo de obtener el valor de su aptitud en dicho ambiente. En este ejemplo, la aptitud se denomina beneficio, pero también puede recibir otros nombres dependientes del dominio, como por ejemplo beneficio, puntaje y valor de la función objetivo.

En la siguiente tabla se muestra la aptitud para cada uno de los $M = 4$ individuos de la población aleatoria inicial de este problema (para simplicidad, la aptitud de cada estrategia ha sido igualada al decimal equivalente del string binario que representa al cromosoma, de modo tal que el global óptimo sea $7):

	Generación 0	
i	String X_i	Fitness $f(x_i)$
1	011	3
2	001	1
3	110	6
4	010	2
Total		12
Peor		1
Promedio		3.00
Mejor		6

Tabla 2. Aptitud de la población inicial.

Superficialmente, el administrador ha aprendido el valor específico de aptitud (fitness) para las cuatro estrategias planteadas. En particular, ha aprendido que la estrategia 110 produce un beneficio de 6$ por semana. Esta estrategia es entonces el mejor individuo de la población de la generación 0. Por lo contrario, la estrategia 001 es el peor individuo de la generación.

La única información utilizada durante la ejecución del algoritmo genético es el conjunto de valores observados de aptitud de los individuos actualmente presentes en la población. El algoritmo genético transforma una población de individuos y sus aptitudes asociadas en una nueva población usando operaciones basadas en el principio Darwiniano de reproducción y supervivencia de los más aptos y usando operaciones genéticas que ocurren naturalmente.

4.2 Operación de reproducción

A continuación se empieza a desarrollar la operación Darwiniana de reproducción, utilizando un determinado método para seleccionar qué individuos serán reproducidos (ver apéndice B «Métodos de selección»). En este ejemplo, utilizaremos el método de selección de ruleta por bondad. Por lo tanto desarrollaremos la operación de reproducción proporcional a la aptitud copiando individuos en la población actual hacia la siguiente generación con una probabilidad proporcional a su aptitud.

La suma de las aptitudes de los cuatro individuos en la población actual es 12. La aptitud del individuo más apto (el 110) es 6. Por lo tanto, la fracción de aptitud de la población atribuida al individuo 110 es 0,5. En la selección proporcional a la aptitud, el individuo 110 recibe una probabilidad de 0,5 de ser seleccionado para cada una de las cuatro posiciones en la nueva población. Por lo tanto, podemos esperar que el string 110 ocupe dos de las cuatro posiciones disponibles. De todos modos, como el algoritmo genético es probabilístico, existe la posibilidad de que dicho individuo aparezca cero, una, dos, tres o cuatro veces en la nueva generación. El método descrito hasta aquí, como ya dijimos antes, corresponde al de ruleta por bondad, ya que cada individuo en la población ocupa un sector de la ruleta cuyo tamaño es proporcional a la aptitud del individuo. Aquí el mejor individuo ocupa 180º de la ruleta, la cual al girar llevará a la selección proporcional a la aptitud.

Similarmente, el individuo 011 tiene una probabilidad de 0,25 de ser seleccionado para cada una de las cuatro posiciones en la nueva población (esperamos entonces que aparezca en sólo una de ellas). La estrategia 010 posee una probabilidad de 1/6 de ser seleccionada para cada una de las cuatro posiciones, mientras que la estrategia 001 posee sólo una probabilidad de 1/12. Por lo tanto, esperamos que 010 aparezca sólo una vez en la nueva población y que 001 esté ausente en ella.

Si los cuatro strings son copiados en la nueva generación del mismo modo que estos valores esperados, la siguiente tabla muestra el resultado de aplicar la operación Darwiniana de reproducción proporcional a la aptitud a la generación 0 de esta población inicial. Esta población resultante suele denominarse «mating pool»:

	Generación 0			"Mating pool" creado luego de la reproducción	
i	String X_i	Fitness $f(x_i)$	$f(x_i)/? f(x_i)$	"Mating pool"	$f(x_i)$
1	011	3	.25	011	3
2	001	1	.08	110	6
3	110	6	.50	110	6
4	010	2	.17	010	2
Total		12			17
Peor		1			2
Promedio		3.00			4.25
Mejor		6			6

Tabla 3. «Mating pool» creado luego de la reproducción.

La reproducción proporcional a la aptitud incrementa la aptitud promedio de la población. La aptitud promedio pasó de ser 3.00 a 4.25. El peor individuo de la población pasó de tener aptitud 1 a tener aptitud 2. Estos incrementos son típicos de la reproducción proporcional a la aptitud, ya que individuos de bajo *fitness* suelen ser eliminados mientras que los de alto *fitness* suelen ser duplicados. Hay que notar que ambos incrementos vinieron a expensas de la diversidad genética de la población, ya que la estrategia 001 se ha extinguido. Por supuesto que la aptitud asociada al mejor individuo de la generación no puede incrementarse como resultado de la reproducción. El mejor individuo de la generación luego de la reproducción en la generación 0 es, en el mejor caso, el mejor individuo creado aleatoriamente.

4.3 Operación de cruce

La operación genética de cruce (recombinación sexual) permite que nuevos individuos sean creados. Así se prueban nuevos puntos en el espacio de búsqueda. La operación de cruce actúa sobre dos padres y produce dos descendientes. Los descendientes usualmente son distintos a sí mismos y distintos a sus padres. Cada descendiente posee algún material genético de cada uno de sus padres.

A partir de los *M* progenitores elegidos por el método de selección adoptado, se forman aleatoriamente *M/2* parejas para aplicarles el operador de cruce. Por cada pareja surgida de la selección de progenitores se genera un número aleatorio n_a del intervalo [0,1]. Cada pareja es cruzada sólo si n_a d» P_C, donde P_C es la «probabilidad de cruce» (para este ejemplo, $P_C = 0{,}5$).

Para ilustrar esta operación, consideremos los primeros dos individuos de la *«mating pool»* ilustrada más arriba:

Padre 1	Padre 2
011	110

Tabla 4. Individuos a ser cruzados.

La operación de cruce comienza seleccionando aleatoriamente un número entre 1 y L – 1 usando una distribución uniforme de probabilidad. Existen L – 1 = 2 locaciones intersticiales entre las posiciones del string de longitud L = 3. Supongamos que la posición intersticial 2 es seleccionada. Esa posición se convierte en el denominado «punto de cruce». Cada padre es entonces partido en este punto de cruce, generándose así un fragmento de cruce y un remanente.

Los fragmentos de cruce de los padres 1 y 2 son los siguientes:

Fragmento de cruce 1	Fragmento de cruce 2
01-	11-

Tabla 5. Fragmentos de cruce de ambos padres.

Los remanentes de los padres 1 y 2 son:

Remanente 1	Remanente 2
--1	--0

Tabla 6. Remanentes de ambos padres.

Luego entonces combinamos el remanente número 1 (—1) con el fragmento de cruce número 2 (11-) para crear el descendiente (111). Ejecutamos luego el proceso análogo para crear el descendiente 010. Los dos descendientes se muestran en la siguiente tabla:

Descendiente 1	Descendiente 2
111	010

Tabla 7. Individuos descendientes creados mediante la operación de cruce.

Como para este ejemplo la probabilidad de cruce es P_c = 0,5 entonces podemos esperar que sólo dos de los cuatro individuos de la *«mating pool»* participen en la operación de cruce. Si esto se cumple, entonces los dos individuos restantes que no han participado en el cruce son transferidos a la nueva generación. Como podemos apreciar, la *«mating pool»* es un paso intermedio entre la generación actual (la 0) y la siguiente (la 1). Así obtenemos la siguiente tabla:

	Generación 0			"Mating pool" creado luego de la reproducción	Luego del cruzamiento (generación 1)			
i	String X_i	Fitness $f(x_i)$	$f(x_i)/?f(x_i)$	"Mating pool"	$f(x_i)$	Punto de cruce	X_i	$f(x_i)$
1	011	3	.25	011	3	2	111	7
2	001	1	.08	110	6	2	010	2
3	110	6	.50	110	6	-	110	6
4	010	2	.17	010	2	-	010	2
Total		12			17			17
Peor		1			2			2
Promedio		3.00			4.25			4.25
Mejor		6			6			7

Tabla 8. Generación creada luego de las operaciones de reproducción y de cruce.

Los cuatro individuos en la penúltima columna de la tabla anterior son la nueva población creada como resultado de las operaciones de reproducción y cruce. Estos cuatro individuos constituyen la generación 1 de esta ejecución del algoritmo genético.

Al evaluar la aptitud de esta nueva población, vemos que el mejor individuo de la generación 1 tiene un valor de aptitud de 7, mientras que el mejor individuo de la generación 0 poseía una aptitud de sólo 6. El cruzamiento hizo algo nuevo y, en este ejemplo, el nuevo individuo tiene una aptitud mayor que la de sus dos padres.

Al comparar la nueva población de la generación 1 contra la anterior población de la generación 0, encontramos lo siguiente:

- La aptitud promedio de la población se ha ampliado de 3 a 4.25.
- El individuo más apto de la población incrementó su aptitud de 6 a 7.
- El individuo menos apto de la población incrementó su aptitud de 1 a 2.

Este ejemplo ilustra cómo el algoritmo genético, usando las operaciones de reproducción y de cruce, puede crear una población con una aptitud promedio incrementada y con individuos mejorados.

Como podemos observar, el cruzamiento permite realizar una búsqueda explotadora (en profundidad), que permite explotar las mejores características de que dispone la población actual. Se encarga de realizar un intercambio estructurado de información.

4.4 Criterio de terminación o parada

El algoritmo genético luego ejecuta estas operaciones en forma iterativa sobre cada generación de individuos para producir nuevas generaciones hasta que cierto criterio de terminación o parada es satisfecho. Este criterio a veces es expresado en términos del número máximo de generaciones a ser ejecutadas. Para problemas donde es posible encontrar una solución perfecta, el algoritmo puede terminar cuando ese individuo es encontrado.

En este ejemplo, la mejor estrategia de negocio en la nueva generación (la 1) es:

- vender hamburguesas a 50 centavos (en vez de 10 pesos),
- servir bebida cola (en vez de vino), y
- ofrecer servicio rápido (en vez de lento).

Por lo tanto, la estrategia 111, que produce 7 pesos en beneficios por semana, es la estrategia óptima. Si sabemos que 7 pesos es el máximo global para el beneficio, entonces podemos terminar el algoritmo genético en la generación 1 en este ejemplo.

4.5 Designación de resultado

Un método para la designación de resultado para una ejecución del algoritmo genético es designar el mejor individuo en la generación actual en el momento de la terminación como el resultado del algoritmo genético. Por supuesto que una ejecución típica del algoritmo genético no terminará en la primera generación como sucede en este ejemplo sencillo. Por lo general, las ejecuciones rondan las decenas, los cientos o los miles de generaciones.

4.6 Operación de mutación

La operación de mutación es también usada en el algoritmo genético convencional que opera sobre strings de longitud fija. La frecuencia con la que se aplica dicha operación es controlada por un parámetro llamado «probabilidad de mutación», P_m. La mutación es llevada a cabo con poca frecuencia durante el trabajo del algoritmo genético. Esta operación es asexual en el sentido de que opera sobre sólo un individuo. El operador de mutación se aplica gen a gen a todos los descendientes generados por cruzamiento. Se genera un número aleatorio n_a por cada gen de cada descendiente. El allelo del gen respectivo es cambiado sólo si n_a d» P_m. Si el alfabeto es binario, el carácter simplemente es complementado.

No hubo mutación en el ejemplo anterior. Sin embargo, si el individuo 4 (010) hubiese sido elegido para mutación en la posición 2, el resultado hubiese sido el string 000. Podemos ver entonces que la operación de mutación posee el efecto de incrementar la diversidad genética de la población al crear el nuevo individuo 000.

La mutación permite entonces desarrollar una búsqueda exploradora (a lo ancho), que se encarga de explorar nuevos dominios en busca de mejores soluciones. Proporciona una garantía de accesibilidad para todos los puntos del espacio de búsqueda.

La mutación es una operación secundaria que es potencialmente útil para devolver la diversidad pérdida en una población. Por ejemplo, en las generaciones tempranas de una ejecución del algoritmo genético, un valor de 1 en una posición particular de un string puede estar fuertemente asociado a una mejor performance. Esto implica que, comenzando con puntos aleatorios en el espacio de búsqueda, el valor 1 en esa posición puede producir un mejor valor de aptitud. Como consecuencia de una más alta aptitud asociada con un valor de 1 en esa posición particular del string, el efecto de la operación de reproducción puede eliminar la diversidad genética al punto de que el valor 0 desaparezca en esa posición para toda la población.

Sin embargo, el óptimo global puede tener un 0 en esa posición del string. Una vez que la búsqueda se focaliza en la parte del espacio de búsqueda que realmente contiene el óptimo global, un valor de 0 en esa posición puede ser lo requerido para alcanzar el óptimo global. Esto es una forma de decir que el espacio de búsqueda no es lineal. Esta situación no es hipotética dado que casi todos los problemas en los cuales podemos interesarnos no son lineales. La mutación provee un camino para restaurar la diversidad genética perdida.

5. Diagrama de flujo del algoritmo genético básico

Los tres pasos en la ejecución del algoritmo genético que opera sobre strings de caracteres de longitud fija pueden ser sintetizados del siguiente modo:

1. Crear en forma aleatoria una población inicial de strings de caracteres de longitud fija.
2. Ejecutar iterativamente los siguientes subpasos sobre la población de strings hasta que cierto criterio de parada sea satisfecho:
   a. Evaluar la aptitud de cada individuo en la población.
   b. Crear una nueva población de strings aplicando por lo menos las primeras dos de las siguientes tres operaciones (las operaciones son aplicadas sobre los strings individuales de la población de acuerdo a cierto criterio de selección):
      i. Copiar strings existentes a la nueva población.
      ii. Crear dos nuevos strings recombinando genéticamente al azar substrings elegidos de dos strings existentes.
      iii. Crear un nuevo string de un string existente mutando al azar el caracter situado en una determinada posición del string original.
3. El mejor string que haya aparecido en cualquier generación (es decir, el mejor individuo hasta el momento) es designado como el resultado del algoritmo genético para la ejecución. Este resultado puede representar una solución perfecta o aproximada para el problema.

La siguiente figura muestra un diagrama de flujo de estos pasos del algoritmo genético convencional operando sobre strings. El índice *i* se refiere a un individuo en una población de tamaño *M*. La variable *GEN* es el número de la generación actual.

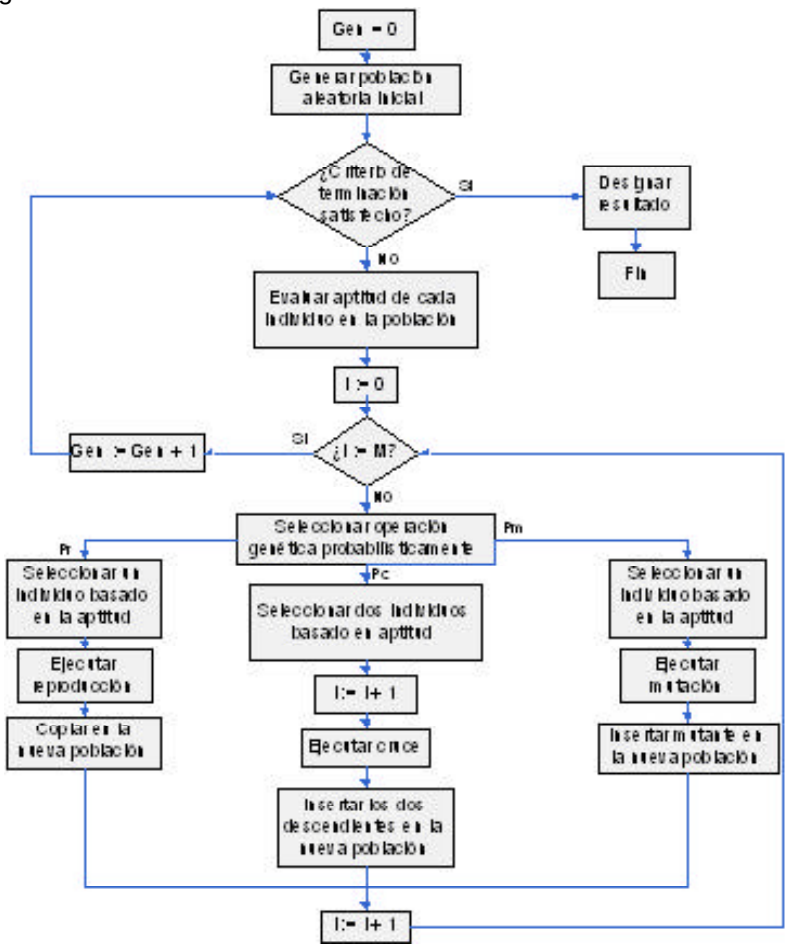

Figura 1. Diagrama de flujo del algoritmo genético básico.

Existen numerosas variaciones menores sobre el algoritmo genético básico. Este diagrama de flujo es meramente una versión. Por ejemplo, el número de veces que un operador genético es ejecutado durante una generación es a veces determinado por un número explícito por cada generación, en vez de determinado probabilísticamente como lo muestra este diagrama.

Este diagrama tampoco muestra en forma explícita la creación del «mating pool», sólo a los fines de simplificar la representación.

6. Algoritmos genéticos como un método débil

Es importante notar que el algoritmo genético trabaja de un modo independiente del dominio sobre los strings de caracteres de longitud fija de la población. Por este motivo, constituyen un «método débil». El algoritmo genético busca el espacio de strings en busca de los más aptos. Para guiar esta búsqueda, sólo utiliza valores numéricos de aptitud asociados con los puntos explícitamente probados del espacio de búsqueda. Sin importar el dominio específico del problema, el algoritmo genético lleva a cabo su búsqueda ejecutando las mismas operaciones generación tras generación.

El usuario puede utilizar conocimiento específico del dominio para seleccionar el esquema de representación y la medida de aptitud, como así también puede utilizar juicio adicional para seleccionar el tamaño de la población, el número de generaciones, los parámetros que controlan la probabilidad de ejecutar varias operaciones, el criterio para terminar una ejecución, y el método para designar un resultado. Todas estas opciones pueden influenciar cuan bien trabaja el algoritmo genético en un dominio particular. De todos modos, el principal punto es que el algoritmo genético es un medio independiente del dominio para buscar rápidamente puntos de alta aptitud en un espacio de búsqueda desconocido.

7. Cuatro pasos principales en la utilización de los algoritmos genéticos

Los cuatro pasos principales para usar el algoritmo genético convencional sobre strings de caracteres de longitud fija son:

- *Determinar el esquema de representación:* el esquema de representación es un mapeo que expresa cada posible punto del espacio de búsqueda del problema como un string de caracteres de longitud fija. La especificación de dicho esquema requiere seleccionar la longitud *L* del string y el tamaño *K* del alfabeto (el cual por lo general es binario). Seleccionar el mapeo entre los cromosomas y los puntos del espacio de búsqueda es a veces fácil y es a veces muy difícil. Seleccionar una representación que facilite la solución del problema a través del algoritmo genético generalmente requiere un conocimiento considerable del problema y buen juicio.
- *Determinar la medida de aptitud:* la medida de aptitud asigna un valor de aptitud o *fitness* a cada string de caracteres de longitud fija en la población. La medida de aptitud es generalmente inherente al problema y debe ser capaz de evaluar cada string que encuentre.
- *Determinar los parámetros y variables que controlan el algoritmo:* los parámetros primarios para controlar el algoritmo genético son el tamaño de la población *(M)* y el máximo número de generaciones a ejecutar *(G)*. Los parámetros secundarios, tales como p_r, p_c y p_m, controlan las frecuencias de reproducción, cruce y mutación, respectivamente.
- *Determinar el criterio de terminación de la ejecución y el modo de designar un resultado:* esto ya fue nombrado más arriba en este apéndice.

Una vez que estos pasos han sido completados, el algoritmo genético puede ser ejecutado.

8. Problemas asociados a los algoritmos genéticos

8.1 Problemas de convergencia estancada y de convergencia prematura

Es crucial mantener en todo momento una población de tamaño adecuado y de adecuada diversidad de individuos y bondades, a fin de evitar los dos enemigos de la convergencia global: la convergencia estancada y la convergencia prematura.

La necesidad de que haya diversidad de individuos es fácil de entender: con poca variedad de individuos hay poca variedad de esquemas, a causa de ello el operador de cruce pierde casi por completo la capacidad de intercambio de información útil entre individuos, y en definitiva, la búsqueda se estanca. Es en ese momento cuando se da la denominada convergencia estancada.

En algún momento puede ocurrir que un individuo o un grupo de ellos obtengan una aptitud notablemente superior a la de los demás. Esto es especialmente probable en las fases tempranas de la evolución, en las cuales de entre una población de individuos mediocres suele surgir un buen candidato por aplicación de los operadores genéticos. En tal circunstancia existe el riesgo de que se produzca una evolución en avalancha: al incrementar los individuos más aptos su presencia en la población, por ser ésta finita la diversidad dismi-

nuye, ello hace que en la siguiente generación se favorezca aún más a los individuos más aptos hasta que éstos terminan dominando por completo la población. A esto se le conoce como el fenómeno de los superindividuos. Habitualmente ocurre que tales superindividuos sólo son los más aptos en cierto momento, pero no los más aptos absolutos (téngase en cuenta que la falta de diversidad estanca la búsqueda), lo que en última instancia provoca una convergencia prematura del algoritmo genético, habitualmente hacia un subóptimo. La única circunstancia en la que este fenómeno es tolerable e incluso deseable es en las fases tardías de la evolución, cuando el algoritmo genético ha «localizado» correctamente la solución óptima, pero nunca antes.

Como solución a este problema, se puede demostrar que, adicionando al ciclo evolutivo la imposición de considerar válido un ciclo si y sólo si hay una mejora absoluta en la bondad media de la población (requisito de contractividad), la convergencia está siempre asegurada, independientemente de la población inicial. En caso de que no produzca la mejora, ese ciclo no se cuenta.

De lo anterior surge que, si bien la convergencia puede ser independiente de la población inicial, la velocidad de convergencia puede ser fuertemente afectada por la misma (además de la influencia de los valores de los otros parámetros del algoritmo).

8.2 Problemas de representación

La representación es un factor clave en el trabajo del algoritmo genético debido a que estos algoritmos directamente manipulan una representación codificada del problema y porque el esquema de representación puede limitar severamente la ventana a través de la cual el sistema observa su mundo. El algoritmo genético convencional que opera sobre strings de longitud fija es capaz de resolver muchos problemas. Sin embargo, el uso de dichos strings también deja algunos temas sin resolución.

En muchos problemas, la representación más natural de una solución es un programa de computadora jerárquico en lugar de un string de caracteres de longitud fija. Por lo general, el tamaño y la forma de dicho programa que resolverá el problema dado no son conocidos con anticipación, por lo que el programa debe tener el potencial para cambiar su tamaño y su forma. Es difícil y antinatural representar programas de computadora de tamaños y formas dinámicamente variables mediante strings de caracteres de longitud fija.

Los esquemas de representación basados en dichos strings no proveen la estructura jerárquica que es central para la organización de programas de computadora (en programas y subrutinas) y para la organización de conducta (en tareas y subtareas). Tampoco proveen una forma conveniente de representar procedimientos computacionales arbitrarios ni proveen mecanismos para incorporar iteración o recursividad cuando estas capacidades son deseables o necesarias para resolver un problema.

Además, estos esquemas de representación no poseen variabilidad dinámica, ya que la selección inicial de la longitud del string limita desde el comienzo el número de estados internos del sistema, limitando así lo que el sistema puede aprender.

Así llegamos a la conclusión de que la predeterminación del tamaño y la forma de las soluciones y la preidentificación de componentes particulares de las soluciones constituyen un obstáculo para el aprendizaje en sistemas computacionales.

Es aquí donde la programación genética nace como una alternativa de solución para este problema.

Apéndice B - Métodos de selección

Existen diversos métodos de selección que pueden ser utilizados durante la selección de progenitores y de sobrevivientes. Algunos de ellos son:

- *Selección elitista:* se garantiza la selección de los *k* miembros más aptos de cada generación.
- *Selección proporcional a la aptitud:* los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.
- *Selección por ruleta por bondad:* una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. Por lo tanto, los valores de la función bondad se utilizan como probabilidades de selección. Conceptualmente, esto puede representarse como un juego de ruleta -cada individuo obtiene una sección de la ruleta, pero los más aptos obtienen secciones mayores que las de los menos aptos. Luego la ruleta se hace girar, y en cada ocasión se elige al individuo que «posea» la sección en la que se asiente la bola que viaja por la ruleta.
- *Selección por ruleta por orden:* los individuos se ordenan según su bondad, y los números de orden resultantes se utilizan como probabilidades de selección para formar la muestra. La selección se basa en este ranking, en lugar de las diferencias absolutas en aptitud (como lo es en el caso de la ruleta por bondad). La ventaja de este método es que puede evitar que individuos muy aptos ganen dominio al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.
- *Selección por torneo:* se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.
- *Selección por torneo binario:* un caso particular del método de selección anterior, en el cual se eligen aleatoriamente *k* pares de individuos de la población base, y se constituye la muestra seleccionando el mejor de cada par.