Universidad Nacional Autónoma de México
Posgrado en Ciencia e Ingeniería de la Computación

Análisis de la neutralidad en Programación Genética

T E S I S

para obtener el grado de Maestro en Ciencias de la Computación

Autor:
Esteban Ricalde González

Directora:
Katya Rodríguez Vázquez

2008, Ciudad Universitaria, México

A mis padres que me educaron para luchar por mis sueños y me apoyaron para alcanzar éste.

A Elisa que se mantuvo a mi lado en las buenas y las malas, recordandome siempre que el esfuerzo valía la pena.

Agradecimientos

Quiero agradecer primero a la doctora Katya Rodríguez, directora de esta tesis. Sus enseñanzas, sugerencias e infinita paciencia permitieron que el presente trabajo llegara a buen término.

También deseo expresar mi reconocimiento a los doctores Pedro Pablo González, Efrén Mezura, David Arturo Rosenblueth y Christopher Stephens. Las valiosas aportaciones realizadas por ellos enriquecieron diferentes aspectos de esta tesis y me permitieron vislumbrar lo que requiere un trabajo de investigación.

Así mismo, agradezco a Alejandro Pérez sus precisas y provechosas observaciones para los temas Evolución Molecular y Teoría Neutral.

Gracias al doctor Boris Escalante, coordinador del posgrado en ciencias e ingeniería de la computación de la UNAM, por siempre dedicar un tiempo para escuchar mis inquietudes y tener paciencia para esperar el término de este proceso.

Mi gratitud quedará con Lulú, Cecilia, Amalia y Diana quienes hicieron hasta lo imposible para apoyarme en cada etapa de la maestría.

De mis compañeros de estudio agradezco a Cristian y Edgar por dedicar tiempo a revisar mis códigos, experimentos y por los consejos cuando las cosas parecían no converger. También a Pedro, Ely, Gaby, Jorge, Edgar, Laura y Orlando por los días y noches de estudio conjunto, su apoyo y amistad.

Agradezco profundamente a mi abuela Zoila por cobijarme en su casa una temporada. Recuerdo con nostalgia las pláticas en el desayuno y su canto acompañado por el sonido de la marimba.

Por último agradezco la ayuda económica brindada por el Consejo Nacional de Ciencia y Tecnología (CONACYT) a través de la beca número 200722 y el apoyo otorgado por la Dirección General de Asuntos del Personal Académico (DGAPA) de la UNAM a través del Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT).

Esteban Ricalde González

Índice general

Índice de figuras

Índice de tablas

Resumen

La Programación Genética es una herramienta del Cómputo Evolutivo que utiliza una representación en forma de programas ejecutables. Debido a esto se puede utilizar para proponer soluciones a problemas de muy diversas áreas.

La Teoría Neutral de la evolución molecular plantea que muchas de las mutaciones que ocurren en el material genético al interior de las células de todo organismo vivo no tienen impacto en el funcionamiento celular y mucho menos en el funcionamiento del organismo completo. A pesar de que esta hipótesis sigue en discusión, múltiples investigadores han modificado diferentes elementos en el proceso del Cómputo Evolutivo inspirados en ella, llamándole a este nuevo enfoque neutralidad. Una de las mejoras observadas al utilizar esta modificación en el modelo evolutivo es evitar que los procesos de búsqueda se estanquen en óptimos locales.

Con el objetivo de demostrar que la Programación Genética se puede beneficiar de la neutralidad, se realizó una modificación para un problema clásico en el que la PG ha demostrado ser poco eficiente. Los resultados indican que el enfoque propuesto puede ser útil en la solución a otros problemas y que la neutralidad puede mejorar el comportamiento de la Programación Genética, así como también el de otros métodos de optimización.

Capítulo 1

Introducción

1.1. Motivación

En los últimos años, la neutralidad[1] se convirtió en un tema controversial para la comunidad que investiga el Cómputo Evolutivo. La controversia se debe a la existencia de evidencia contradictoria sobre su posible utilidad para mantener una continua diversidad en la población y evitar así el estancamiento en óptimos locales.

Son necesarios una recopilación sistemática sobre los diferentes trabajos que abordan este tema y un análisis detallado sobre las diferencias entre los resultados reportados.

Esta tesis pretende contribuir con un análisis de los efectos neutrales en Programación Genética. Como caso de estudio se analizará lo que sucede en un paisaje de aptitud[2] no-trivial (como lo es el trazo Santa Fe para el problema de la Hormiga Artificial) cuando se permite que aumente la cantidad de neutralidad existente.

La mejora en el desempeño de la Programación Genética en dicho paisaje de aptitud sería un resultado importante, debido a que Langdon y Poli [65] aclararon que el trazo Santa Fe presenta características parecidas al espacio de búsqueda de los programas reales.

1.2. Antecedentes

Motoo Kimura propuso en 1968 la Teoría Neutral de la Evolución Molecular. En ella se define una mutación neutral como aquella que cambia el genotipo sin afectar el fenotipo. Con base en resultados experimentales se postula que en los sistemas biológicos una fracción considerable de todas las mutaciones que ocurren son neutrales, siendo benéficas solamente

[1] En Programación Genética, la neutralidad tiene una estrecha relación con el código inactivo, en el capítulo 4 se trata a detalle este tema.

[2] Se llama paisaje de aptitud (*fitness landscape*) a la evaluación mediante la función de aptitud de todos los posibles individuos.

una pequeña fracción de las mutaciones no-neutrales. Según Kimura [57] son estas mutaciones neutrales las que permiten la diversidad de las poblaciones de especies.

Varios investigadores han tratado de estudiar la forma en la que la neutralidad opera en sistemas de Cómputo Evolutivo. Algunos de ellos han agregado elementos al proceso evolutivo artificial buscando obtener mejores soluciones a problemas complejos. En algunos casos, los resultados obtenidos son contradictorios.

Banzhaf [6] propuso la Programación Genética Binaria (*Binary Genetic Programming* o PGB) con redundancia en la codificación de operadores para algunos problemas de regresión simbólica. Concluyó que la separación de las operaciones de búsqueda y el uso de mapeos genotipo-fenotipo otorga mayor flexibilidad al proceso evolutivo que trabajar únicamente sobre fenotipos.

En 1996, Harvey y Thompson [44] estudiaron algunos efectos de redes neutrales en un problema de *hardware* evolutivo. Obtuvieron un buen resultado y llegaron a la conclusión de que para dicho problema usar neutralidad en el proceso evolutivo permite alcanzar un óptimo global sin preocuparse de la convergencia prematura.

En 2000, Shipman et al. [90] exploraron los beneficios de la neutralidad con una representación compleja redundante. A partir de resultados experimentales, concluyeron que el flujo neutral permite el descubrimiento de más fenotipos que en caso de usarse una codificación directa.

En el mismo año, Miller y Thomson [75] propusieron un Algoritmo Evolutivo redundante llamado Programación Genética Cartesiana (*Cartesian Genetic Programming* o PGC). Probaron este modelo con algunos problemas clásicos (incluyendo el problema de la Hormiga Artificial) y concluyeron que CGP es al menos igual de efectiva que otras formas de PG.

Smith et al. [91] analizaron los efectos de la presencia de redes neutrales en la evolución de una rutina para robots. Concluyeron que la existencia de redes neutrales en el espacio de búsqueda, las cuales deberían permitir al proceso evolutivo escapar de óptimos locales, no necesariamente provee alguna ventaja.

Yu y Miller han reportado el uso de modificaciones neutrales a CGP y al Algoritmo Genético Simple en diferentes paisajes de aptitud [109, 110, 111], concluyendo que la neutralidad mejora la probabilidad de éxito del proceso evolutivo.

Tres años más tarde, Collins [19] analizó los resultados de [110] y concluyó que para dicho problema, la afirmación de que la neutralidad es benéfica es falsa.

Chow [15] propuso una modificación a los Algoritmos Genéticos que utiliza individuos que contienen múltiples cromosomas en lugar de uno solo. Dicha modificación es una forma de representar mapeos genotipo-fenotipo. Probó este método en problemas conocidos y obtuvo buenos resultados.

Galván, Rodríguez y Poli [37] analizaron los efectos de la adición de un elemento extra en el conjunto de funciones de la PG para permitir neutralidad explícita en varios problemas de *hardware* evolutivo. Concluyeron que la neutralidad explícita tiene un mejor desempeño

general en términos de consistencia en el alcance de soluciones factibles.

En 2006, Downing [26] se valió de la neutralidad implícita de los Diagramas Binarios de Decisión (BDDs) para diseñar un Algoritmo Evolutivo que reduce el número de evaluaciones necesarias para evolucionar problemas de *hardware* evolutivo.

En el mismo año Vanneschi et al. [106] definen una serie de métricas sobre las redes neutrales aplicables a diferentes problemas y diversas representaciones. Dichas métricas fueron probadas en el problema de la paridad y concluyeron que dependiendo del conjunto de funciones seleccionado, la aptitud de las redes con mayor neutralidad mejora con respecto a las de menor.

1.3. Contribución

Este trabajo presenta una función que aumenta la cantidad de código inactivo para la representación en forma de árbol de la Programación Genética. Dicha función se utilizó en el problema de la Hormiga Artificial con el trazo Santa Fe con buenos resultados.

El desarrollo de esta tesis sirvió como base para la presentación de un artículo como *Late breaking paper* en *Genetic and Evolutionary Computation Conference* (GECCO) 2007 celebrado en UCL, Londres, Inglaterra en julio de 2007 [85] y en el Tercer Congreso Internacional de Computación Evolutiva (COMCEV) celebrado en la Universidad Autónoma de Aguascalientes, Aguascalientes, México en octubre de 2007 [86].

1.4. Objetivo

El objetivo general de la presente tesis es analizar la relación existente entre la neutralidad y la Programación Genética, tanto en las investigaciones reportadas en la literatura como en un paisaje de aptitud no-trivial (problema de la Hormiga Artificial).

Esto da pie a los siguientes objetivos específicos:

- Solucionar el problema de la Hormiga Artificial mediante el uso de técnicas de Programación Genética.
- Integrar conceptos de neutralidad en la metodología de la Programación Genética y analizar el efecto que dichos factores neutrales provocan en el desempeño del algoritmo.
- Comparar los resultados obtenidos con aquellos producidos en otras investigaciones.
- Analizar el comportamiento de la Programación Genética en la obtención de estos resultados.

1.5. Metodología de la investigación

Los pasos a seguir para el cumplimiento de los objetivos son los siguientes:

1. Implementación de la Programación Genética Simple.
2. Solución del problema de la Hormiga Artificial.
3. Integración de la neutralidad en la representación genética por medio de una función neutral.
4. Análisis y evaluación de resultados.
5. Comparación con otras implementaciones las cuales pueden o no considerar aspectos sobre neutralidad.

1.6. Estructura de la tesis

El presente documento está organizado en ocho capítulos, los cuales se describen brevemente a continuación:

El *capítulo 2* presenta un esquema general de los Algoritmos Evolutivos. En él se habla sobre los diferentes modelos que integran esta área del Cómputo Evolutivo y se hace una breve descripción de cada uno de ellos.

En el *capítulo 3* se hace una descripción con mayor grado de detalle sobre la Programación Genética dado que es la herramienta que se utilizará en la presente tesis. Se habla a detalle sobre las variaciones de los distintos operadores que se han diseñado buscando mejorar el desempeño del modelo evolutivo.

En el *capítulo 4* se realiza una recopilación de la información disponible sobre neutralidad. Se parte desde una descripción detallada de la Teoría Neutral de Kimura, pasando por los diferentes enfoques sobre evolución molecular que han surgido recientemente, siguiendo con un análisis de los distintos trabajos publicados sobre neutralidad en el área del Cómputo Evolutivo y finalizando con una profundización de los trabajos sobre neutralidad relacionados con Programación Genética.

El *capítulo 5* presenta las características del problema seleccionado como caso de estudio. Se habla sobre el origen del problema de la Hormiga Artificial. Se explica la dinámica de evaluación de los programas de control y se resumen los diferentes artículos que han abordado el problema.

El *capítulo 6* presenta una contribución al caso de estudio. Consiste en la adición al conjunto de funciones de un elemento que incorpora el concepto de neutralidad (tratado en el capítulo 4) para buscar mejorar el desempeño de la Programación Genética en dicho problema de prueba.

El *capítulo 7* contiene la descripción de los experimentos realizados, presenta los resultados obtenidos y la discusión generada tras dichos resultados.

Finalmente, el *capítulo 8* se reserva para las conclusiones generales, las conclusiones del caso de estudio y los trabajos a futuro que se podrían realizar sobre el problema tratado en el capítulo 5.

Capítulo 2

Algoritmos Evolutivos

En 1859 Charles Darwin escribió el libro en el cual se plantean las bases de la teoría que hasta el día de hoy es comúnmente aceptada como la mejor explicación de la evolución de las especies [21]. La clave central de su planteamiento es lo que él llamó selección natural.

Dicho mecanismo se puede interpretar como un proceso de competencia adaptativa al interior de las especies. El Cómputo Evolutivo surge como simulación computacional, una metáfora a dicha interpretación.

Este proceso de simulación resulta en técnicas estocásticas de aproximación que pueden ser aplicadas a una amplia gama de problemas (incluso problemas NP). Día con día crece el listado de áreas de aplicación del Cómputo Evolutivo; el recorrido toca áreas muy diversas, algunos ejemplos son: música [83, 81], control de tráfico aéreo [1], administración [13], tareas militares [14, 32], videojuegos [24, 68], finanzas [29, 108], diseño [39, 42, 71], meteorología [53], medicina [56, 67] y biología [105, 11].

Los *Algoritmos Evolutivos* (AE) son una rama del Cómputo Evolutivo que sólo toma en cuenta las técnicas que pretenden implementar los mecanismos inspirados en la evolución biológica. Existen otros métodos que se consideran parte del Cómputo Evolutivo pero que no surgen como metáfora de la evolución biológica sino de una formalización del funcionamiento de los AE o de conceptos de evolución distintos a los biológicos. Algunos de los más utilizados son:

- Evolución Diferencial (*Differential Evolution* o ED) [100].
- Algoritmos de Estimación de Distribuciones (*Estimation of Distribution Algorithms*) [66].
- Algoritmos Culturales (*Cultural Algorithms*) [84].

La Figura 2.1 muestra el origen de cada uno de los modelos considerados por el autor como Algoritmos Evolutivos. Los modelos enmarcados en gris son considerados los AE principales. La flecha indica que la Programación Genética tomó muchos elementos de los Algoritmos Genéticos.

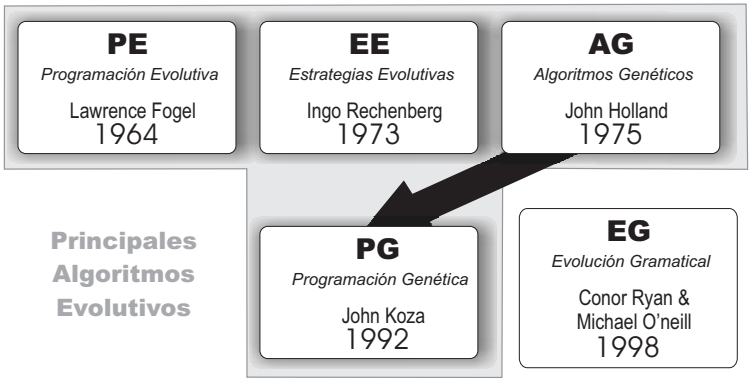

Figura 2.1: *Historia de los Algoritmos Evolutivos.*

Con el paso de los años se han desarrollado diferentes versiones de los Algoritmos Evolutivos. Sin embargo, todas comparten los siguientes componentes centrales [28]:

- **Codificación** de una solución al problema como un cromosoma.
- Existencia de una **función** para evaluar la **aptitud** o capacidad de sobrevivencia de los individuos.
- **Inicialización** de la población.
- Operadores de **selección**.
- Operadores de **reproducción**, divididos por Banzhaf [7] en: operadores de conservación y operadores de innovación. Son los encargados de la explotación y exploración del espacio de búsqueda.

Pero aún cuando los elementos básicos de los Algoritmos Evolutivos son puntos en común para todas sus variantes, existen muchos modelos diferentes. A continuación se abordarán brevemente los más utilizados.

Cabe aclarar que una de las principales diferencias entre las distintas técnicas agrupadas en los AE es la forma de representar los individuos de la población (máquinas de estados finitos, vectores de valores reales, cadenas binarias, árboles sintácticos, etc.).

2.1. Programación Evolutiva

La Programación Evolutiva (PE) fue propuesta en 1962 por Fogel (Lawrence) [51]. Surgió como una metáfora de los cambios de comportamiento a nivel especie y tiene como objetivo posibilitar el aprendizaje automático. Parte del principio de que el comportamiento inteligente requiere de la acción compuesta por la capacidad de predecir el ambiente circundante y de la traducción de dichas predicciones en una respuesta adecuada en busca de un objetivo dado.

Un ambiente simulado puede describirse como una secuencia de símbolos tomados de un alfabeto finito. El problema se reduce entonces a crear un algoritmo que sea capaz de operar en la secuencia de símbolos observados a un tiempo t, de tal manera que pueda producir un símbolo de salida que maximice el beneficio del algoritmo para el próximo símbolo a aparecer en el ambiente y una función de ganancia bien definida. Según su creador, las máquinas de estado finito (Autómatas Finitos ó AF) proveen una representación útil para esto.

La Programación Evolutiva opera sobre AF de la siguiente manera [30]:

1. Generación de la población inicial: Se construye una población padre de AF de manera aleatoria.

2. Evaluación: Los padres son expuestos al medio ambiente. Para cada máquina, conforme cada símbolo de entrada se vaya presentando, el símbolo de salida producido se compara con el siguiente símbolo de entrada. El valor de esta predicción es medido con respecto a la función de aptitud dada (todos ceros, error absoluto, error cuadrático).

3. Mutación: Se crean autómatas hijos mediante la mutación aleatoria de cada padre. Hay cinco métodos de mutación:

   a) Cambiar un símbolo de salida.

   b) Cambiar la transición de un estado.

   c) Agregar un estado.

   d) Eliminar un estado.

   e) Cambiar el estado inicial. Al ocurrir una mutación se selecciona de manera aleatoria cuál de estos métodos se va a aplicar.

4. Evaluación′: Se evalúa al nuevo conjunto de máquinas mediante un procedimiento parecido al descrito en el paso 2.

5. Selección: Aquellos autómatas que prevean una mayor ganancia se mantienen para ser utilizados como padres de la nueva generación. Típicamente la población de padres se mantiene del mismo tamaño.

6. Ciclo: Los pasos 3 al 5 se repiten hasta que se requiera hacer una predicción actual del siguiente símbolo (todavía no probado) del entorno. La mejor máquina se selecciona

para generar la siguiente predicción. El nuevo símbolo se agrega al ambiente y se regresa al paso 2.

La figura 2.2 muestra el funcionamiento general de la PE.

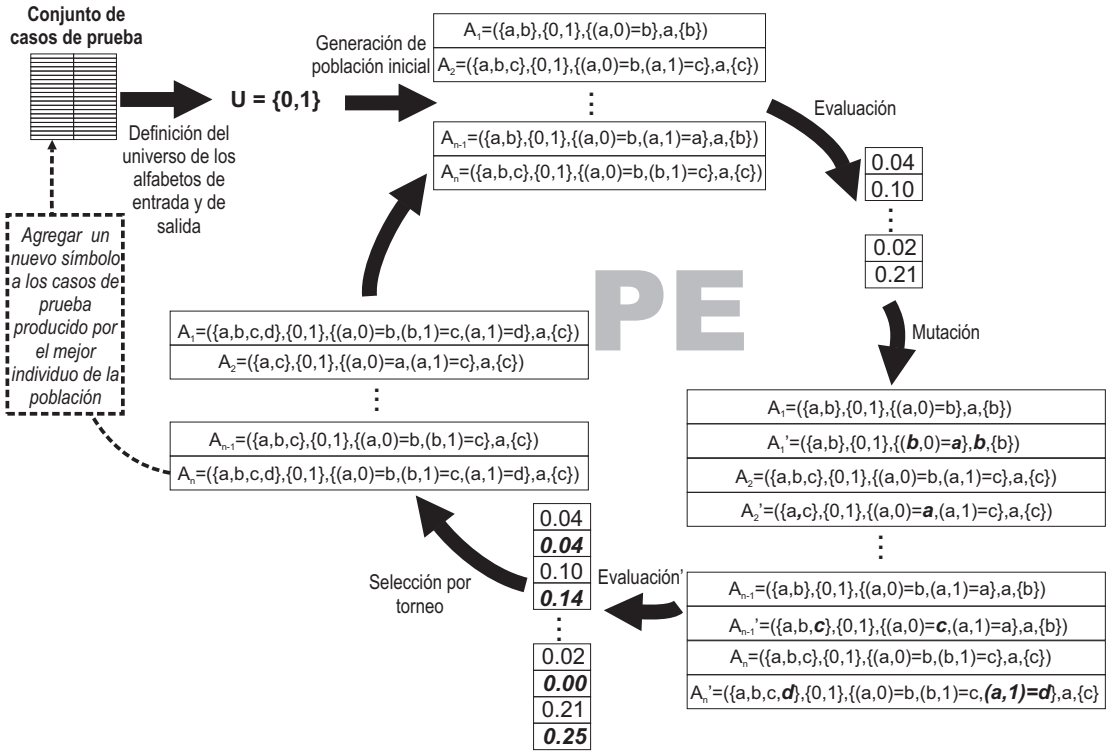

Figura 2.2: *Funcionamiento de la Programación Evolutiva.*

Pensando en el modelado de sistemas cuyo comportamiento cambia conforme el tiempo transcurre, el problema de la predicción se puede ver como una secuencia de problemas de optimización estática en los cuales la función objetivo varía en el tiempo. Este proceso puede extenderse a situaciones abstractas en las que la ganancia del comportamiento individual depende no solamente de una función de ganancia extrínseca, sino también del comportamiento de otros individuos de la población.

2.2. Estrategias Evolutivas

El modelo conocido como Estrategias Evolutivas (EE) fue propuesto en 1973 por Rechenberg [82]. Surgió como una metáfora de los cambios de comportamiento a nivel individual. En él sólo existe una cadena de valores reales que corresponde al comportamiento general de una población, no se piensa en términos de los individuos particulares de ésta.

Normalmente, las EE se implementan como un proceso de minimización. El método básico de funcionamiento es el siguiente [30]:

1. Definición del problema: Se define como encontrar el vector n-dimensional $\vec{x}$ de valores reales que está asociado al extremo de la función $F(x) : R^n \rightarrow R$.
2. Generación de la población inicial: La población inicial del vector padre $\vec{x}$ de tamaño $(i = 1, 2, \ldots, n)$ se selecciona de manera aleatoria.
3. Generación de nueva población: Un vector nuevo $\vec{x}'$ del mismo tamaño se crea del padre $\vec{x}$ sumándole a cada componente de $\vec{x}$ una variable aleatoria sacada de una distribución Gaussiana con cero de media y una desviación estándar seleccionada desde el inicio.
4. Selección: Determina cuál de los dos vectores se mantendrá mediante la comparación de $F(\vec{x})$ y $F(\vec{x}')$. El vector que posea un error menor se convierte en $\vec{x}$ de la próxima generación.
5. Ciclo: Se repiten los pasos 3 y 4 hasta que se alcance un error lo suficientemente pequeño o se agoten los recursos computacionales.

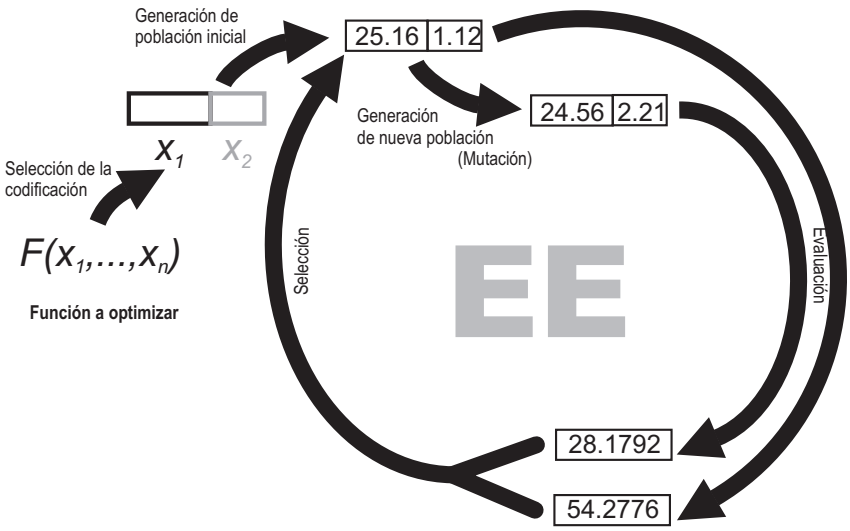

Figura 2.3: *Funcionamiento de las Estrategias Evolutivas.*

La figura 2.3 muestra el funcionamiento general del modelo propuesto por Rechenberg. Como se puede ver, las EE se caracterizan por una búsqueda un solo padre - un solo hijo. A esto se le llamó posteriormente $(1 + 1) - ES$ y consiste en que un padre crea un hijo y se elimina al que aporte una peor solución. Este enfoque presenta dos problemas:

- La desviación estándar constante hace que el procedimiento converja lentamente a buenas soluciones.

- La pobre naturaleza de la búsqueda punto a punto hace al procedimiento susceptible de estancarse en mínimos locales.

Para solucionar esto Schwefel propuso en 1977 [89] el uso de múltiples padres e hijos. Actualmente se utilizan dos enfoques para ello, llamados $(\mu + \lambda) - E$ y $(\mu, \lambda) - E$. En ambos se utilizan μ padres para crear λ hijos. En el primero de los enfoques tanto padres como hijos compiten por sobrevivir. En el segundo enfoque sólo los λ hijos compiten por sobrevivir mientras que los μ padres son reemplazados completamente cada generación.

2.3. Algoritmos Genéticos

Los Algoritmos Genéticos (AG) surgen como una metáfora de la herencia genética a nivel individual. Una solución a un problema asemeja un cromosoma de un individuo. Los operadores que se usan para generar nuevas soluciones son una simplificación de los procesos de replicación y variación del material genético que ocurren en el núcleo celular.

Al igual que muchos de los modelos considerados dentro de los Algoritmos Evolutivos, los AG son algoritmos de aproximación que pueden ser utilizados en problemas de optimización de todo tipo de funciones matemáticas y en clasificación.

Los AG se originaron en el trabajo realizado por biólogos que usaban computadoras para tratar de simular los sistemas genéticos naturales en la década de los 50. Sin embargo, no fue hasta 1975 que John Holland [45], a quien también le interesaba más estudiar la evolución en sí que generar una herramienta de optimización, consolidó una técnica general de optimización muy fácil de implementar y además, un teorema que explica claramente cómo y por qué dicha técnica funciona: El teorema del esquema.

Por todas las ventajas de esta herramienta y por la amplia difusión que los estudiantes de Holland le dieron, los Algoritmos Genéticos son el Algoritmo Evolutivo que más sobresalió durante los 80 y 90, a pesar de no haber sido el primero en surgir.

El AG simple (AGS, primera versión propuesta) utiliza una representación fija binaria y hace un amplio uso del operador de cruzamiento. Una condición de este modelo es que se debe buscar la forma de codificar soluciones como cadenas de dígitos binarios. Normalmente esto se realiza mediante la definición de los rangos mínimo y máximo y la precisión de cada una de las variables independientes. Definiendo con estos parámetros el número de dígitos binarios que corresponderán a cada una de dichas variables.

Un elemento clave que Holland aportó con el modelo del AG es la selección proporcional a la aptitud (*fitness-proportional selection*). Dicho método es un proceso estocástico que permite a individuos con un mayor valor de aptitud tener una mayor oportunidad de reproducirse que aquellos de menor aptitud. Se ha demostrado que este principio es responsable de la velocidad de la evolución.

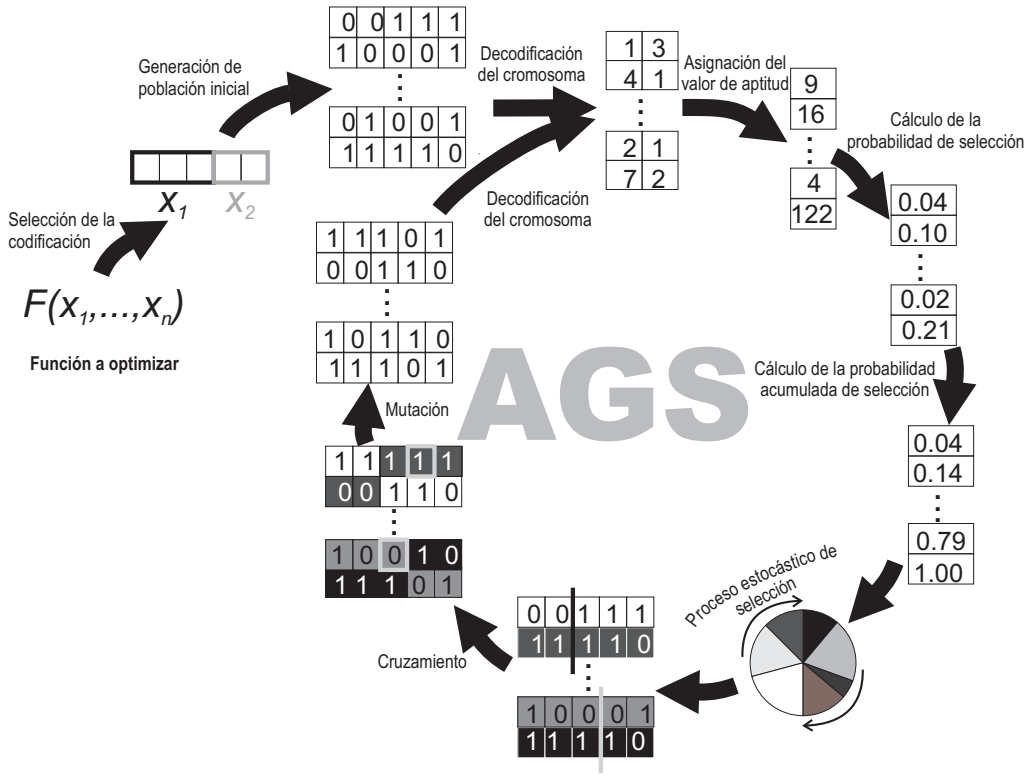

Figura 2.4: *Funcionamiento del Algoritmo Genético Simple.*

La Figura 2.4 muestra el funcionamiento general del AGS. A continuación se presenta una descripción más formal del modelo [28]:

1. Asignar $g = 0$.
2. Generar la población de la primera generación C_g.
3. Mientras no converja.
   - *a*) Evaluar la aptitud de cada individuo $\vec{C}_{g,i} \in C_g$ donde i va de 1 a N, siendo N el tamaño de la población.
   - *b*) $g = g + 1$.
   - *c*) Seleccionar los padres del conjunto C_{g-1}.
   - *d*) Combinar los padres seleccionados mediante el operador de cruza para generar la descendencia O_g.
   - *e*) Mutar la descendencia O_g.
   - *f*) Asignar el conjunto almacenado en O_g como la nueva población C_g.

Con el paso de los años se han hecho múltiples modificaciones a este algoritmo. Mencionemos algunas:

- Cambio en la representación a números enteros [73] y números racionales [73].
- Diferentes métodos de selección: por Torneo [74], Muestreo Estocástico Universal [5], Vasconcelos [61], por mencionar algunos.
- Diferentes operadores de cruzamiento: Cruza en dos puntos [38], Cruza uniforme [101], el método de corte y empalme (*cut & splice*) [40], entre otros.
- Diferentes métodos de mutación: por inversión [41], uniforme [73], no-uniforme [73], etc.
- Elitismo [22].
- Auto-adaptación [23].

Los Algoritmos Genéticos se han aplicado con éxito a diversos problemas de optimización y búsqueda. Algunos problemas requirieron rediseñar el proceso evolutivo e incluso incorporar información local (por ejemplo el problema del agente viajero [102, 41]) para lograr obtener un buen resultado. Los AG han demostrado ser una técnica eficiente y fácil de aplicar para solucionar problemas de áreas muy diversas.

2.4. Evolución Gramatical

La Evolución Gramatical (EG) fue propuesta por Ryan y O'Neill [88] como una forma (diferente a la Programación Evolutiva y la Programación Genética) de generar automáticamente programas. Este modelo está inspirado en el proceso biológico de la generación de una proteína mediante el material genético de un organismo.

La Figura 2.5 muestra el funcionamiento general de la EG mediante un diagrama de bloques. Básicamente se utiliza un Algoritmo Genético que trabaja sobre cadenas binarias de longitud variable para generar operadores de selección sobre las posibles opciones de transformación de reglas para un lenguaje especificado en forma *Backus-Naur* [4].

Este enfoque tiene como ventaja la facilidad para *adoptar* cualquier otro modelo del Cómputo Evolutivo como motor de búsqueda. Gracias a esto la EG puede fácilmente valerse de los nuevos adelantos hechos en el área para mejorar su desempeño en la generación automática de programas.

Ryan y O'Neill han probado la Evolución Gramatical para múltiples problemas. En algunos casos han mejorado el resultado producido por la Programación Genética.

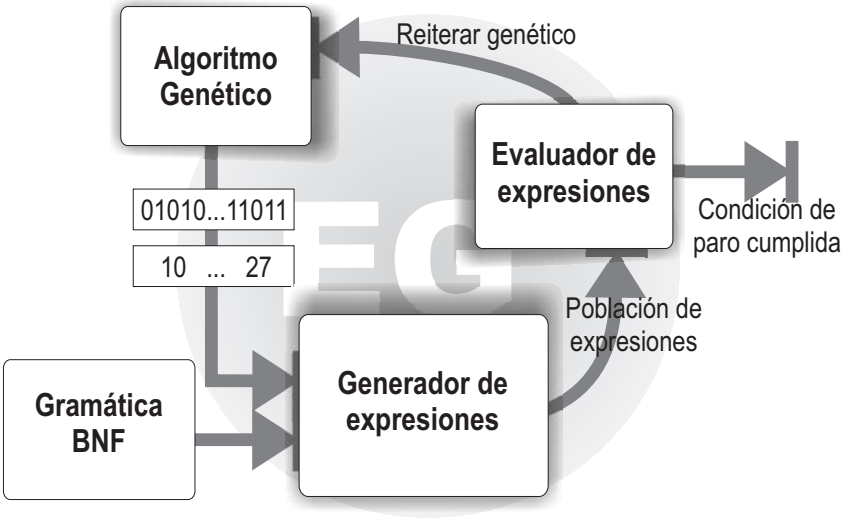

Figura 2.5: *Diagrama de bloques de la Evolución Gramatical.*

La Tabla 2.1 resume algunas de las características de los diferentes Algoritmos Evolutivos presentados en este capítulo.

Tabla 2.1: *Características de algunos Algoritmos Evolutivos.*

Modelo	Representación	Operadores	Tipo de selección	Tipo de población
Programación Evolutiva	Autómatas Finitos	Mutación	Torneo	Multipoblación
Estrategias Evolutivas	Números reales, tamaño fijo	Mutación (gaussiana)	Determinista	Un padre, un hijo $(1+1)-ES$
Algoritmos Evolutivos	Binaria, tamaño fijo	Cruza, mutación	Proporcional a la aptitud	Multipublación
Evolución Gramatical	Gramáticas BNF	Variable	Variable	Multipoblación

Tal como se mencionó en la sección 1.5, en el siguiente capítulo se abordará el AE conocido como Programación Genética. Se presentarán las semejanzas y diferencias que dicho modelo tiene con los Algoritmos Genéticos, la Programación Evolutiva y la Evolución Gramatical.

Capítulo 3

Programación Genética

La Programación Genética (PG) puede verse como una especialización de los Algoritmos Genéticos. De manera similar a estos, la PG se concentra en la evolución genética a nivel individual. La principal diferencia es el esquema de representación utilizado: los AG usan una cadena de bits o números para la representación de los individuos mientras que la Programación Genética los representa como programas ejecutables (normalmente en forma de árboles).

La meta de la PG es, en sentido figurado, *evolucionar* programas de cómputo.

3.1. Origen

Desde que Friedberg [33, 34] comenzó a dar solución a problemas sencillos mediante la modificación automática de programas de cómputo, uno de los objetivos del Cómputo Evolutivo ha sido *evolucionar* programas.

El modelo propuesto por Fogel, Owens y Walsh, que lleva el nombre de Programación Evolutiva (ver sección 2.3) fue un intento temprano de exploración en el espacio de programas simples, representados mediante Autómatas Finitos.

Tras la gran cantidad de problemas a los que los Algoritmos Genéticos dieron respuestas satisfactoriamente acertadas, múltiples investigadores buscaron cambiar la representación y/o los operadores de los AG para tratar con el problema de la inducción de programas.

Dos investigadores, Cramer y Koza, sugirieron que los árboles (estructuras de datos) debían ser usados como la representación de los individuos para codificar programas en el proceso evolutivo del AG. Sin embargo, fue Koza con su libro de 1992 [58], el primero en demostrar la importancia del dicha representación y que el proceso evolutivo artificial es una opción viable para la programación automática. Fue el mismo Koza quien llamó Programación Genética a esta variación de los AG.

3.2. Funcionamiento

Los operadores aplicados en la Programación Genética son similares a los operadores presentes en AG. Por esta razón los pasos definidos en la sección 2.3 también describen el comportamiento de la PG, tal como se puede apreciar en la Figura 3.1.

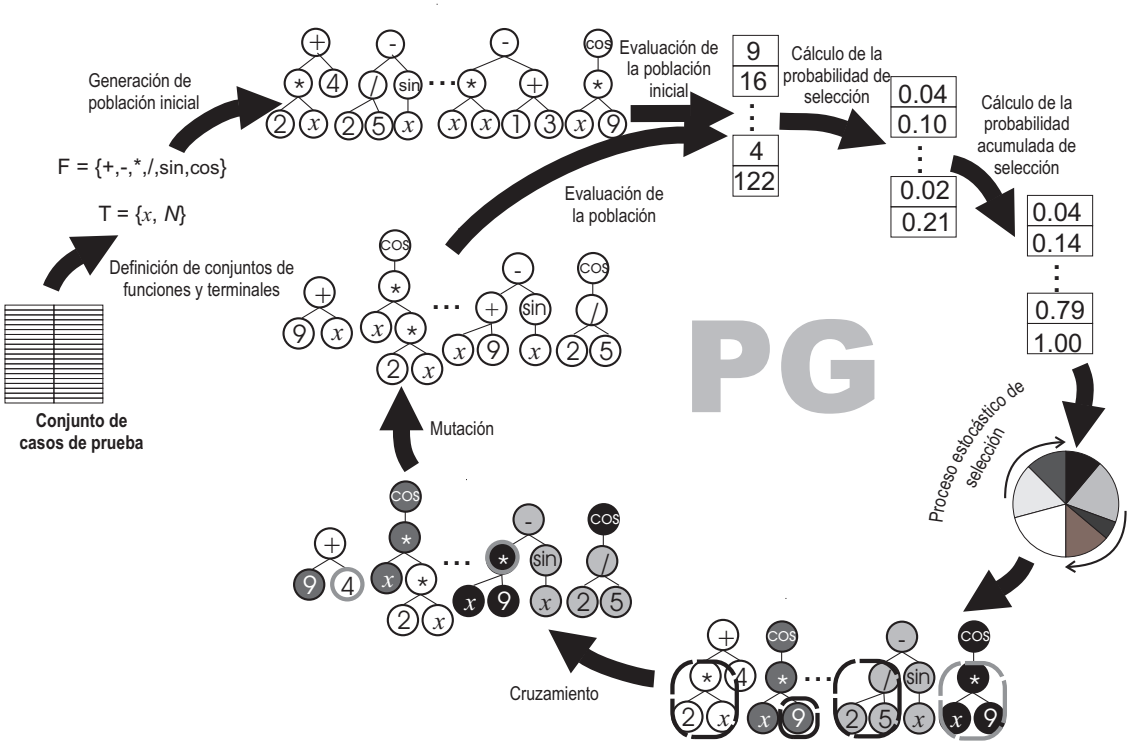

Figura 3.1: *Funcionamiento de la Programación Genética.*

A continuación se hará una descripción detallada de las etapas de la PG.

3.2.1. Representación

Como se comentó al inicio de este capítulo, la principal diferencia entre los Algoritmos Genéticos y la Programación Genética es que esta última utiliza representación en forma de programas ejecutables.

Banzhaf [7] identifica tres métodos diferentes para representar estos programas ejecutables: en forma de árbol, de manera lineal y en forma de gráficas. Estos métodos difieren en la estructura interna utilizada para almacenar el código y en el manejo de la memoria.

Las ciencias de la computación toman prestado el término "árbol" de la teoría de gráficas.

Una *gráfica* es una pareja $G = (V, A)$ donde V es un conjunto de puntos, llamados *vértices*, y A es un conjunto de pares de vértices, llamados *aristas*.

Ciclo es todo camino, es decir sucesión de aristas adyacentes, que no recorre dos veces la misma arista y regresa al vértice inicial. Un *árbol* es una gráfica que no tiene ciclos (*acíclico*) y que conecta a todos sus vértices (*conexo*).

Una *gráfica dirigida* es una gráfica donde A es un conjunto de pares ordenados de vértices, comúnmente llamados arcos. Un arco $a = (x, y)$ se considera dirigido de x a y. En dicho caso, y es sucesor directo de x y x es predecesor directo de y. De manera general, si un camino lleva de x a z entonces z es sucesor de x y x es predecesor de z.

Árbol dirigido es una gráfica dirigida que equivale a un árbol si se ignoran las direcciones de las aristas. Un *árbol con raíz* es un árbol dirigido en el que existe un vértice sin predecesores, llamado *raíz* y que cumple la siguiente propiedad: Cada vértice del árbol dirigido tiene solamente un predecesor directo.

En computación un árbol es una estructura de datos que equivale a un árbol con raíz. **Nodo** es la unidad sobre la que se construye el árbol y puede tener cero o más nodos hijos conectados a él. Se dice que un nodo a es **padre** de un nodo b si a es predecesor directo de b[1]. El vértice raíz se conoce como nodo raíz. Todo nodo que no tiene hijos se conoce como **hoja**.

Los árboles como estructuras de datos se pueden utilizar para almacenar, representar y procesar datos jerárquicos[2]. Es frecuente que los compiladores e intérpretes almacenen el código de los programas en árboles dado que estos son datos jerárquicos y su procesamiento se facilita al guardarlos con esta representación. A modo de ejemplo, la Figura 3.2 muestra un árbol equivalente a la expresión matemática $((x + 4) * (y - 5))/10$.

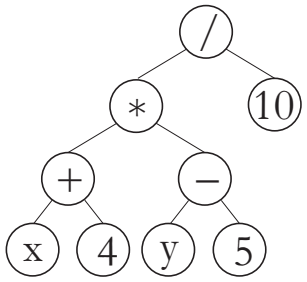

Figura 3.2: *Estructura de árbol de una función matemática.*

Por ello no es de extrañar que frecuentemente se utilice la Programación Genética con representación en forma de árbol. Cuando se trabaja con esta representación, la longitud de

[1]En dicho caso, también se dice que b es **hijo** de a.

[2]Los datos jerárquicos son aquellos que están organizados de manera que cada elemento del sistema presenta algún tipo de relación de dependencia con otro elemento, la excepción es el dato tope.

los individuos puede aumentar o disminuir durante el proceso evolutivo. Esto trae una serie de problemas que no presenta el Algoritmo Genético Simple y que se explicarán a detalle en el próximo capítulo.

Es común que se fijen dos valores de profundidad máxima:

- *Profundidad máxima inicial*: Tamaño máximo que pueden alcanzar los individuos en el proceso de generación de la población inicial (ver sección 3.2.3).

- *Profundidad máxima final*: Tamaño máximo que pueden alcanzar los individuos una vez que el proceso evolutivo ha iniciado. Este límite sirve para evitar que la memoria de la computadora se sature.

3.2.2. Conjuntos de terminales y funciones

Los conjuntos de funciones y terminales son los elementos base sobre los que se construye la Programación Genética. Estos conjuntos juegan diferentes papeles: los terminales proveen valores al sistema, mientras que las funciones procesan dichos valores.

Toda hoja es un miembro del conjunto de terminales. Por ejemplo las variables, las constantes y las funciones sin argumentos son terminales.

El conjunto de funciones está compuesto por sentencias, operadores y funciones que toman uno o más argumentos disponibles para el sistema de PG. Dependiendo del problema a resolver, el conjunto de funciones puede estar construido por funciones booleanas, aritméticas, trascendentes, sentencias condicionales, de control, de transferencia, de ciclos y/o funciones intrínsecas del problema a resolver.

La selección de los conjuntos de funciones y terminales es un paso muy importante. Estos deben contener los elementos suficientes para que la Programación Genética pueda resolver el problema. Esta característica se conoce como propiedad de *suficiencia* de dichos conjuntos. Si no se cumple esta propiedad, la PG no encontrará una buena solución. Caso contrario es que alguno de los conjuntos (o ambos) tenga demasiados elementos. Esto aumentará el espacio de búsqueda y la PG necesitará un mayor número de corridas para encontrar una solución satisfaciente. Por ello se puede afirmar que la tarea de seleccionar los conjuntos de funciones y terminales no es trivial.

Cada elemento del conjunto de funciones debe poder manejar los valores de todos los argumentos que recibe. Esto se conoce como la propiedad de *cerradura* y es muy importante manejarla de manera adecuada. De no hacerlo así es posible que el sistema colapse o que la solución obtenida no represente de manera adecuada lo modelado.

3.2.3. La población inicial

El primer paso de la Programación Genética es generar la población inicial. Esto significa crear una variedad de estructuras de programa para su posterior evolución. El parámetro de la profundidad máxima inicial se ocupa en esta etapa para definir el tamaño máximo permitido de los individuos.

Existen tres métodos para realizar la inicialización de las estructuras de los árboles:

- *Completo o Full*: Genera árboles completos de profundidad máxima. Partiendo desde el nodo raíz se selecciona de manera aleatoria un elemento del conjunto de funciones para ser insertado en el árbol. Esta operación se repite hasta llegar a la profundidad máxima. Al llegar a este punto, los elementos a seleccionar se toman del conjunto de terminales. La Figura 3.3 muestra un ejemplo de un árbol generado con el método completo con una profundidad máxima de tres niveles.

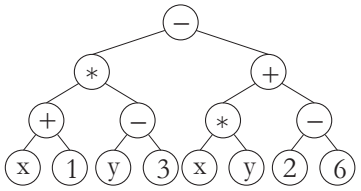

Figura 3.3: *Árbol creado con método completo.*

- *Creciente o Grow*: Genera árboles con forma irregular. Sólo el nodo raíz se genera usando el conjunto de funciones. Para el resto de ellos se seleccionan de manera aleatoria elementos de los conjuntos de funciones y terminales. Cuando una rama contiene un terminal se toma a este nodo como hoja (no importando si se alcanzó la profundidad máxima inicial). Si se alcanza la profundidad máxima, los nodos faltantes son seleccionados del conjunto de terminales. La Figura 3.4 muestra un árbol generado con el método creciente para una profundidad máxima de tres niveles.

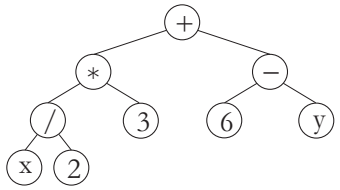

Figura 3.4: *Árbol creado con método creciente.*

- *Mitad y mitad o Ramped Half-and-Half*: Cuando se utiliza el método completo para generar la población inicial puede suceder que el conjunto de estructuras sea demasiado uniforme. Para evitarlo se creó este método. Lo que hace es dividir el número

de individuos a generar entre la profundidad máxima. Obteniendo así intervalos en los que se generarán individuos para cada una de las profundidades menores o iguales a la máxima, la mitad de las veces mediante el método completo y la otra mitad mediante el método creciente.

3.2.4. Cruzamiento

El operador genético denominado cruzamiento o cruza combina el material genético de dos árboles. Funciona mediante el intercambio de una parte de un elemento de la población seleccionado de manera aleatoria con otra parte de otro individuo seleccionado de la misma manera. Normalmente, a los árboles originales se les llama padres y a los que se obtienen tras la cruza se les llama hijos o descendencia.

Existen diferentes métodos para realizar el cruzamiento [7]:

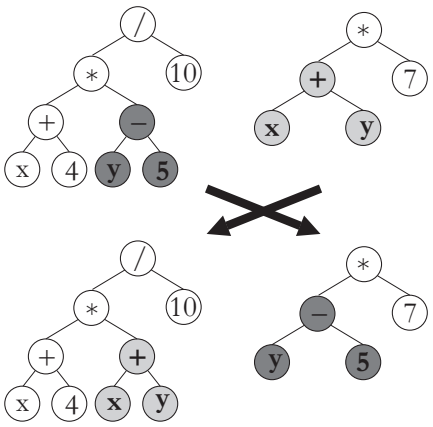

Figura 3.5: *Método de cruzamiento por intercambio de subárboles.*

- *Intercambio de subárboles o Subtree exchange crossover*: Se selecciona un subárbol de cada padre de manera aleatoria[3] y se intercambian los subárboles seleccionados entre los dos padres. La Figura 3.5 da un ejemplo del funcionamiento de este método.

- *Autocruzamiento o Selfcrossover*: Se puede considerar un caso particular del método anterior. Los padres son una copia del mismo árbol y entre los dos subárboles seleccionados para realizar la cruza cubren toda la estructura del individuo excepto la raíz. La Figura 3.6 da un ejemplo del funcionamiento de este método.

[3]Dicha selección se puede sesgar para que los terminales tengan una menor probabilidad de ser escogidos.

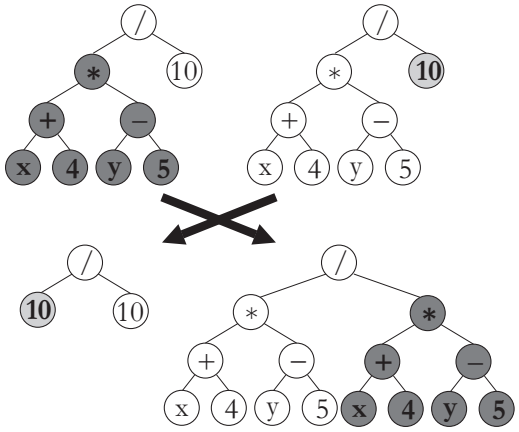

Figura 3.6: *Método de autocruzamiento.*

- *Cruzamiento de módulos o Module crossover*: Parecido al método de intercambio de subárboles pero no se seleccionan subárboles completos para realizar la cruza sino segmentos de tamaño variable (*módulos*) del árbol. La implementación de este método no es sencilla ya que se debe vigilar que no queden ligas sueltas y que no se rompa con la propiedad de cierre del conjunto de funciones. La Figura 3.7 da un ejemplo del funcionamiento de este método.

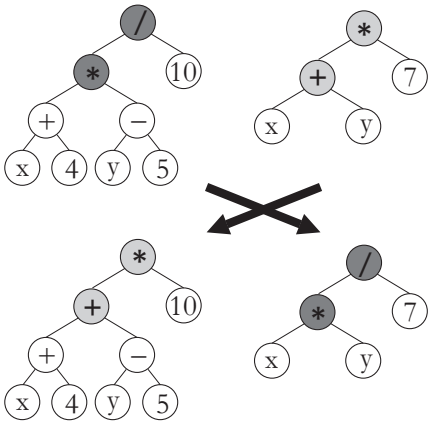

Figura 3.7: *Método de cruzamiento de módulos.*

3.2.5. Mutación

El operador de mutación se aplica a un solo individuo. Cada hijo producido por la cruza es sometido a mutación con una probabilidad definida por el usuario. Dependiendo del problema a resolver se puede utilizar también de manera individual tanto el operador de cruza como el de mutación (macromutación).

Existen diferentes métodos para realizar la mutación [7]:

- *Mutación de nodos o Point mutation*: Se selecciona un nodo de manera aleatoria. Si se trata de una hoja se intercambia por un elemento del conjunto de terminales diferente al original, en otro caso se intercambia por un elemento del conjunto de funciones diferente al original pero que tenga la misma aridad[4]. La Figura 3.8.b da un ejemplo del funcionamiento de este método de mutación sobre el árbol original de la Figura 3.8.a.

- *Intercambio o Permutation*: Se seleccionan dos hojas o dos nodos internos con misma aridad del individuo de manera aleatoria y se intercambian el uno con el otro. La Figura 3.8.c da un ejemplo del funcionamiento de este método de mutación sobre el árbol original de la Figura 3.8.a.

- *Mutación de subárboles o Subtree mutation*: Se selecciona un nodo cualquiera que se intercambia por un nuevo subárbol. Se debe verificar que el tamaño del nuevo individuo producido no rebase la profundidad máxima final. La Figura 3.8.d da un ejemplo del funcionamiento de este método de mutación sobre el árbol original de la Figura 3.8.a.

- *Mutación por alzamiento o Hoist*: Se selecciona un subárbol aleatorio del individuo (de profundidad mayor o igual a dos) y se intercambia al individuo por el subárbol seleccionado. La Figura 3.8.e da un ejemplo del funcionamiento de este método de mutación sobre el árbol original de la Figura 3.8.a.

- *Expansión o Expansion mutation*: Se selecciona una hoja de manera aleatoria y se intercambia por un nuevo subárbol. Se debe verificar que el tamaño del nuevo individuo producido no rebase la profundidad máxima final. La Figura 3.8.f da un ejemplo del funcionamiento de este método de mutación sobre el árbol original de la Figura 3.8.a.

- *Eliminación de subárboles o Collapse subtree mutation*: Se selecciona un nodo interior del árbol. Dicho nodo se elimina y se intercambia por un elemento del conjunto de terminales seleccionado de manera aleatoria. La Figura 3.8.g da un ejemplo del funcionamiento de este método de mutación sobre el árbol original de la Figura 3.8.a.

[4] **Aridad:** número de argumentos de entrada de la función.

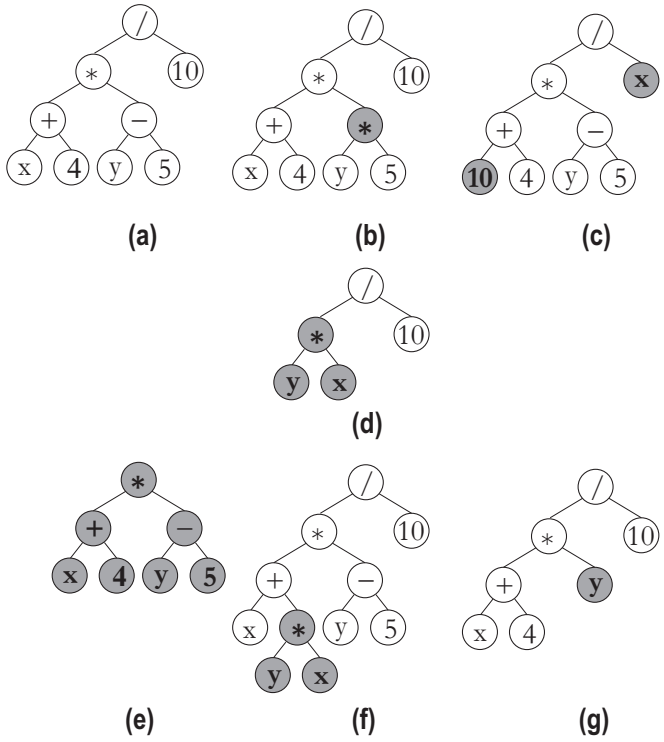

Figura 3.8: *Métodos para realizar la mutación.*

Probabilidad de mutación

Un aspecto a considerar con respecto a la mutación es la diferencia existente entre la probabilidad de mutación por nodo y la probabilidad de mutación por individuo.

La probabilidad de mutación por nodo está relacionada directamente con el método de mutación de nodos y la probabilidad de mutación por individuo con el método de mutación de subárboles. Estas relaciones se deben a que es más sencillo implementar cada tipo de mutación cuando se está utilizando la probabilidad que está relacionada con éste.

Pocos autores aclaran cuál de ellas utilizaron para producir sus resultados y vale la pena recalcar que la diferencia entre una y otra es abismal. Suponiendo por ejemplo que se está trabajando con una población de árboles binarios completos de profundidad cinco, una probabilidad de mutación por individuo del 100 % equivale aproximadamente a una probabilidad de mutación por nodo ligeramente mayor al 3 %.

Con el objetivo de aclarar más este punto se realizó un experimento sobre el caso de estudio planteado en el capítulo 5. Se fijó la probabilidad de cruza a 60 %, la profundidad inicial a dos y la final a cuatro. Se utilizó el método de selección por torneo (ver sección 3.2.7) con tamaño de torneo de tres individuos, un tamaño de la población de 150 individuos y se corrió el proceso de Programación Genética por 333 generaciones. Se seleccionaron siete probabilidades de mutación ($5, 10, 20, 40, 60, 80$ y 100).

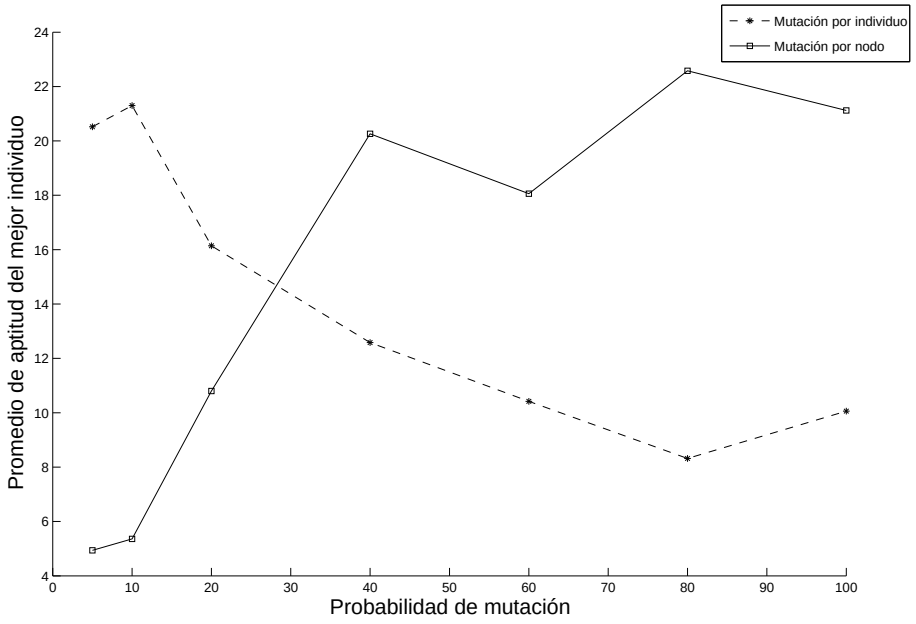

Figura 3.9: *Comparación del promedio de la aptitud del mejor individuo para mutación de nodos y mutación de subárboles por individuo.*

Con estos parámetros se ejecutaron 50 corridas para el método de mutación de nodos con probabilidad de mutación por nodo y la misma cantidad de corridas para el método de mutación de subárboles con probabilidad de mutación por individuo. La Figura 3.9 muestra el promedio de la aptitud del mejor individuo de cada una de las 50 corridas para ambas series. Cabe aclarar que se busca minimizar el valor de la función objetivo cuyo rango es $[0, 89]$.

Como se esperaba, el valor del promedio de la aptitud del mejor individuo decrece con valores muy altos de la mutación por individuo (80 %). Curiosamente, éste es el peor punto para la mutación por nodo. Analizando la gráfica se puede observar que en los puntos en los que la probabilidad de mutación por nodo alcanza valores bajos de aptitud, la probabilidad de mutación por individuo se encuentra en su pico más alto y viceversa, siguiendo lo que pareciera aproximarse a un comportamiento inverso.

3.2.6. La evaluación

El objetivo de tener una evaluación de la aptitud de los individuos es brindarle información a la PG sobre cuáles individuos deben tener una mayor probabilidad de multiplicarse y reproducirse y cuáles deben tener una mayor probabilidad de ser removidos de la población. Normalmente, la función de aptitud se calcula sobre un grupo de casos de prueba de la función.

La *aptitud estandarizada* es una función de aptitud o una transformación de ésta, en la cual cero es el valor asignado al mejor individuo.

La *aptitud normalizada* es una función de aptitud o una transformación de ésta, en la cual la aptitud siempre se encuentra entre cero y uno.

Cuando se cuenta con un conjunto de casos de prueba, una forma sencilla de evaluar el desempeño de los individuos es calcular la suma del valor absoluto de la diferencia entre la salida actual y la esperada (el error). Partiendo de que o_i es la salida del i-ésimo elemento del conjunto de prueba y que la salida de un programa p generado por la PG obtiene un valor p_i para la i-ésima entrada. Cuando n es el número de elementos del conjunto de casos de prueba, se tiene que la función de aptitud de error estándar f_p de p se puede obtener mediante:

$$f_p = \sum_{i=1}^{n} |p_i - o_i| \tag{3.1}$$

Tal como puede verse en (3.2), el error cuadrático es una forma de volver más grande la diferencia entre dos individuos con valores de aptitud parecidos entre sí. Esta fórmula es semejante a la ecuación para el error estándar, con la diferencia de que en lugar de calcular el valor absoluto la resta se eleva al cuadrado. Con esto, lo que se busca es aumentar la diferencia existente entre individuos parecidos entre sí. Este método se desempeña mejor que el anterior cuando la población es relativamente homogénea.

$$f_p = \sum_{i=1}^{n} (p_i - o_i)^2 \tag{3.2}$$

Otras funciones de aptitud comúnmente utilizadas en PG son:

- Número de píxeles que quedaron ajustados con la imagen original en una aplicación de ajuste de imágenes.
- Número de golpes con las paredes para un robot controlado por la PG para una tarea de evasión de objetos.
- Número de ejemplos clasificados correctamente para una tarea de clasificación.
- Desviación estándar de la diferencia entre la predicción y la realidad en una tarea de predicción.
- Dinero ganado por un agente controlado por PG en un juego de apuestas.
- Cantidad de comida encontrada por un agente artificial en una aplicación de vida artificial.

Existen otras funciones de aptitud menos comunes. Por ejemplo, en algunos métodos de co-evolución los individuos compiten los unos contra los otros sin un valor explícito de aptitud. En aplicaciones de juegos, el ganador en un juego tendrá una mayor probabilidad de reproducirse que el perdedor.

La función objetivo es un elemento completamente configurable según el problema al que se pretenda dar una respuesta mediante la Programación Genética. Incluso se puede utilizar un esquema de múltiples objetivos si se desea encontrar una solución que tome en cuenta varias funciones de aptitud.

3.2.7. La selección

Una vez aplicada la función de aptitud y determinada la calidad de un individuo, se deben escoger los elementos que se mantendrán en la población y desechar el resto. Esta tarea se denomina selección y es asignada a un operador especial, **el operador de selección**.

La selección es responsable de la velocidad de la evolución y frecuentemente es identificada como la culpable cuando ocurre la convergencia prematura en un Algoritmo Evolutivo. Existe una gran cantidad de métodos de selección (cada día surgen nuevos). Algunos de los más ocupados son:

- *Selección proporcional a la aptitud*: Método creado por Holland para los AG que se puede aplicar perfectamente en la PG. Mediante (3.3) se define la probabilidad de que cada uno de los individuos sea seleccionado para los procesos de cruza y mutación que darán lugar a la siguiente generación.

$$p_i = f_i / \sum_j f_j \tag{3.3}$$

  donde f_i es el valor de aptitud del i-ésimo individuo, f_j es el valor de aptitud del j-ésimo individuo y j recorre todos los individuos de la población.

- *Selección por rango*: Método basado en el ordenamiento de los individuos en base a la aptitud. La probabilidad de selección se asigna a los individuos en función de su rango en la población. Se pueden ocupar funciones de rango lineales y exponenciales. La probabilidad en el rango lineal se calcula mediante (3.4).

$$p_i = \frac{1}{N}\left[p^- + (p^+ - p^-)\frac{i-1}{N-1}\right] \tag{3.4}$$

  Donde p^-/N es la probabilidad de que el peor individuo sea seleccionado, p^+/N es la probabilidad de que el mejor individuo sea seleccionado, N es el tamaño de la población, i es la posición ocupada por el individuo en el ordenamiento de la población y la propiedad (3.5) se debe cumplir para que el tamaño de la población se mantenga constante.

$$p^- + p^+ = 2 \tag{3.5}$$

  Para el rango exponencial, la probabilidad se puede calcular mediante (3.6) donde c es una constante de sesgo y debe cumplir $0 < c < 1$.

$$p_i = \frac{c-1}{c^{N-1}}c^N - i \tag{3.6}$$

- *Selección por torneo*: Método basado en la competencia de un subconjunto de la población. Se escoge de manera aleatoria un número de individuos definido por el parámetro **tamaño del torneo**. Se realiza una competencia en este grupo. Seleccionando al mejor individuo del torneo.

  Una de las ventajas de este método es que permite al usuario ajustar la presión selectiva mediante el tamaño del torneo. Un valor pequeño causa una baja presión selectiva, mientras que un valor grande provoca una alta presión selectiva. Otra de las ventajas es que favorece la paralelización del modelo.

3.2.8. Elitismo

El elitismo busca evitar que los n_e mejores individuos de la población se pierdan. Antes de aplicar los operadores de selección, cruza y mutación se copian directamente los n_e mejores individuos a la siguiente generación, logrando con esto evitar que dichos individuos se pierdan.

El valor de n_e debe ser pequeño (normalmente uno o dos individuos), de lo contrario la población difícilmente variará con el paso de las generaciones.

3.3. Aplicación

La Programación Genética se ha utilizado para resolver problemas de diversas áreas que van desde el diseño de circuitos digitales [76] hasta la clasificación de empresas según su comportamiento en la bolsa de valores [48].

La regresión simbólica es un área donde la PG se ocupa frecuentemente [54]. Este éxito se debe a que en comparación con otras técnicas que sirven para resolver el mismo tipo de problemas, la Programación Genética tiene menos probabilidades de estancarse en óptimos locales (Escalado de colinas) y su salida es realmente una función matemática que no necesita del resto del modelo para funcionar (Redes Neuronales Artificiales).

Otra de las áreas donde se usa la PG es en la generación de programas simples[5] de control para agentes o robots que buscan resolver diferentes problemas:

- Navegación en un cuarto desconocido [87].
- Confrontación contra otro(s) agente(s) en competencias *deportivas* [18].
- Compra-venta de acciones en la bolsa de valores [112].
- Confrontación contra otro(s) agente(s) en ambientes de videojuegos [24].
- Etc...

[5]Para un conjunto de funciones limitado o pequeño.

Sin embargo la evolución de programas para un conjunto de funciones más grande ha sido algo para lo que la Programación Genética no ha dado una respuesta satisfactoria [20]. Esto se debe al enorme tamaño del espacio de búsqueda que los problemas de este tipo significan.

La Tabla 3.1 presenta las características de los Algoritmos Evolutivos incluyendo a la Programación Genética.

Tabla 3.1: *Características de los Algoritmos Evolutivos.*

Modelo	**Representación**	**Operadores**	**Tipo de selección**	**Tipo de población**
Programación Evolutiva	Autómatas Finitos	Mutación	Torneo	Multipoblación
Estrategias Evolutivas	Números reales, tamaño fijo	Mutación (gaussiana)	Determinista	Un padre, un hijo $(1+1)-ES$
Algoritmos Evolutivos	Binaria, tamaño fijo	Cruza, mutación	Proporcional a la aptitud	Multipublación
Evolución Gramatical	Gramáticas BNF	Variable	Variable	Multipoblación
Programación Genética	Programas ejecutables	Cruza, mutación	Proporcional a la aptitud	Multipublación

Existe una gran cantidad de investigadores que buscan mejorar las capacidades de la PG mediante la modificación de los pasos que se describieron en la sección 3.2. En el siguiente capítulo se presentará uno de los enfoques propuestos con dicha meta y la polémica que éste ha despertado.

Capítulo 4

Neutralidad

La Teoría Neutral de la evolución molecular plantea que muchas de las mutaciones que ocurren en el material genético celular de todo organismo vivo no tienen impacto en el funcionamiento celular y mucho menos en el funcionamiento del organismo completo. A pesar de que esta hipótesis sigue en discusión, múltiples investigadores han modificado diferentes elementos en el proceso del Cómputo Evolutivo inspirados en ella. Llamándole a este nuevo enfoque "neutralidad".

El presente capítulo profundiza en la teoría biológica de la neutralidad y los diferentes enfoques utilizados para tratar con ella al interior del Cómputo Evolutivo.

4.1. Teoría Neutral de la Evolución Molecular

4.1.1. Conceptos básicos

A continuación se definen algunos términos de biología molecular que se ocupan en las secciones 4.1.2, 4.1.3 y 4.1.4.

Gen: Segmento de ADN que contiene información biológica y codifica para un ARN y/o una molécula polipeptídica [10].

Ácido Desoxirribonucleico (ADN): Polinucleótido formado de unidades de desoxiribonucléotidos unidos covalentemente. Es la molécula que almacena la información hereditaria [3]. Las bases nitrogenadas de los desoxiribonucléotidos pueden ser de cuatro tipos: Adenina (A), Timina (T),Guanina (G) y Citosina (C) (ver Figura 4.1).

Ácido ribonucleico (ARN): Es una macromolécula compuesta de cadenas de nucleótidos que forman un polímero. El ARN es fundamental en el proceso de transcripción y traducción de información genética del ADN en proteínas, así como en la transferencia de ésta a los ribosomas. Un tipo de ARN (ARNm) actúa como mensajero entre el ADN y los ribosomas, otros forman porciones vitales de la estructura de los ribosomas, actuando como transpor-

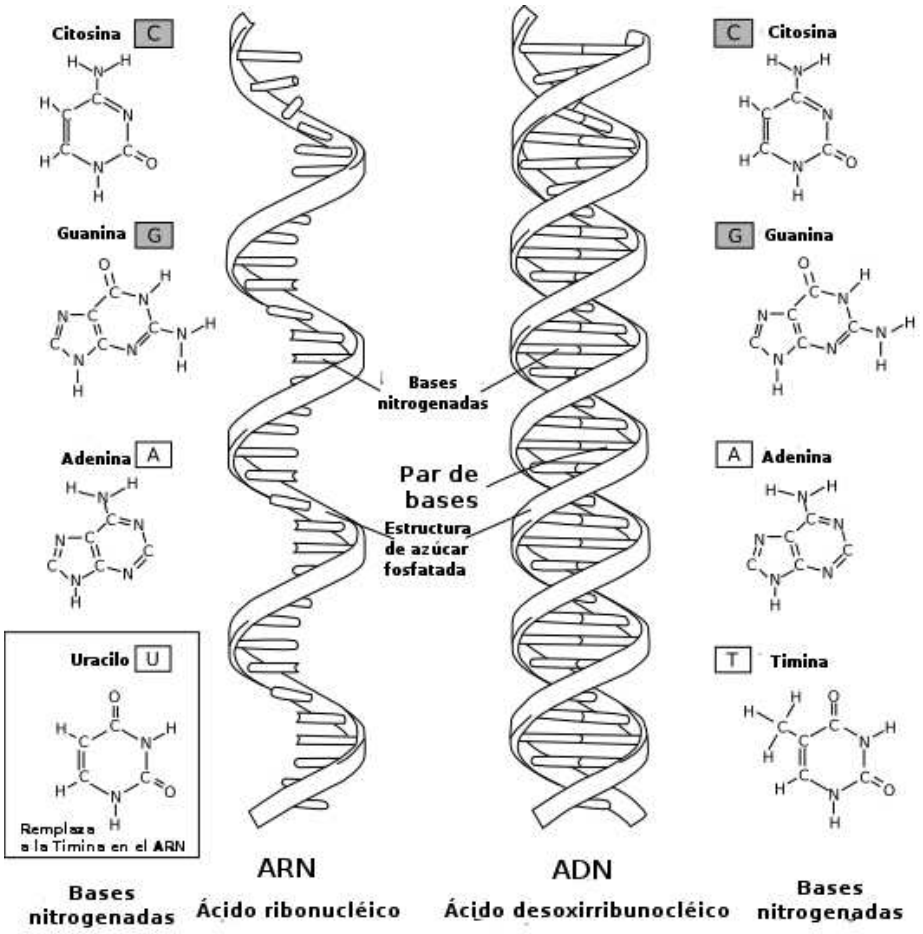

Figura 4.1: *Los ácidos nucleicos.*

tadores esenciales de moléculas de aminoácidos para ser usados en la síntesis de proteínas o para cambiar cuando los genes estén activos.

El ARN es similar estructuralmente al ADN, pero difiere en algunos detalles importantes: El ADN en estado natural forma una doble hélice, mientras que el ARN forma una sola. Los nucleótidos de ARN contienen un tipo de azucar con cinco átomos de carbono llamado ribosa, mientras que los del ADN contienen desoxirribosa (ribosa a la que le falta un átomo de oxígeno). El ARN utiliza en su composición el nucleótido Uracilo (U) en lugar de la Timina (T) presente en el ADN (ver Figura 4.1).

Ribosoma: Organelo citoplasmático compuesto de ARN ribosomal y proteínas ribosomales que cuando se asocian al ARNm catalizan la síntesis de una proteína [3].

Intrón: Derivado del término *región intragénica*, Es una sección no-codificante de ADN. Una vez que la secuencia de ADN se transcribe al precursor del ARNm, los intrones se eliminan de la secuencia. La secuencia de ARNm resultante se traduce entonces en una proteína.

Alelo: En evolución molecular, es una forma alternativa de un gen que se ubica en una posición específica de un cromosoma específico y que difiere de éste en secuencia o en función.

Codón: Unidad funcional del ARN formada por tres nucleótidos de la cadena.

	U	C	A	G
U	UUU Fenilalanina *Phe* UUC UUA Leucina *Leu* UUG	UCU UCC UCA Serina *Ser* UCG	UAU Tirosina *Tyr* UAC UAA Paro UAG	UGU Cisteina *Cys* UGC UGA — Paro UGG — Triptófano *Trp*
C	CUU CUC CUA Leucina *Leu* CUG	CCU CCC CCA Prolina *Pro* CCG	CAU Histidina *His* CAC CAA Glutamina *Gln* CAG	CGU CGC CGA Arginina *Arg* CGG
A	AUU AUC AUA Isoleucina *Ile* AUG — Metionina *Met*	ACU ACC ACA Treonina *Thr* ACG	AAU Asparagina *Asn* AAC AAA Lisina *Lys* AAG	AGU Serina *Ser* AGC AGA Arginina *Arg* AGG
G	GUU GUC GUA Valina *Val* GUG	GCU GCC GCA Alanina *Ala* GCG	GAU Ácido aspártico *Asp* GAC GAA Ácido glutámico *Glu* GAG	GGU GGC GGA Glicina *Gly* GGG

Figura 4.2: *Tabla del código genético.*

Código genético: Es la regla de correspondencia entre la serie de nucleótidos de los ácidos nucleicos y las series de aminoácidos de las proteínas. Durante el proceso de traducción (síntesis de proteínas) el mensaje genético es leído de una cadena de ARNm, colocando cada vez el aminoácido indicado por el codón siguiente según la regla llamada código genético (ver Figura 4.2).

Genotipo: Constitución genética de una célula u organismo [3].

Fenotipo: Expresión del genotipo en un ambiente determinado.

4.1.2. Evolución Molecular

En 1859, Charles Darwin [21] planteó las bases de la teoría que hasta el día de hoy es comúnmente aceptada como la mejor explicación de la evolución de las especies.

Casi diez años después de la controvertida publicación de Darwin, Gregor Johann Mendel [72] realizó una primera descripción de las leyes que rigen la herencia genética, por medio de los trabajos que realizó con diferentes variedades del chícharo. Sin embargo, su trabajo no fue valorado hasta 1900 cuando varios botánicos redescubrieron las leyes que él había planteado más de 30 años atrás.

Durante las siguientes décadas las discusiones entre Mendelismo y Darwinismo se dieron con gran frecuencia hasta que en 1930 surgió la genética de poblaciones como un esfuerzo para proveer una base genética a la teoría de la selección natural de Darwin.

Ronald A. Fisher, John B. S. Haldane y Sewall G. Wrigth se pueden considerar los padres de la genética de poblaciones.

La evolución molecular emergió como un campo científico en 1960 cuando investigadores de áreas como la biología molecular, la biología evolutiva y la genética de poblaciones comenzaron a entender descubrimientos recientes sobre la estructura y función de los ácidos nucleicos y las proteínas.

La evolución molecular está relacionada con los cambios en la información genética que han ocurrido durante la historia de las especies diferenciándolas de sus ancestros. Se puede decir que es el proceso evolutivo a nivel del ADN, ARN, aminoácidos y proteínas.

A nivel molecular, se han identificado cuatro procesos que afectan la sobrevivencia de una característica genética, es decir, la frecuencia de un alelo:

- **Mutación**: También llamada **sustitución** en la teoría neutral. Cambio permanente y transmisible del material genético (ADN o ARN) de una célula. Las mutaciones pueden ser causadas por errores al copiar el material genético durante la división celular, por la exposición a radiación, químicos, virus o pueden ocurrir deliberadamente bajo el control celular durante ciertos procesos biológicos, por ejemplo la meiosis.

- **Deriva génica**: Es la fuerza evolutiva que provoca los cambios en la frecuencia genética que no forman parte de la presión selectiva y que son causados por eventos que no están relacionados con rasgos heredados. La deriva juega un papel importante en la superviviencia de poblaciones pequeñas que simplemente no pueden tener la suficiente descendencia como para mantener la distribución genética heredada de la generación anterior.

- **Migración**: También conocido en evolución molecular como flujo genético. Es el proceso de incorporación y pérdida de individuos en las poblaciones. Es el único de los agentes que mantiene genéticamente cercanas a las poblaciones que se encuentran geográficamente distantes. Permite la construcción de grandes reservas génicas.

- **Selección**: En particular la selección natural producida por la mortalidad diferenciada y la fertilidad diferenciada.

  La mortalidad diferenciada es la proporción de individuos que mueren antes de llegar a edad reproductiva. Si sobreviven, serán entonces seleccionados por la fertilidad diferenciada. Aquellos que logren pasar ambas etapas contribuirán con la totalidad de sus genes a la siguiente generación. De este modo, aumentará la frecuencia de los alelos que estos sobrevivientes contribuyan al acervo genético.

Con el paso del tiempo han surgido varias hipótesis para explicar la evolución molecular. Cada una de estas hipótesis da importancia a uno de los procesos de modificación que se acaban de mencionar. Por ejemplo, la teoría neo-Darwinista plantea que la selección de alelos ventajosos es el eje central de la evolución molecular.

4.1.3. Teoría Neutral

La Teoría Neutral es una de las principales explicaciones de la evolución molecular. Fue introducida por Motoo Kimura a finales de los 60 y principios de los 70. A pesar de que fue recibida por algunos investigadores como un argumento en contra de la Teoría de la selección natural de Darwin, Kimura mantuvo (coincidiendo con la opinión de la mayoría de los biólogos modernos) que ambas teorías son compatibles.

La Teoría Neutral enfatiza la importancia de la mutación y la deriva génica.

De acuerdo con Kimura [57] al comparar entre sí los genes de las especies existentes se observa que la gran mayoría de las diferencias moleculares son selectivamente neutrales. Esto quiere decir que los cambios moleculares representados por estas diferencias no tienen influencia en la aptitud del organismo. Como resultado, la Teoría identifica estas características genéticas como elementos no explicables por la selección natural.

Estos cambios moleculares de influencia mínima se pueden dar a tres diferentes niveles:

- Toda mutación que ocurra en las regiones de ADN conocidas como intrones tendrá un efecto nulo en el fenotipo de la célula o el organismo, por lo que se le puede considerar como una sustitución neutral.

- Todas las proteínas de todos los seres vivos son construidas utilizando un alfabeto de 20 aminoácidos. Las proteínas son construidas por los ribosomas armando secuencialmente aminoácidos codificados en los codones.

  Existe cierto grado de redundancia en el código genético. Hay $4^3 = 64$ posibles combinaciones de 3 nucleótidos para producir sólo 20 aminoácidos. Esta redundancia provoca que muchas posibles substituciones individuales de bases sean *silenciosas* o neutrales. Dichas mutaciones no tienen efecto en la expresión del gen.

  Como muestra la Figura 4.2, normalmente el tercer nucleótido influye poco en el aminoácido producido.

- En algunos casos, cuando la sustitución provoca un cambio en algún aminoácido de la proteína, ésta conserva su estructura, organización y función. A este tipo de mutación se le puede llamar mutación casi neutral y se debe al polimorfismo de la proteína en cuestión.

La Teoría Neutral postula que estas mutaciones neutrales o casi neutrales ocurren con mayor frecuencia que las sustituciones definitivamente ventajosas.

Una segunda afirmación de la Teoría Neutral es que la mayoría de los cambios evolutivos son el resultado de la deriva génica actuando sobre alelos neutrales. Un nuevo alelo surge típicamente a través de mutaciones espontáneas de un solo nucleótido al interior de la secuencia de un gen.

Las sustituciones neutrales crean nuevos alelos neutrales. A través de la deriva estos nuevos alelos se volverán más comunes para la población. Con el paso del tiempo dichos alelos pueden perderse o fijarse[1] en la población. De esta manera las substituciones neutrales tienden a acumularse y los genomas tienden a evolucionar.

Cuando se observan poblaciones divergentes, la mayoría de las substituciones individuales de nucleótidos se acumulan bajo una misma tasa de mutación desde el nacimiento de los individuos. Esta tasa se puede obtener de la tasa de error de las encimas que realizan la replicación del ADN. Dichas encimas han sido estudiadas ampliamente y su comportamiento es parecido para todas las especies. Por ello, la Teoría Neutral es el origen de la técnica del reloj molecular que es usada por los biólogos moleculares para medir el tiempo transcurrido desde que las especies divergieron de un ancestro común.

Además de Kimura, varios biólogos moleculares y genetistas de poblaciones contribuyeron al desarrollo de la Teoría Neutral. Esta teoría es una de las subramas de la síntesis moderna de la evolución.

4.1.4. Trabajos recientes

Huynen et al. [46] plantearon una hipótesis sobre la forma en la que la neutralidad facilita el proceso de adaptación. Para ello definieron el concepto de "red neutral". Se dice que dos secuencias están conectadas entre sí si éstas difieren en una o a lo más dos sustituciones simples. Una *red neutral* se define como un conjunto de genotipos con estructura idéntica, tales que cada secuencia está conectada con al menos otra secuencia.

[1] El que un alelo se fije quiere decir que la mutación se convierte en una característica permanente de la población.

Los autores realizaron un experimento con un modelo computacional que permite predecir la estructura secundaria del ARN. Tomaron una secuencia de ARN y generaron una población de copias idénticas a ésta. Posteriormente, corrieron un proceso de simulación sobre dicha población con una tasa baja de mutación (uno por ciento) como único método de variación de las secuencias y usando la estructura secundaria de cada copia como medida de variación de la población.

Tras transcurrir 500 generaciones se esperaría que la población se hubiera transformado lo suficiente para no existir una estructura secundaria en común para todas las secuencias. Sin embargo, esta suposición difiere del resultado que Huynen et al. obtuvieron al realizar el experimento. Los autores encontraron bloques de la población con la misma estructura secundaria.

El resultado cobra sentido cuando se toma en cuenta la existencia de redes neutrales. Al interior de dichas redes la población se divide en subgrupos. Mientras las secuencias navegan en una red neutral, la población produce una fracción de mutantes fuera de la red. Estos mutantes exploran nuevos genotipos. Generando transiciones entre las diferentes redes neutrales en los puntos en los cuales se acercan unas a otras.

Wagner [107] cuestionó lo que él llama definición esencialista de neutralidad. Mostró la incapacidad de identificar toda posible perturbación de la aptitud tras la ocurrencia de una mutación. Dio varios ejemplos en los que la mutación se podría considerar neutral pero que considerando otros aspectos de la aptitud, tomando en cuenta lo que ocurre tras mutaciones posteriores o en la presencia de cambios en el ambiente se produce una diferencia significativa en el comportamiento molecular.

Tomando estas ideas como premisa, propuso redefinir una *mutación neutral* como aquella que no cambia la función de un aspecto de un sistema biológico en un ambiente y trasfondo genético específicos. Esta nueva definición a su vez permite entender como los cambios neutrales pueden llevar a la innovación cuando se trabaja con sistemas robustos.

Uno de los comentarios a resaltar en el artículo es que para todo sistema biológico altamente robusto, un número dado de mutaciones tendrá una menor cantidad de efectos fenotípicos que los que ocurren en sistemas más sencillos. Por lo tanto, la robustez reduce la cantidad de variación genética heredable sobre la que la selección puede actuar.

4.2. Neutralidad en Algoritmos Evolutivos

Los términos genotipo y fenotipo son frecuentemente usados en el Cómputo Evolutivo pero su definición difiere de la presentada al inicio del capítulo actual.

Al igual que en biología, para el CE el genotipo está constituido por cromosomas. Se utiliza generalmente un único cromosoma por individuo-solución. Por ello, en Cómputo Evolutivo los términos genotipo, cromosoma e individuo normalmente se consideran sinónimos.

Para los AG, el fenotipo es la decodificación del genotipo a valores significativos para el cálculo del valor de aptitud. En PG con representación de árbol no se puede especificar lo que es el fenotipo.

Uno de los principales problemas de los Algoritmos Evolutivos es la convergencia prematura de la población en óptimos locales. Este evento se da cuando una gran parte de la población converge a un mismo genotipo. Esta convergencia impide que el proceso continúe explorando el espacio de búsqueda.

Es convencional considerar a la aptitud como una función del fenotipo. Dicha función se representa matemáticamente mediante [96]:

$$f_Q : Q \longrightarrow R^+ \tag{4.1}$$

donde Q es el espacio de los fenotipos y el dominio real de f_Q es un conjunto finito de R^+.

Sin embargo, la materia prima de todo sistema evolutivo es el genotipo. Esta afirmación es claramente cierta en el Cómputo Evolutivo donde para poder explicar la dinámica del proceso es necesario entender la forma en la que la aptitud se manifiesta a nivel genotípico. Para percibir esta relación se define un mapeo entre genotipo y fenotipo $\phi : G \longrightarrow Q$, donde G es el espacio de genotipos.

La existencia de las dos funciones ϕ y f_Q permite definir en G una función de aptitud $f_G = f_Q \circ \phi$ [96].

Cuando el mapeo genotipo-fenotipo es redundante (muchos a uno) la función f_G se considera degenerada. Dicha redundancia radica en la existencia de genotipos diferentes entre sí pero con el mismo valor de aptitud. La Figura 4.3 representa un mapeo genotipo-fenotipo redundante.

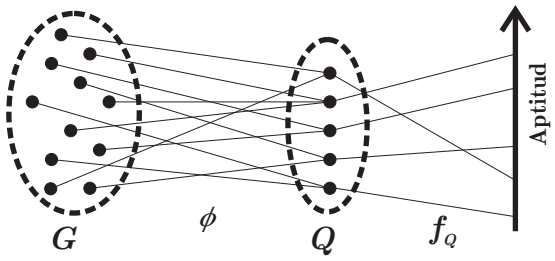

Figura 4.3: *Mapeo genotipo-fenotipo redundante.*

Supóngase el caso de un mapeo genotipo-fenotipo redundante. Bajo este supuesto existe la posibilidad de que los operadores de cruza y/o mutación produzcan descendencia con el mismo fenotipo que el de los padres. Si esto ocurre, se dice que los operadores tienen un efecto **neutral** en la población. En este contexto el término neutral se utiliza como analogía a la mutación neutral definida por Kimura.

En Cómputo Evolutivo *red neutral* se define como un conjunto de genotipos conectados unos a otros por una aplicación del operador de mutación o una aplicación del operador de cruza que mapean en el mismo fenotipo.

Varios autores ([91, 69, 106, 26, 90, 75, 47], entre otros) identifican los mapeos genotipo-fenotipo redundantes, los operadores neutrales, las redes neutrales y el efecto que todos estos elementos tienen en los Algoritmos Evolutivos con el nombre de **neutralidad**.

Al inicio de esta sección se comentó que para la PG con representación de árbol el fenotipo no está definido. Por esta situación la neutralidad para dicho modelo se identifica como redundancia en f_G. La Figura 4.4 representa esta relación.

Por la misma razón, en Programación Genética con representación de árbol el concepto de mapeo genotipo-fenotipo redundante pierde significado. Cuando se habla de neutralidad para dicha representación se hace referencia a individuos con diferente genotipo pero con el mismo valor de aptitud; así como a los operadores neutrales, las redes neutrales y el efecto que todos estos elementos tienen en la PG. Es en este sentido que se utilizará el término neutralidad en el caso de estudio de la presente tesis.

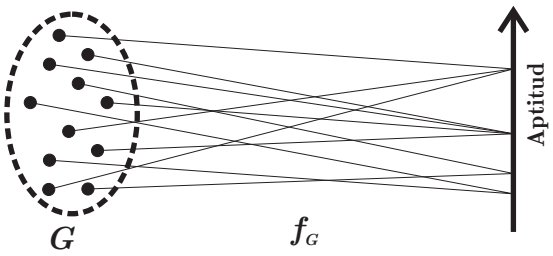

Figura 4.4: *Neutralidad en Programación Genética.*

En algunos casos se ha encontrado que la neutralidad es benéfica para los AE porque la convergencia prematura ocurre con menor frecuencia. La explicación de dicho beneficio se encuentra en que aunque toda la población pudiera llegar a tener un mismo valor de aptitud no necesariamente todos los individuos comparten el mismo genotipo. Esta característica provoca que al transcurrir una cierta cantidad de generaciones los operadores de cruza y mutación le permitan a los individuos moverse a través de la red neutral en la que se encuentran y presumiblemente salir de ésta en algún momento.

Los primeros en identificar que las secciones del genotipo que no modifican el valor de aptitud pueden llegar a ser útiles bajo alguna otra configuración de los elementos que si lo modifican fueron Harvey y Thompson. En [44] mostraron dichas características para un problema de *hardware* evolutivo.

Existen muchas diferencias entre la evolución biológica y el proceso evolutivo modelado en los AE. En los Algoritmos Evolutivos la aptitud se puede calcular fácilmente aplicando la función 4.1. Por el contrario, en biología molecular, tal como lo plantea Wagner [107], la aptitud es difícil de definir y es casi imposible de medir de manera rigurosa.

A causa de esta diferencia los problemas identificados por Wagner para definir aptitud no pueden reflejarse directamente en el Cómputo Evolutivo. Sin embargo, hay un concepto que se puede extraer de su artículo y que es importante tener en cuenta. La neutralidad debe ser motivo de estudio siempre que tenga un efecto fenotípico a largo plazo en la población. El hecho de agregar genes que en ningún momento provoquen un cambio a nivel fenotípico no propiciará ninguna mejora directa en el proceso evolutivo.

Existen AE que implícitamente permiten el mapeo de múltiples genotipos a un mismo fenotipo. Modelos como la Programación Genética y la Evolución Gramatical tienen un cierto grado de "neutralidad implícita" (dicho término se tratará en extenso en la sección 4.3.1). Sin embargo, valiéndose de modificaciones a la representación o al modelo del AE es posible agregar o controlar la neutralidad del proceso evolutivo. A este tipo de redundancia agregada se le llama neutralidad explícita.

El problema es ¿Cómo lograr que múltiples genotipos codifiquen hacia un mismo fenotipo sin caer en la adición de genes que no propicien alguna mejora?, es decir ¿Cómo incorporar neutralidad que le permita al Algoritmo Evolutivo salir con mayor facilidad de óptimos locales?

Con base en la lectura de artículos publicados sobre el tema se propone la siguiente clasificación de las estrategias diseñadas para permitir la neutralidad en los Algoritmos Evolutivos:

- Agregar elementos a los cromosomas.
- Modificar el modelo del Algoritmo Evolutivo.
- Modificar características del paisaje de aptitud.

A continuación se explicará en qué consisten estas clasificaciones, se hablará sobre las ventajas y desventajas que tienen y se darán ejemplos de su funcionamiento.

4.2.1. Modificación a los cromosomas

La forma más sencilla y directa de agregar neutralidad al proceso de búsqueda evolutiva es adicionar un elemento externo que no codifique directamente en el cromosoma. Se debe idear alguna manera para que dicho elemento tenga un efecto fenotípico a mediano o largo plazo.

Este método es sencillo de implementar y al funcionar sobre Algoritmos Genéticos permite analizar formalmente el comportamiento del modelo evolutivo en la presencia de la neutralidad. Sin embargo la neutralidad agregada es relativamente pobre, ya que permite la existencia de una limitada cantidad de redes neutrales y dichas redes no guardan una relación con el valor de aptitud de los individuos.

Galván y Poli [35] propusieron un claro ejemplo de modificación simple a los cromosomas. Consiste en agregar un bit al cromosoma típico de un AG cuyo único operador de variación es la mutación. Si el bit es uno el valor de aptitud del individuo se fija a un número constante durante la corrida. Si el bit es cero el individuo se evalúa normalmente.

Probaron este método con dos paisajes de aptitud diferentes y concluyeron que no siempre es benéfico para el proceso evolutivo y que es difícil caracterizar el comportamiento del modelo evolutivo al agregarle neutralidad.

4.2.2. Algoritmos Evolutivos neutrales

En Algoritmos Evolutivos neutrales se agrupan modificaciones a diferentes niveles de los AE. Esta categoría incluye operadores especialmente diseñados para aprovechar la neutralidad en la mejora del desempeño de un Algoritmo Evolutivo específico, mejoras de algoritmos típicamente no considerados evolutivos logradas gracias a la inclusión de neutralidad, AE específicamente diseñados para paisajes de aptitud altamente neutrales y modelos evolutivos completamente innovadores que utilizan elaborados mapeos genotipo-fenotipo.

Es difícil hablar de ventajas y desventajas de esta categoría ya que la innovación puede darse a diversos niveles del modelo evolutivo. A continuación se presentan algunos trabajos que, tomando el concepto de neutralidad, han presentado alguna modificación ventajosa a los Algoritmos Evolutivos:

Stewart [99] propuso un nuevo método de selección. Dicho método toma en cuenta la existencia de redes neutrales. Cuando una parte de la población se encuentra en una de estas redes, se le da preferencia a los individuos que se encuentren más lejanos al centroide. Concluyó que dicho operador puede ser benéfico en paisajes de aptitud con un alto grado de neutralidad.

Barnett [9] presentó una variación a la búsqueda mediante Escalador de Colinas que incorpora la idea de moverse a través de redes neutrales. Las modificaciones hechas al algoritmo incluyen realizar la sustitución del hijo si tiene la misma aptitud que su predecesor, mutaciones basadas en distribuciones de probabilidad y una tasa de mutación variable (adaptable). Sin embargo no comparó los resultados obtenidos por su método contra los obtenidos por algún otro.

Shipman et al. [90] exploraron los beneficios de la neutralidad existente en las redes booleanas aleatorias. Dichas redes son una intrincada representación en la que los genes son activados y desactivados mediante la consulta de tablas de verdad, usando otros genes como parámetros. A partir de resultados experimentales concluyeron que el flujo neutral permite el descubrimiento de más fenotipos que cuando se ocupa una codificación directa. Ebner et al. [27] extendieron este trabajo agregando diferentes tipos de mapeos. Concluyeron que los mapeos indirectos aumentan la movilidad sobre el espacio de búsqueda en comparación con los mapeos uno a uno.

4.2.3. Características redundantes del paisaje de aptitud

Existen paisajes de aptitud que intrínsecamente mapean múltiples genotipos en un solo fenotipo. Un claro ejemplo son todos los paisajes de aptitud a los que la Programación Genética se enfrenta (ver sección 4.3). El aprovechar esta neutralidad implícita surge como una posibilidad.

Downing presentó en [26] un AE especialmente diseñado para modificar Diagramas Binarios de Decisión (*Binary Decision Diagrams* o DBD) [2] aprovechando que múltiples DBD pueden tener el mismo resultado. Para ello definió varios operadores de mutación específicos para la representación usada (cuatro neutrales y uno adaptativo). Al aplicar el algoritmo y tener identificado el operador ocupado, si éste se trata de una mutación neutral ahorra tiempo en la evaluación.

Además de ahorrar tiempo de ejecución, gracias a los movimientos a través de redes neutrales (caminatas neutrales) presenta una mejora en el desempeño del algoritmo para varios problemas de paridad y el diseño de multiplexores. Downing retomó el tema en [25] para realizar un análisis del comportamiento del algoritmo conforme se modifica la probabilidad de mutación y se da preferencia a las mutaciones neutrales. Observó un mejor comportamiento conforme la tasa de mutación decrece y las mutaciones neutrales ocurren con una mayor frecuencia que la adaptativa.

Otro posible planteamiento es, dado un paisaje de aptitud no neutral, generar un paisaje de aptitud muy parecido al original pero que incluya neutralidad. Existe la probabilidad de que al realizar esta modificación se disminuya la dificultad para que un AE encuentre una solución.

Por ejemplo, Barnett [8] introdujo un parámetro p al paisaje de aptitud NK, llamándolo NKp. La idea es que por medio de este parámetro se pueda modificar la cantidad de neutralidad del problema. Tras experimentar observó que la neutralidad le permitía a un AG salir de óptimos locales.

4.2.4. Métricas de la neutralidad

Las definiciones que se han dado de neutralidad en el presente documento son cualitativas más que cuantitativas y sólo se ha explicado de manera intuitiva la forma en la que las redes neutrales permiten que las poblaciones salgan de óptimos locales. Sin embargo recientemente se han desarrollado investigaciones que analizan de manera más formal y cuantitativa los fenómenos relacionados con neutralidad en Cómputo Evolutivo. A continuación se resumirán algunos de estos artículos.

Igel y Toussaint [47] mostraron formalmente la importancia de la auto-adaptación de los Algoritmos Evolutivos, cuyo efecto fenotípico es neutral. Aclarado esto incluyeron en la medida del número de evaluaciones promedio un factor de neutralidad necesitado por un AE con auto-adaptación para encontrar una solución satisfactoria.

Fonseca y Correia [31] propusieron un modelo evolutivo simple (mutación-selección) al que le agregan redundancia y pleitropía[2]. Concluyeron que la redundancia no necesariamente es perjudicial para su modelo.

Galván y Poli [69] presentaron un análisis matemático del método para adicionar neutralidad detallado en [35] valiéndose de la correlación de distancia de aptitud (*Fitness Distance Correlation* [52]). Concluyeron que la neutralidad puede ser benéfica en algunos casos. Sobre todo cuando la mayor parte de la población se inicializa en una parte mala del espacio de búsqueda, aclarando que esto ocurre frecuentemente cuando se trabaja con espacios de búsqueda muy grandes y tamaños de población pequeños.

Vanneschi et al. [106] propusieron varias métricas para el comportamiento de las redes neutrales. Todas estas métricas implican muestreos extensivos o cuasi-extensivos de los espacios de búsqueda a analizar y permiten analizar a detalle el comportamiento y evolución de las redes neutrales. Los resultados obtenidos sobre un mismo paisaje de aptitud de la PG con diferentes conjuntos de funciones explican claramente la diferencia del comportamiento de dicho modelo evolutivo sobre el problema.

Cuando se analiza el flujo de la evolución de poblaciones (tanto en biología como en computación), la aptitud normalmente se asocia con el éxito reproductivo de los individuos. Si en el transcurso del proceso evolutivo únicamente se toma en cuenta la selección reproductiva para el cálculo de aptitud (aptitud reproductiva) de los individuos, los mapeos genotipo-fenotipo redundantes no presentarían ninguna ventaja sobre aquellos que no lo son.

Tal como se maneja en el presente capítulo, existen ocasiones en los que la neutralidad aporta ventajas al proceso evolutivo. Dichas situaciones llevan a la conclusión de que la noción tradicional de aptitud, a la que llamaremos aptitud reproductiva, no considera todos los aspectos involucrados en el proceso evolutivo.

La aptitud efectiva es una métrica que toma en cuenta el efecto que tienen las cruzas y mutaciones destructivas en los individuos y explica el fenómeno conocido como "hinchamiento" (*bloat*, ver sección 4.3.1) y por qué en ocasiones la neutralidad presenta mejoras en el proceso evolutivo.

Nordin y Banzhaf [77, 78] plantearon una forma de calcular un límite inferior para la aptitud efectiva para la PG estándar.

De manera independiente Stephens y Waelbroeck [98] plantearon que la aptitud efectiva se puede calcular agregando a la fórmula de la aptitud reproductiva el efecto que los operadores de mutación y cruza tienen en la población. Los autores trabajaron sobre ecuaciones de evolución de esquemas para el AGS. Stephens [93, 94] desarrolló con mayor formalidad dichos planteamientos. De esta serie de artículos surgió el Teorema exacto del esquema (*Exact Schema Theorem*).

[2]Un gen puede afectar simultáneamente varios rasgos fenotípicos

Poli [80] valiéndose del trabajo realizado en [98] planteó la versión del Teorema exacto del esquema para la PG estándar y una fórmula para encontrar el valor exacto de la aptitud efectiva en dicho modelo evolutivo.

Finalmente Stephens y Mora [96, 97] plantearon una definición de aptitud efectiva que se puede ocupar para cualquier AE y analizaron formalmente dicho concepto para diferentes paisajes de aptitud de los AG.

4.3. Neutralidad en Programación Genética

Los primeros trabajos relacionados con Cómputo Evolutivo en los que se comenzó a utilizar el término neutralidad se presentaron para el método conocido como Programación Genética [6, 55, 44]. Esto no es de sorprender ya que como se presentará a continuación uno de los principales problemas que aqueja a la PG es el crecimiento incontrolable de código inactivo. Dicho crecimiento claramente provoca mapeos de múltiples genotipos a un solo fenotipo.

A continuación se explicará con mayor detalle este proceso y se abordarán las formas en las que se ha agregado y manipulado la neutralidad en la PG.

4.3.1. Intrones, hinchamiento y neutralidad implícita

Es normal que en el transcurso del proceso evolutivo de la Programación Genética el tamaño de los árboles crezca de manera incontrolable hasta alcanzar la profundidad máxima permitida. Este fenómeno es conocido como hinchamiento (*bloat*). Cabe aclarar que la mayor parte de los segmentos de código agregados por hinchamiento tienen efecto nulo en la aptitud de los individuos. A modo de analogía con los fragmentos no-codificantes de ADN, a dichos segmentos de código se les conoce como intrones.

Los intrones son segmentos de código emergentes que no afectan directamente el valor de aptitud de los individuos. Esto significa que cualquier código que emerja en el transcurso de la ejecución de la PG y que no afecte directamente la aptitud de los individuos es considerado un **intrón** [7]. Algunos ejemplos típicos son:

- $(\neg(\neg\ \mathrm{X}))$
- $(\ldots \mathrm{AND}(\mathrm{X\ OR\ X}))$
- $(\cdots + (x - x))$
- $(x * 1)$
- $(\cdots * (x/x))$

- (*rigth left*)
- (if$(2 = 1) \ldots x$)
- $(A := A)$

Se ha reportado que en ocasiones durante las generaciones iniciales e intermedias los intrones llegan a ocupar entre el 40 y el 60 % de todo el código de la población y que en las últimas generaciones estos tienden a crecer exponencialmente y ocupar la mayor parte de éste [7, pp. 182].

Los intrones son benéficos durante las primeras etapas del proceso evolutivo por dos razones:

- Sirven como *reserva génica* mediante la cual se pueden ir generando bloques constructores de soluciones potenciales por medio de mutaciones y cruzas benéficas.
- Brindan una protección global a los individuos en contra de mutaciones y cruzas destructivas. Normalmente, el operador de cruza es destructivo en mas del 75 % de sus aplicaciones [92]. Una forma en la que un individuo puede proteger sus genes de dicho operador es mediante la inserción de intrones. Dichos intrones reducirán la probabilidad de que la cruza separe la buena solución que el individuo pueda contener.

Sin embargo, conforme la PG se acerca a sus últimas generaciones las ventajas de contener intrones se vuelven en contra de ésta. Esto se debe a que el proceso habrá encontrado valores de aptitud difíciles de mejorar. La gran proporción de intrones con respecto a código activo de los individuos provocarán que los operadores tengan un efecto casi completamente neutral en su funcionamiento.

Por acarrear problemas como el estancamiento del proceso de búsqueda de soluciones, saturación de memoria y aumento en el tiempo de ejecución (sobre todo al momento de la evaluación de los individuos), es en este punto cuando el hinchamiento se vuelve un problema grave para la PG. Una de las soluciones más comunes es dividir el proceso evolutivo en dos etapas:

- **Exploración del espacio de búsqueda**: Se desea encontrar genotipos con un buen valor de aptitud no importando su tamaño.
- **Reducción del tamaño de las soluciones**: Una vez que una parte de la población (de preferencia la mayoría de ésta) se encuentra en el espacio de soluciones satisfacientes se buscará reducir el tamaño de los árboles. Dicha reducción se consigue mediante la eliminación de los intrones o la penalización en el número de nodos de la función de aptitud.

De este modo, la Programación Genética aprovecha los beneficios de los intrones y escapa al problema del hinchamiento. Sin embargo, dicha división del proceso de búsqueda provoca un problema cuando se trabaja sobre paisajes de aptitud de los que no se tiene información previa ¿Cuál es el momento indicado para cambiar de una fase a la otra? Desafortunadamente no hay una respuesta precisa a este cuestionamiento.

Los intrones son elementos que agregan neutralidad a la PG ya que son una de las causas de que, no importando las modificaciones realizadas en la herramienta, existan diferentes genotipos con un mismo valor de aptitud. Sin embargo, la nula influencia directa que tienen sobre el valor de aptitud y la imposibilidad para aumentarlos, medirlos o controlarlos los convierte en neutralidad de la que normalmente se puede sacar poco provecho, llamada **neutralidad implícita**.

4.3.2. Variantes neutrales de la PG

En los últimos diez años han surgido una gran cantidad de métodos que modifican el proceso de la Programación Genética y que buscan aprovechar la neutralidad de los paisajes de aptitud de dicho modelo. Algunos incluso se parecen muy poco al modelo evolutivo propuesto por John Koza. Sin embargo, todos estos métodos sirven para resolver el mismo tipo de problemas y por eso son considerados en esta tesis como variantes neutrales de la Programación Genética.

Los siguientes trabajos son algunos ejemplos:

El Algoritmo Evolutivo conocido como Evolución Gramatical [79] propuesto por O'Neill y Ryan (ver sección 2.4) tiene cierto grado de neutralidad ya que puede suceder que algunos valores de la secuencia entera no se ocupen en la generación de las gramáticas.

Stephens et al. [95] analizaron el surgimiento de robustez evolutiva en un complejo mapeo genotipo-fenotipo redundante como motor de búsqueda de la EG para un problema simple de regresión simbólica ($f(x) = 4x$). Con un análisis detallado de una corrida identificaron que el proceso evolutivo opta por soluciones óptimas que contienen símbolos que no afectan el valor de aptitud. La explicación de este proceso es que dicho mecanismo protege a los individuos de la acción destructiva de los operadores reproductivos, siendo equivalente al hinchamiento en PG.

La Programación Genética Binaria (*Binary Genetic Programming* o PGB) propuesta por Banzhaf [6] funciona de una manera parecida al algoritmo propuesto por Ryan y O'neill. Sin embargo, elimina el componente de las gramáticas y utiliza una asignación redundante de los operadores, imitando la redundancia existente en el código genético.

En [6] Banzhaf presentó los resultados al utilizar PGB para resolver algunos problemas de regresión simbólica y concluyó que la separación de las operaciones de búsqueda y el uso de los mapeos genotipo-fenotipo otorga mayor flexibilidad al proceso evolutivo que el trabajar únicamente sobre fenotipos. Keller y Banzhaf [55] extendieron el trabajo obteniendo nuevamente resultados favorables.

Otra variante neutral de la Programación Genética es presentada por Miller y Thomson en [75]. Esta herramienta es llamada Programación Genética Cartesiana (*Cartesian Genetic Programming* o PGC). Lo innovador en este método es la representación de los elementos. Un Programa Cartesiano es un conjunto fijo de enteros que representan las conexiones entre una serie de nodos y las entradas y las salidas del problema, así como también el índice de la función que se aplicará en cada nodo. Es probable que algunos nodos queden completamente desconectados de las salidas. Dicha desconexión posibilita la existencia de neutralidad en la PGC.

La mutación de la cadena de enteros permite modificar las conexiones de los elementos (entradas, salidas y nodos) y las funciones utilizadas en los nodos. Si se aplican estos operadores a un conjunto de Programas Cartesianos por un cierto número de generaciones es probable que se obtenga un programa que aproxime adecuadamente funciones matemáticas, booleanas, etc. Tras aplicar este método a diversos paisajes de aptitud los autores concluyeron que PGC no es menos efectiva que otras formas de PG en la solución de problemas de regresión simbólica, *hardware* evolutivo e Inteligencia Artificial. Miller y Thomson comentaron que la neutralidad ayuda a que se obtengan estos resultados.

Yu y Miller han reportado el uso de PGC en diferentes paisajes de aptitud [109, 110, 111]. Concluyeron que la neutralidad mejora la probabilidad de éxito del proceso evolutivo. Sugirieron que dicha mejora sucede porque la mutación es destructiva conforme la población se vuelve apta y la neutralidad provee espacio de protección contra ésta.

Sin embargo, Collins [19] analizó mediante un modelo matemático los resultados de [110]. Concluyó que, para dicho problema, la afirmación de que la neutralidad es benéfica es falsa.

4.3.3. Modificar el conjunto de funciones

Una forma de incrementar la neutralidad en la Programación Genética sin tener que modificar todo el modelo, es agregar elementos al conjunto de funciones que incrementen la cantidad de código inactivo y que, tras la ocurrencia de algún operador genético, tengan algún efecto fenotípico.

Ejemplo de ello es el trabajo presentado por Galván, Rodríguez y Poli [37]. En dicho artículo, los autores analizaron los efectos de la adición de un elemento extra en el conjunto de funciones de la PG para permitir neutralidad explícita en varios problemas de *hardware* evolutivo. Este elemento extra funciona como una liga a otra sección del árbol (a pesar de que dicha función tiene dos subárboles que son inactivos). Tras una serie de pruebas para problemas de *hardware* evolutivo concluyeron que la neutralidad explícita tiene un mejor desempeño general en términos de consistencia en el alcance de soluciones factibles.

Posteriormente, Galván y Rodríguez [36] depuraron el método y obtuvieron mejores resultados sobre paisajes de aptitud parecidos.

Capítulo 5

Caso de estudio

El problema seleccionado como caso de estudio fue planteado por Jefferson et al. [50] como un modelo de Vida Artificial (*Artificial Life*). En el artículo original se presentaron soluciones mediante AG y redes neuronales artificiales. Sin embargo, ambas requerían una cantidad considerable de tiempo de cómputo para encontrar una solución óptima.

Durante los siguientes años, varios investigadores propusieron soluciones utilizando diferentes modelos computacionales. Por ello el problema de la Hormiga Artificial se convirtió en un caso de prueba tradicional para las personas que trabajan en el área de aprendizaje automático y sobre todo para aquellos que trabajan con la Programación Genética.

En este capítulo se hace una descripción del problema de la Hormiga Artificial y un breve resumen de los diferentes métodos que se han utilizado para resolverlo. Se pone énfasis especial en las diversas modificaciones hechas a la Programación Genética para mejorar su desempeño en este caso de prueba.

5.1. Descripción del problema

El objetivo del problema de la Hormiga Artificial es encontrar un programa que navegue satisfactoriamente un agente (u Hormiga Artificial) a través de un rastro discontinuo y con muchos quiebres de 89 piezas de comida en una matriz toroidal de 32×32 casillas. El agente debe recorrer una sola configuración del rastro de comida. Es decir, el agente sólo debe resolver un trazo a definir.

La Figura 5.1 muestra el trazo Santa Fe. Cabe aclarar que las casillas de color gris claro no son parte de éste; simplemente sirven para visualizar la ruta más rápida para atravesarlo. La Hormiga Artificial se encuentra en la posición superior izquierda mirando hacia el este.

Además del trazo Santa Fe existe el trazo John Muir. Este último fue utilizado en [50]. La primera aparición del trazo Santa Fe fue en el artículo de Koza [60]. En dicho artículo se menciona que fue Christopher Langton quien lo propuso como un reto para los algoritmos de

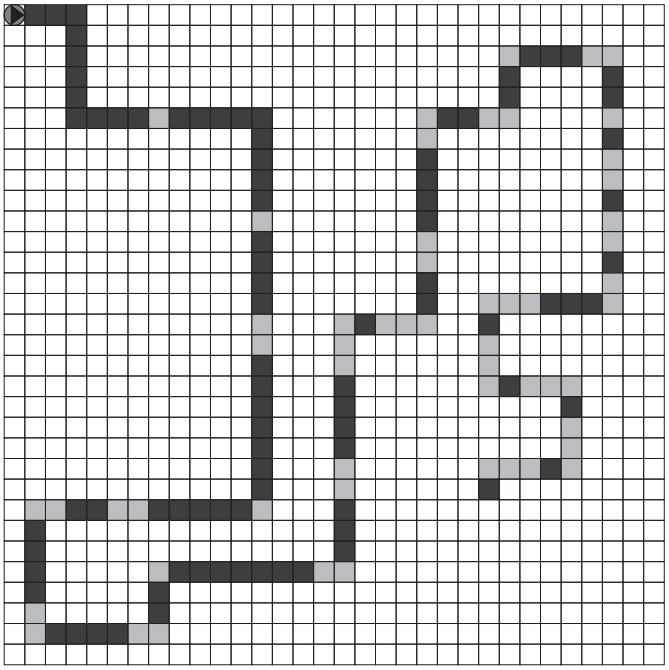

Figura 5.1: *El trazo Santa Fe.*

búsqueda. El trazo John Muir cayó rápidamente en desuso por ser más sencillo de resolver que su sucesor.

El trazo Santa Fe se ha utilizado normalmente cuando se pibúsquedaensa en el problema de la Hormiga Artificial como caso de prueba. Esto se debe al alto grado de dificultad que tiene. En la Figura 5.1 puede verse claramente cómo la dificultad va aumentando gradualmente conforme se avanza en el trazo hasta llegar al punto en el que se tienen *gaps*[1] dobles y triples separando casillas individuales de comida.

Es por esta razón que la presente tesis sólo toma en cuenta el trazo Santa Fe como caso de prueba para la generación de experimentos.

El agente cuenta con un conjunto reducido de posibles acciones para solucionar el trazo Santa Fe:

- Moverse una casilla adelante (y si la nueva casilla tiene comida comérsela).
- Girar a la derecha.
- Girar a la izquierda.

[1]En este contexto, *gap* se define como una casilla o una serie de casillas sin rastro que el agente debe atravesar para poder llegar a la siguiente posición de comida.

Cada una de estas operaciones consume una unidad de tiempo. También puede mirar la casilla que tiene enfrente para determinar si ésta tiene o no comida.

Cuando se aborda este problema mediante PG el conjunto de terminales esta compuesto por {*move, right, left*}. Cada uno de sus elementos corresponde a una de las acciones básicas que se describieron anteriormente. El conjunto de funciones está compuesto por:

- La función perceptiva *IfFoodAhead* toma dos argumentos. Ejecuta el primer argumento si la siguiente casilla contiene comida. Ejecuta el segundo argumento si la siguiente casilla no contiene comida.
- *Progn2* toma dos argumentos y los ejecuta de manera secuencial.
- *Progn3* toma tres argumentos y los ejecuta de manera secuencial.

El agente cuenta con 600 unidades de tiempo para la ejecución de un programa de control. Si el recorrido del árbol termina antes de haber transcurrido dicha cantidad de tiempo, el programa se volverá a correr desde el principio. La ejecución se repetirá mientras el límite de tiempo no se alcance.

La Tabla 5.1 resume las características del problema a resolver.

Tabla 5.1: *Características del problema de la Hormiga Artificial para el trazo Santa Fe.*

Objetivo	Encontrar un programa que controle a una Hormiga Artificial de manera que ésta pueda encontrar todas las piezas de comida del trazo.
Conjunto de terminales	{*move, left, right*}
Conjunto de funciones	{*IfFoodAhead, Progn2, Progn3*}
Función de aptitud	Número de piezas de comida consumidas.
Función estandarizada de aptitud	Piezas de comida remanentes tras la ejecución del programa.
Caso de prueba	600 unidades de tiempo para recorrer una configuración específica de 89 casillas de comida (trazo Santa Fe).

5.2. Corridas de ejemplo

A continuación se presenta la evaluación de dos programas de control para aclarar cualquier duda sobre el problema en cuestión.

Se utiliza una representación gráfica compacta de los terminales y las funciones para facilitar la lectura e interpretación de los esquemas que representan a los programas de control. Las equivalencias entre una representación y la otra son las siguientes:[2]

[2]La idea de la representación compacta se tomó de [65].

- Las hojas se representan con una sola letra. La **m** equivale al terminal *move* o mover. La **l** representa un terminal *left* o girar a la izquierda. La **r** equivale al terminal *right* o girar a la derecha.
- La función *IfFoodAhead* se representa mediante la palabra **If**. Los conectores secuenciales (*Progn2* y *Progn3*) se representan mediante un círculo negro y se pueden diferenciar el uno del otro por la cantidad de hijos.

5.2.1. Programa de control básico

La Figura 5.2 muestra un programa de control generado con el algoritmo usado para crear la población inicial. Como es de esperarse, dicho programa no representa un individuo que resuelva el problema. Sin embargo, el árbol contiene todos los elementos de los conjuntos de terminales y funciones por lo que es un programa que sirve bien de ejemplo.

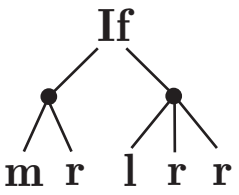

Figura 5.2: *Un programa de control generado de manera aleatoria.*

Supóngase que se parte de la configuración inicial del trazo Santa Fe (agente en la esquina superior izquierda mirando hacia el este).

Lo primero que hace el agente es verificar si hay comida en la siguiente posición. Dado que está mirando hacia el este la respuesta a este cuestionamiento es afirmativa por lo que se ejecuta el subárbol del lado izquierdo del *IfFoodAhead*, entrando así al bloque *Progn2*. Esto provocará que se llame a *move*. Con ello la hormiga gastará su primera unidad de tiempo en moverse una casilla (consumiendo así la primera posición de comida del trazo). Tras esto se ejecuta el argumento derecho del *Progn2*, es decir *right*, girando el agente hacia la derecha. Las unidades de tiempo consumidas se aumentan a dos y el primer ciclo sobre el programa de control ha terminado.

Se realiza una segunda invocación pues la cantidad de unidades de tiempo todavía no alcanza el límite (600). Se inicia nuevamente con la función perceptiva *IfFoodAhead*. Dado que, en este caso la hormiga se encuentra mirando hacia el sur, la respuesta es negativa. Por esto se invoca el subárbol derecho de la función, el cual es un *Progn3*. Esto provocará que el agente gire hacia la izquierda mediante el terminal *left*, volteando nuevamente hacia el este. Sin embargo, después de esto se llama dos veces consecutivas al terminal *right*. Provocando con estas acciones que al terminar el segundo ciclo de ejecución la hormiga quede en la misma posición que al inicio de éste pero mirando hacia el oeste y con una cantidad de cinco unidades de tiempo consumidas.

Es claro que este procedimiento se repetirá dos veces más hasta que el agente quede mirando nuevamente hacia el este, llevando así un total de 11 unidades de tiempo consumidas. Esta secuencia de giros le permite a la hormiga inspeccionar todas las casillas circundantes en busca de trazo. Sin embargo, no es la mejor manera de hacerlo ya que consume tres unidades de tiempo en una actividad que podría costarle únicamente una.

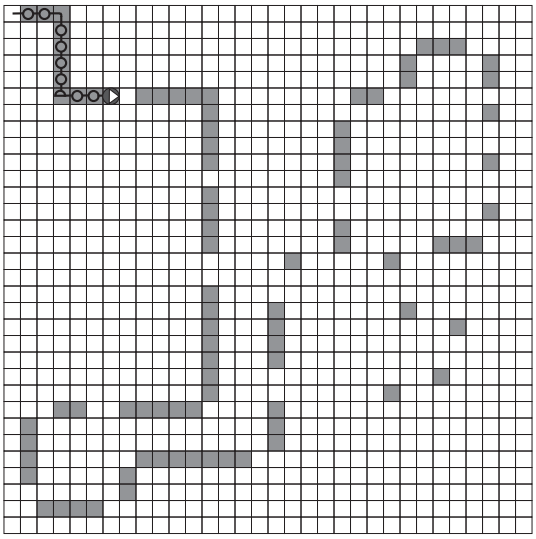

Figura 5.3: *Recorrido seguido usando el programa de control básico.*

Esta secuencia le permite avanzar una casilla a la hormiga tras haber girado a todas las direcciones. Dicha secuencia se realiza dos veces más comiendo las segunda y tercera posiciones del rastro y llevando 24 unidades de tiempo consumidas. La forma en la que funciona el programa tras el primer giro del trazo es un poco diferente ya que el agente queda viendo hacia el sur y la siguiente casilla está en esa misma dirección. La función *IfFoodAhead* detectará que la siguiente casilla tiene comida llamando con ello la rama izquierda del árbol y provocando así que la hormiga avance y se coma la comida sin necesitar girar.

Sin embargo, de ahí en adelante volverá a ser necesario que el agente ejecute tres veces el programa para girar en todas las direcciones (norte, este y sur) antes de avanzar y consumir la siguiente posición de comida del rastro. La secuencia girar y avanzar se repetirá cuatro veces llevando a la hormiga al siguiente codo del trazo.

En este codo solo será necesario que el agente gire la mitad de veces para toparse con la siguiente posición de comida. Nuevamente la consumirá y volverá a necesitar del proceso de gira y avanza para consumir las últimas dos casillas que este programa de control le permitirá alcanzar. Para llegar a este punto el agente ha invertido 100 unidades de tiempo.

Con esto la hormiga llega al primer *gap* del trazo y dado que la única manera en la que el agente avanza es teniendo una posición de comida frente a él y a estas alturas no hay ninguna directamente accesible, el programa de control lo hará rotar hacia la derecha sin mover su posición para consumir las unidades de tiempo restantes.

La Figura 5.3 muestra el recorrido seguido por la hormiga artificial bajo el control del programa representado por el árbol de la Figura 5.2. Puede verse que se consumieron 11 posiciones de comida por lo que el valor de aptitud estandarizado (posiciones de comida restantes del trazo) del programa de control es 78.

5.2.2. Programa de control óptimo

La Figura 5.4 muestra un programa de control óptimo que fue encontrado por una corrida de la Programación Genética con los parámetros descritos en el apéndice A. Dicho programa óptimo no es el único ni el de menor tamaño.

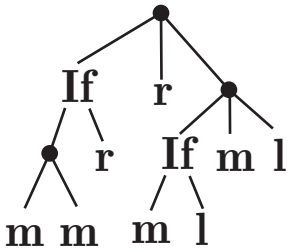

Figura 5.4: *Un programa de control óptimo.*

Puede verse claramente que es un programa más elaborado que el ejemplo anterior y que no tendrá el problema de quedarse estancado si no tiene posiciones de comida inmediatas ya que cuenta con al menos un terminal *move* afuera de todo lado derecho de cualquier *IfFoodAhead*.

Supóngase nuevamente que se parte de la configuración inicial del trazo Santa Fe (agente en la esquina superior izquierda mirando hacia el este).

En la primera ejecución del programa de control, la hormiga localiza que tiene una posición de comida enfrente de ella y entra al subárbol izquierdo del *IfFoodAhead* izquierdo. Con ello alcanza las primeras dos posiciones de comida. Tras esto gira a la derecha y vuelve a buscar comida, dado que no la encuentra gira a la izquierda (volviendo a mirar al este) y avanza consumiendo la tercera posición de comida del rastro. Es importante observar que este programa de control alcanzó tres posiciones de comida en una sola ejecución y utilizando únicamente seis unidades de tiempo.

Básicamente este programa de control le permite al agente ir verificando si al girar a la izquierda o a la derecha de la posición en la que se encuentra hay rastro. Si es así, avanza en esa dirección; en caso contrario avanza una casilla hacia adelante en la dirección en la cual estaba orientado originalmente. Tal como puede verse en la Figura 5.5 este mecanismo es suficiente para resolver el trazo Santa Fe consumiendo 572 unidades de tiempo. Las 28 unidades de tiempo restantes son utilizadas por el agente para continuar explorando casillas en busca de rastro.

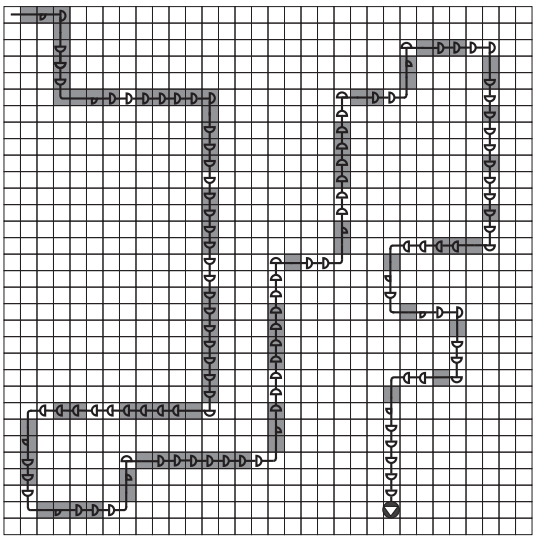

Figura 5.5: *Recorrido seguido usando el programa de control óptimo.*

Al haber consumido las 89 posiciones de comida del trazo y partiendo del supuesto que en la función objetivo no se tienen en cuenta las unidades de tiempo transcurridas, el valor de aptitud estandarizado de este programa de control es 0.

5.3. Trabajos realizados

La descripción original del problema de la Hormiga Artificial se presentó en el artículo de Jefferson et al. [50]. Los autores de dicho artículo plantearon las metas buscadas al diseñar y tratar de solucionar un problema como el presentado en este capítulo y su importancia para la Vida Artificial.

En dicho artículo los autores presentaron soluciones mediante una modificación al AG que trabaja con cadenas binarias que representan autómatas finitos y redes neuronales artificiales. Tras una serie de experimentos concluyeron que se obtienen mejores resultados para el primer tipo de representación que para el segundo. Es interesante ver que el tamaño de población seleccionado es excesivamente grande (65536 individuos) lo que provocó que se necesitara una gran cantidad de tiempo de procesamiento de un supercomputador para realizar los experimentos.

Curiosamente, en [50] sólo se presentan resultados para el trazo John Muir. En ningún momento se menciona el trazo Santa Fe.

Un año antes de la publicación de [50]; Koza, en el reporte técnico [59], presentó una solución al mismo problema empleando su naciente Programación Genética. Utilizar la representación de árbol y cambiar los parámetros del proceso evolutivo redujo a tal grado el

consumo de procesador y memoria que pudo encontrar una solución mediante una computadora estándar.

En el mismo congreso en el que Jefferson presentó su solución, Koza incluyó el artículo [60] en el que aborda por vez primera el trazo Santa Fe. Sin embargo, tras una larga serie de experimentos, análisis y cálculos Koza concluyó que la Programación Genética en su forma estándar necesita de ocho corridas independientes para asegurar que al menos una de ellas encontrará un árbol que le permita a la Hormiga Artificial colectar todas las posiciones de comida del rastro.

Los años 1997 y 1998 fueron productivos para el problema de la Hormiga Artificial en Programación Genética. Harries y Smith [43] compararon el desempeño de los operadores estándar de la PG contra una serie de operadores diseñados por ellos, así como también contra una variación de la búsqueda mediante Escalado de Colinas que opera sobre árboles por medio de operadores semejantes a los de PG.

Uno de los paisajes de aptitud que seleccionaron para realizar dicha comparación fue el problema de la Hormiga Artificial con el trazo Santa Fe. Sorprendentemente, los resultados obtenidos para la mayoría de paisajes de aptitud (incluido aquel seleccionado como caso de estudio en la presente tesis) favorecieron al Escalado de Colinas sobre la Programación Genética. Cabe aclarar que no se definió ningún límite a la profundidad máxima para este último método.

Chellapilla [12] publicó el mejor resultado reportado hasta mediados de 2007 en este problema. En dicho artículo el autor abordó diversos paisajes de aptitud típicamente resueltos mediante PG (incluido el problema de la Hormiga Artificial con el trazo Santa Fe) mediante una versión de PE (ver sección 2.3) con modificaciones a los métodos de mutación. Los resultados que obtuvo muestran una clara superioridad de su método sobre la PG estándar de Koza. Sobre todo para el problema de la Hormiga Artificial en el que la PG necesita prácticamente cuatro veces el número de evaluaciones que las requeridas por PE para encontrar una solución óptima.

Luke y Spector [70] presentaron una comparación entre el comportamiento de la PG funcionando únicamente mediante cruza y únicamente mediante macromutación para diversos paisajes de aptitud (entre ellos el problema de la Hormgia Artificial con el trazo Santa Fe) con diferentes tamaños de población y número de generaciones. Los resultados apoyan la afirmación hecha por Koza sobre el mejor comportamiento de cruzamiento sobre macromutación. Sin embargo, no presentaron ningún resultado utilizando diferentes probabilidades de ambos operadores en conjunto.

Otros autores como Ito et al. [49] y Kuscu [62] publicaron en ese periodo artículos en los que utilizaban el problema de la Hormiga Artificial como un caso de prueba sobre el que podían presentar las mejoras logradas mediante diferentes modificaciones realizadas al proceso evolutivo de la PG.

Caben destacar los artículos presentados por Langdon y Poli [63, 65, 64] en los que realizaron diferentes análisis del comportamiento de la Programación Genética sobre el trazo

Santa Fe.

El primero de ellos analiza la relación existente entre el hinchamiento, el operador de cruza y la selección basada en la aptitud. Para ello los autores realizaron múltiples corridas sobre el paisaje de aptitud del problema de la Hormiga Artificial con y sin selección, midiendo el tamaño de los árboles generados. Concluyeron que el hinchamiento es necesario para que la PG pueda avanzar y que está relacionado con el proceso de selección más que con el operador de cruza.

En [65] realizaron una exploración exhaustiva sobre el espacio de programas pequeños (con un máximo de 14 nodos) y un muestreo mediante series de Monte Carlo sobre el espacio de los programas grandes (árboles hasta con 500 nodos). Descubrieron que no importando el número de nodos, la cantidad de individuos con valor de aptitud bajo siempre será mucho mayor que la cantidad de individuos con valor de aptitud alto.

También observaron que la proporción de soluciones óptimas con respecto al total de programas alcanza su valor máximo (1×10^{-5}) cuando se tienen 18 nodos. Es claro que la proporción de soluciones óptimas es muy baja para este paisaje de aptitud, por lo que el problema de la Hormiga Artificial con el trazo Santa Fe se puede clasificar como un problema difícil de resolver para los métodos de búsqueda.

En este mismo artículo Langdon y Poli mostraron una comparación del número de evaluaciones necesitadas por diversas heurísticas para encontrar una solución al problema de la Hormiga Artificial con el trazo Santa Fe. Sorprendentemente la búsqueda aleatoria requiere el mismo "esfuerzo"[3] que la PG estándar.

En [64], Langdon y Poli presentaron una serie de modificaciones al problema de la Hormiga Artificial:

- **Límite de comida visible**: Se numeran las posiciones de comida del trazo y se le impone al agente un límite en cuanto al número de casillas de comida que puede observar.

- **Límite en la energía inicial**: El agente tiene una cantidad limitada de energía. Consume ésta con cada operación realizada y la aumenta con las posiciones de comida alcanzadas.

- **Agregar la velocidad a la aptitud**: El tiempo ocupado en atravesar el trazo es agregado a la función de aptitud.

Realizaron corridas de la PG mezclando las diferentes modificaciones al problema. Obtuvieron el mejor resultado con un límite de una sola posición de comida visible.

Miller y Thomson [75] utilizaron el problema de la Hormiga Artificial en el trazo Santa Fe como uno de los casos de prueba para la PGC (ver sección 4.3.2).

[3] Ver definición en sección 7.1

Un intento poco convencional por llevar a la realidad el problema de la Hormiga Artificial con el trazo Santa Fe fue presentado por Teuscher et al. en [103]: el trazo Lausanne y la arquitectura de un micro-robot. El objetivo es generar una población de robots que de manera paralela vayan buscando un programa de control óptimo para el paisaje de aptitud. Sin embargo, por diferencias existentes entre la implementación real y la simulada los experimentos no arrojan información útil.

O'neill y Connor [79] abordaron varios paisajes de aptitud (entre ellos el problema de la Hormiga Artificial) con la heurística Evolución Gramatical (ver secciones 2.4 y 3.2). Para trabajar sobre una versión sin modificaciones del problema el resultado obtenido es bueno.

Tomassini et al. [104] presentaron un estudio sobre la diversidad de la población en PG Multipoblación. Este enfoque facilita la paralelización del proceso evolutivo y agrega el concepto de islas. Las islas sirven para limitar la vecindad de la población. En dicho artículo utilizaron el problema de la Hormiga Artificial como uno de los paisajes de aptitud sobre los que realizaron el análisis y concluyeron que el comportamiento de la PG mejora con el enfoque propuesto.

Recientemente Christensen y Oppacher [17] aplicaron varias veces un método de generación aleatoria de árboles para resolver el problema de la Hormiga Artificial con el trazo Santa Fe. En cada iteración seleccionaron árboles pequeños con un buen valor de aptitud y generaron una nueva función que los caracteriza y los insertan en el conjunto de funciones del problema.

Reportaron que en la tercera iteración el método encuentra una solución tras 20696 evaluaciones y argumentaron que es el menor esfuerzo reportado hasta el momento. Sin embargo, no consideraron en dicha cantidad el número de evaluaciones que fueron necesarias en las corridas anteriores por lo que dicho resultado no refleja el esfuerzo total (o real).

La Tabla 5.2 muestra un resumen de los diferentes resultados reportados en la literatura para el problema de la Hormiga Artificial con el trazo Santa Fe.

Tabla 5.2: *Trabajos previamente publicados sobre el problema de la Hormiga Artificial con el trazo Santa Fe.*

Método	**Esfuerzo/1000**	**Probabilidad acumulada de éxito** $P(M, i)$
PG de Koza [58]	450	48/111
Búsqueda aleatoria [65]	450	-/-
PGDP [65]	336	-/-
Escalado de Colinas [43]	188	8/50
PGC neutral [75]	173	32/100
PE restringida [12]	136	**47/50**
Limite de comida visible 1 [64]	120	19/50
Evolución Gramatical [79]	102	-/-
Búsqueda mediante árboles aleatorios [17]	**21**	-/-

Capítulo 6

La función neutral

6.1. Origen

Como se comentó en el capítulo anterior, Langdon y Poli [65] realizaron una exploración exhaustiva sobre el espacio de programas pequeños (con un máximo de 14 nodos) y un muestreo mediante series de Monte Carlo sobre el espacio de los programas grandes (árboles hasta con 500 nodos).

La Figura 6.1 fue tomada de [65] y agrupa los resultados obtenidos en dichos muestreos. Cabe aclarar que los autores utilizaron un enfoque de maximización (valor de aptitud sin estandarizar) y miden el tamaño de los individuos mediante la cantidad de nodos de los programas.

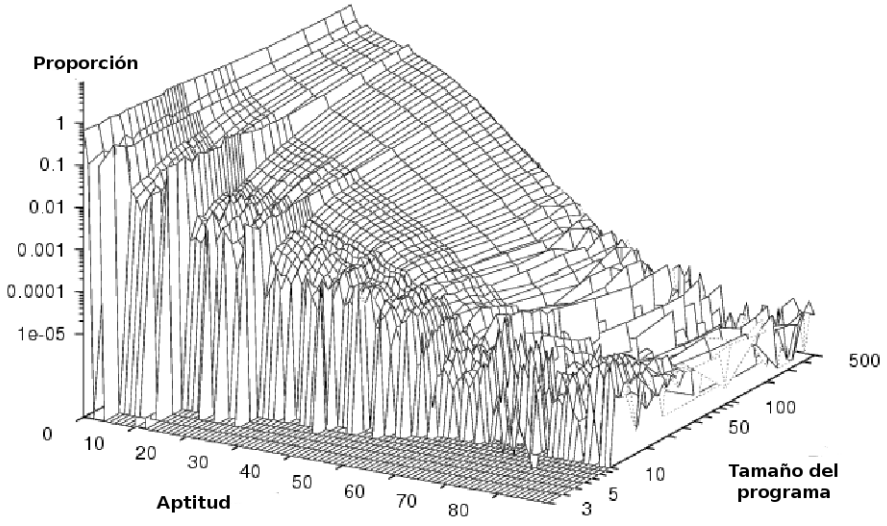

Figura 6.1: *Proporción de programas con un valor de aptitud dado.*

De la Figura 6.1 puede concluirse que a pesar de haber más individuos de mayor tamaño que individuos de menor tamaño, la proporción de programas óptimos (valor de aptitud 89) se mantiene relativamente constante para códigos con 10 o más nodos. También es claro que la cantidad de individuos con un valor de aptitud bajo siempre será mayor que la cantidad de individuos con valor de aptitud alto.

Estas propiedades convierten al trazo Santa Fe en un problema difícil de resolver mediante Programación Genética (y muchos otros algoritmos de exploración de espacios de búsqueda). Las causas de esto son el gran número de óptimos locales y que el paisaje de aptitud está lleno de redes neutrales que vinculan programas del mismo valor de aptitud en un laberinto denso y sofocante [65].

En esta tesis se propone aumentar la neutralidad de la representación para tratar de alisar el escarpado paisaje de aptitud conocido como el problema de la Hormiga Artificial con el trazo Santa Fe.

6.2. Funcionamiento

Para lograr aumentar la neutralidad del problema se propone agregar un elemento al conjunto de funciones. El elemento a agregar se llama *Neut_progn* y tiene aridad 3. Su rama izquierda, llamada **selector**, es un número entero en el intervalo $[0, 3]$. Las ramas central y derecha son subárboles típicos.

Para permitir la neutralidad explícita, el comportamiento de *Neut_progn* cambia dependiendo del valor del selector. Si vale 0 ningún otro argumento se ejecuta. Cuando vale 1 sólo se llama el segundo argumento (rama central). Si es 2 se llama al tercer argumento (rama derecha). Finalmente, si vale 3 ambos (segundo y tercer argumentos) se llaman secuencialmente. La Tabla 6.1 resume esta información.

Tabla 6.1: *Comportamiento de la función Neut_progn.*

Valor de selector	Comportamiento
0	No se llama ningún otro argumento.
1	Solo se ejecuta el segundo argumento.
2	Solo se ejecuta el tercer argumento.
3	Segundo y tercer argumentos se llaman de manera secuencial.

La Figura 6.2 muestra cómo la cantidad de código inactivo está relacionada con el valor del selector. Es importante enfatizar que el comportamiento de un programa de control que incluye al menos una aparición de *Neut_progn* puede cambiar drásticamente si se realiza una modificación mínima en el valor de algún selector.

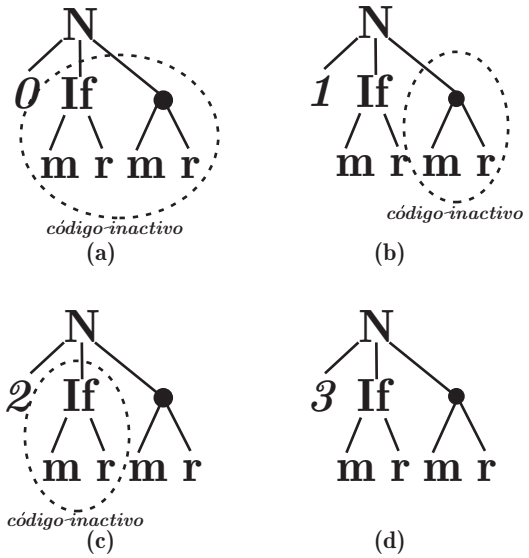

Figura 6.2: *Ejemplo del comportamiento de la función Neut_progn.*

Parece correcto afirmar que agregar *Neut_progn* al conjunto de funciones del problema de la Hormiga Artificial aumentará la cantidad de código inactivo de los individuos. Sin embargo, cuando se utiliza mutación por nodo los intrones agregados se pueden convertir fácilmente en código activo mediante la mutación de un selector. Mientras esto no ocurra los operadores de cruza y mutación pueden ir construyendo bloques útiles en las ramas inactivas del programa.

6.3. Peculiaridades

Agregar la función *Neut_progn* modifica ciertas propiedades del conjunto de funciones. Esto provoca que se deban realizar algunas validaciones y modificaciones al proceso de Programación Genética. Dichos cambios se deben principalmente a la diferencia de tipos existente entre el selector y el resto de elementos terminales.

6.3.1. Cruza

El operador de cruzamiento no tiene gran afectación. Únicamente se debe validar que no se seleccione como punto de cruza de ninguno de los dos padres la rama entre cualquier *Neut_progn* y su selector. Si esto llega a ocurrir se debe escoger de manera aleatoria un nuevo punto de cruza. Dicha validación debe mantenerse vigente para evitar que el caso se vuelva a presentar.

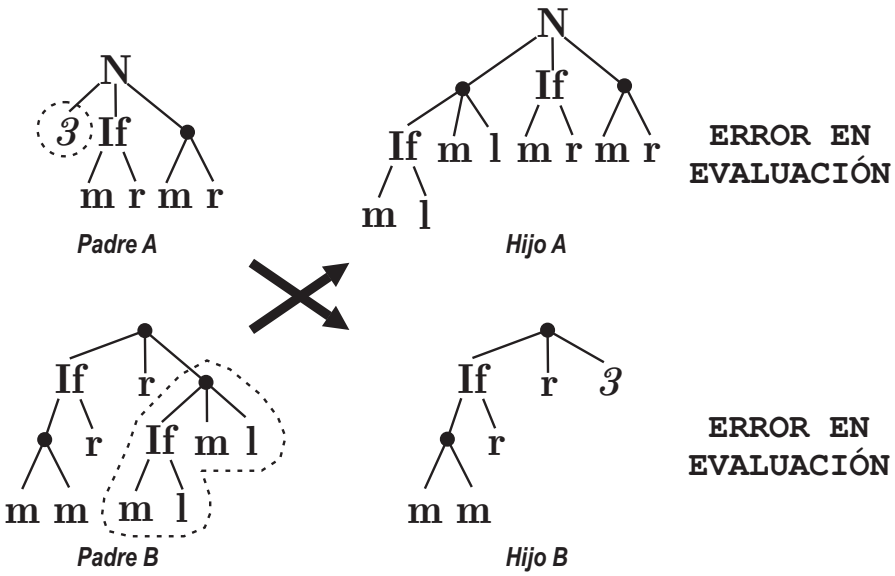

Figura 6.3: *Proceso no vigilado de cruza.*

Esto debe realizarse por la diferente tipificación de nodos entre el selector y cualquier otra rama que se pueda generar para el problema de la Hormiga Artificial. Tal como se muestra en la Figura 6.3 en caso de no evitar la presencia de un cruzamiento de este tipo ocurrirá un error en la evaluación de ambos individuos.

6.3.2. Mutación

Cuando se ocupa mutación de nodos (ver sección 3.2.5) con los conjuntos de funciones y terminales estándar del problema de la Hormiga Artificial (F=\{*IfFoodAhead, Progn2, Progn3*\} y T=\{*move, left, right*\}) se puede intercambiar cada terminal con cualquiera de los otros dos. Pero debido a diferentes aridades solamente se pueden intercambiar *Progn2* e *IfFoodAhead*. Esto quiere decir que se deben ignorar las mutaciones que ocurran sobre *Progn3*.

Sin embargo, cuando se agrega *Neut_progn* surgen dos nuevas posibles soluciones:

La primera es ignorar las mutaciones en *Progn3* y las mutaciones en *Neut_progn* (**método simple**).

La segunda opción (**método avanzado**) es cambiar *Neut_progn* con *Progn3* (misma aridad) y viceversa. Sin embargo, al hacer esto también se debe cambiar la rama izquierda del subárbol:

- En caso de una mutación de *Neut_progn* a *Progn3* se debe cambiar el selector por un subárbol nuevo mediante una mutación por expansión (ver sección 3.2.5). La Figura 6.4 ejemplifica este proceso.

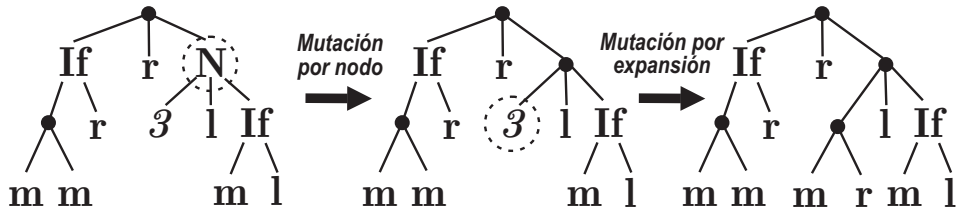

Figura 6.4: *Mutación de Neut_progn a Progn3.*

- En el caso de una mutación de *Progn3* a *Neut_progn* el subárbol izquierdo se debe borrar y en su lugar se debe colocar un selector con un valor entero aleatorio tomado del intervalo $[0, 3]$. Dicho cambio es un tipo especial de mutación por eliminación de subárboles (ver sección 3.2.5). La Figura 6.5 muestra un ejemplo.

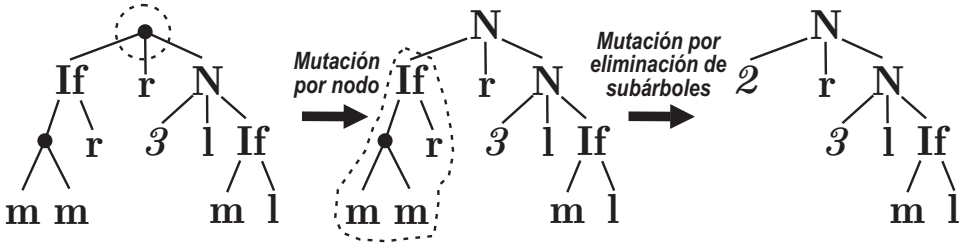

Figura 6.5: *Mutación de Progn3 a Neut_progn.*

Finalmente, siempre que se deba mutar un selector se asigna un valor entero aleatorio del intervalo $[0, 3]$ (diferente de su valor actual) como valor mutado de éste.

Se tiene con esto que el operador de mutación a aplicar en el método avanzado, a pesar de darse normalmente en forma de mutación de nodos, deberá resolver ciertas combinaciones valiéndose de métodos de mutación diferentes, por lo que se le considera un método de mutación mixto.

Capítulo 7

Experimentos

Se realizaron tres experimentos. En ellos se observó, analizó y discutió el comportamiento de la Programación Genética en el problema de la Hormiga Artificial bajo la influencia de la función neutral descrita en el capítulo anterior y sin la presencia de ésta.

7.1. Diferentes probabilidades de cruza y mutación

El primer experimento se planteó para analizar si al agregar la función neutral al conjunto de funciones del Problema de la Hormiga Artificial la Programación Genética encuentra con mayor facilidad una solución óptima.

Se compararon los resultados producidos por la PG cuando la función neutral se agrega al conjunto de funciones y cuando esto no ocurre. Se realizaron múltiples corridas con diferentes probabilidades de cruza y mutación. Se consideraron dos medidas de desempeño diferentes.

7.1.1. Métricas

Las dos medidas de desempeño usadas en el primer experimento fueron:

- Esfuerzo [58]
- Promedio de aptitud del mejor individuo de cada corrida.

Esfuerzo se define como el número de programas a evaluar necesarios para asegurar que se encontrará una solución óptima. Para poder calcularlo (7.2) se debe encontrar el número mínimo de corridas independientes $R(z)$ necesarias para alcanzar una probabilidad de éxito dada (7.1).

$$R(z) = \left\lceil \frac{\log(1-z)}{\log(1-P(M,i))} \right\rceil \tag{7.1}$$

$$I(M,i,z) = MR(z)i \tag{7.2}$$

Donde z es la probabilidad de éxito deseada (comúnmente seleccionada como 0.99), M representa el tamaño de la población, i el número total de generaciones y $P(M,i)$ es la probabilidad acumulada de éxito.

El esfuerzo fue planteado por John Koza [58] y es una de las métricas más utilizadas para comparar el desempeño de diferentes métodos de búsqueda. Esto se debe a que las variables son fácilmente adaptables a los diferentes algoritmos existentes. A pesar de que Christensen y Oppacher [16] criticaron la precisión de la ecuación planteada por Koza actualmente sigue siendo muy ocupada por investigadores del área.

El promedio de aptitud del mejor individuo en la generación t se define mediante:

$$\overline{f_{min}}(t) = \frac{\sum_{j=1}^{N} min(f_{I_1}^t, f_{I_2}^t, \ldots f_{I_n}^t)_j}{N} \tag{7.3}$$

donde n representa el tamaño de la población, N el número de corridas independientes realizadas, $f_{I_i}^t$ corresponde al valor de aptitud del i-ésimo individuo en la generación t y $min(f_{I_1}^t, f_{I_2}^t, \ldots f_{I_n}^t)_j$ hace referencia a la aptitud más baja de toda la población en la generación t para la j-ésima corrida.

Para el primer experimento se utilizó (7.3) en la última generación (con $t = 333$). En otras palabras, sólo se calculó $\overline{f_{min}}(333)$.

La métrica del promedio de aptitud del mejor individuo de cada corrida permite identificar la distancia promedio del mejor individuo de cada una de las diferentes corridas con respecto a algún óptimo global. Mientras que para el esfuerzo un individuo con aptitud uno y otro individuo con aptitud 27 corresponden a programas de control que no consumieron todas las casillas del rastro, dicha diferencia se ve reflejada en el promedio de aptitud del mejor individuo de cada corrida.

Por esta razón se incluyeron ambas métricas como medidas de desempeño de la Programación Genética en la presente tesis. Sin embargo, no son las únicas medidas de desempeño utilizadas en los Algoritmos Evolutivos.

7.1.2. Planteamiento

Tabla 7.1: *Parámetros para la PG en primer experimento.*

Parámetro	Valor
Tamaño de población	150
Número de generaciones	333
Selección	Torneo (con tamaño de 3 individuos).
Profundidad inicial	3
Profundidad final	8
Probabilidad de cruza	Variable.
Probabilidad de mutación (por nodo)	Variable.
Función de Aptitud	Comida remanente del trazo.

Se realizaron una serie de experimentos previos (uno de ellos descrito en el apéndice A). Con base en los resultados obtenidos en dichos experimentos se definieron los parámetros que se presentan en la Tabla 7.1. Estos parámetros fueron utilizados en todas las corridas del presente experimento.

Las siguientes aclaraciones aplican para todos los experimentos de la presente tesis:

- Se utilizó el método mitad y mitad para generar todas las poblaciones iniciales (ver sección 3.2.3).
- El operador de cruza utilizado fue el cruzamiento por intercambio de subárboles (ver sección 3.2.4).
- El operador de mutación utilizado fue la mutación de nodos (ver sección 3.2.5).
- El mejor individuo de toda la población en cada generación pasa automáticamente a la siguiente (elitismo de un individuo, ver sección 3.2.8).

Se utilizaron dos configuraciones diferentes del problema. Para cada configuración se realizaron tres series de corridas. En la primera de dichas series se utilizó el conjunto de funciones tradicional y se llama **serie normal**. La segunda incluye la función *Neut_progn* con el método simple de mutación (ver sección 6.3.2) y se llama **serie neutral simple**. La última incluye la función *Neut_progn* con el método neutral avanzado de mutación y se llama **serie neutral avanzada**.

En la Tabla 7.1 no se definieron probabilidades de cruza y mutación porque para cada serie se mapearon todas las combinaciones existentes de las probabilidades de cruza {20, 45, 60, 85, 95, 100} y las probabilidades de mutación por nodo {0, 1, 2, 3, . . . , 11}.

Para que los resultados fueran significativos, se realizaron 50 corridas independientes para cada combinación. Se calcularon el esfuerzo y el promedio de la aptitud del mejor individuo de la última generación para cada combinación.

A continuación se determinarán las características de cada una de las configuraciones usadas.

Configuración base

La configuración base toma las características estándar del problema de la hormiga artificial para las tres series de corridas que se describieron anteriormente (serie normal, serie neutral simple y serie neutral avanzada).

Configuración con agente condicionado

Langdon y Poli [64] realizaron diversas modificaciones al paisaje de aptitud del problema de la Hormiga Artificial con el trazo Santa Fe. Obtuvieron el valor de aptitud más bajo cuando limitaron la capacidad perceptiva del agente a una sola posición del rastro. Es decir, cuando el agente se encuentra en la posición de salida, no importando los movimientos que haga, solo podrá percibir la primera posición del rastro. Una vez haya consumido dicha casilla podrá percibir la segunda posición, etc.

Esta modificación es útil al proceso evolutivo porque elimina la posibilidad de que el agente se salte fragmentos del trazo. Evitando con esto zonas del espacio de búsqueda que difícilmente lo conducirían hacia programas óptimos. En otras palabras, limitar la capacidad perceptiva del agente del modo en que Langdon y Poli lo sugieren cambia el paisaje de aptitud del trazo Santa Fe. Este cambio consiste en la eliminación de algunos óptimos locales.

Se propone agregar la función neutral a la versión del problema de la Hormiga Artificial con un límite en la capacidad perceptiva del agente. Dicha combinación le permitirá a la PG moverse a través de redes neutrales en una una simplificación del paisaje de aptitud del trazo Santa Fe. Se sugiere que estas características se reflejarán en una mejora en el desempeño de la Programación Genética en el problema de la Hormiga Artificial con el trazo Santa Fe.

En la configuración con agente condicionado se utilizaron los mismos parámetros para la PG en las tres series usadas en la configuración base. La única diferencia en dicha configuración es la inclusión de la modificación para limitar la capacidad perceptiva del agente. Por motivos de tiempo sólo se limitó la capacidad a una posición de rastro (valor con el que Langdon y Poli encontraron la mejor solución [64]).

7.1.3. Resultados

Tras realizar todas las corridas se obtuvieron los siguientes resultados.

Esfuerzo

Dado que se realizó una gran cantidad de corridas, el problema inicial a resolver con el primer experimento fue encontrar la manera de presentar los resultados en conjunto sin que se perdiera la información arrojada por el cálculo del esfuerzo. Para ello se optó por mapas comparativos de los valores obtenidos por la serie normal y las series neutrales.

En dichos mapas se pierde la información relacionada con el valor de esfuerzo obtenido para cada combinación. Sin embargo, se muestra claramente cuál de los dos métodos (normal o neutral) requiere una menor cantidad de evaluaciones para encontrar un programa óptimo. Si el lector requiere datos con mayor grado de detalle, el apéndice B presenta gráficas y tablas con el esfuerzo obtenido para cada serie de ambas configuraciones.

A continuación se presentarán y discutirán los mapas comparativos de esfuerzo de las series neutrales y normales. Primero se comentarán los resultados obtenidos con la configuración base. Tras esto, se detallarán los resultados obtenidos para la configuración con agente condicionado.

La Figura 7.1 muestra el mapa comparativo de esfuerzo de la serie neutral simple contra la serie normal para la configuración base. En éste las regiones blancas indican que el valor de esfuerzo de la versión neutral simple es menor que el valor de esfuerzo de la serie normal. Regiones negras significan que el valor de esfuerzo de la serie normal es menor que el valor de esfuerzo de la serie neutral simple. Finalmente, las regiones grises significan que ambas tuvieron el mismo valor de esfuerzo.

Se puede observar en la Figura 7.1 que fuera de la zona comprendida entre la probabilidad de mutación uno a siete por ciento y probabilidad de cruza de 20 a 90 % la serie neutral simple obtuvo un valor menor de esfuerzo que la serie normal.

Lo que ocurre en la zona intermedia queda poco claro en este mapa. Sin embargo, se verá que la existencia de zonas bien definidas en las que las series neutrales obtienen valores de esfuerzo iguales o menores a los valores obtenidos por la serie normal son una constante en todas las figuras de esta sección.

La Figura 7.2 muestra el mapa comparativo del esfuerzo de la serie neutral avanzada contra la serie normal para la configuración base. Se utiliza la misma dinámica de relación que la descrita para la Figura 7.1.

Igual que en el caso anterior, la Figura 7.2 muestra una zona bien definida en la cual la serie neutral avanzada obtiene un valor de esfuerzo mayor al encontrado por la serie normal. Comparando ambas figuras se puede observar que en la segunda de ellas la zona en cuestión se localiza con valores bajos de la probabilidad de mutación (menores a 4 %).

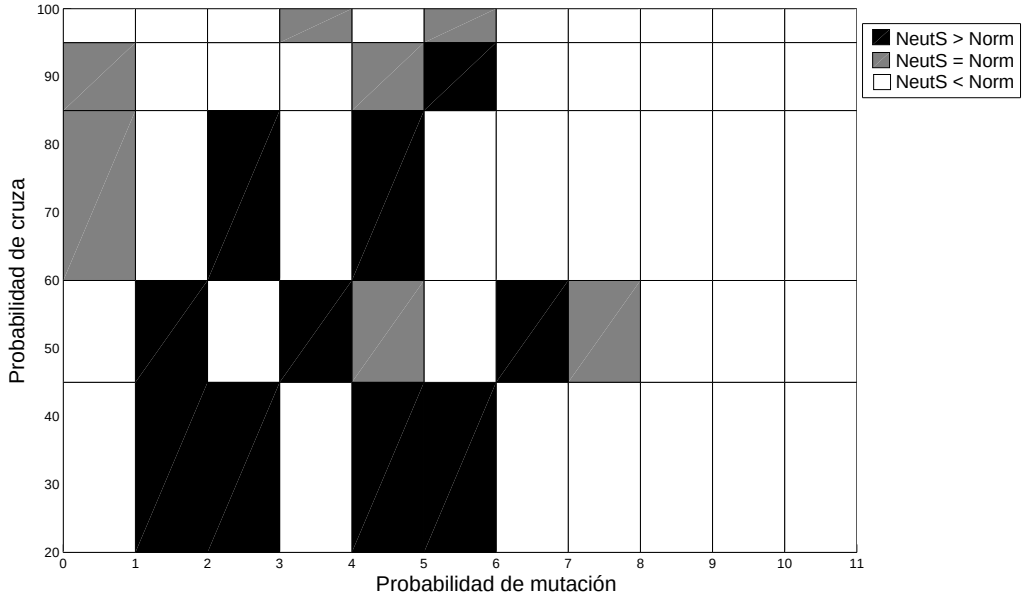

Figura 7.1: *Comparación de esfuerzo entre la serie normal y la neutral simple para configuración base.*

Esto indica que la serie neutral avanzada presenta mayores dificultades que la serie neutral simple para encontrar programas de control óptimo cuando la probabilidades de mutación es baja. Sin embargo, cuando la probabilidad de mutación es cero ambos métodos (neutral simple y neutral avanzado) deberían comportarse de la misma manera. Dado que los resultados obtenidos en las dos series de corridas difieren para dicho caso, sería prudente en un futuro realizar una mayor cantidad de corridas para disminuir la influencia de corridas atípicas.

Como se comentó en sección 7.1.2 se realizaron las tres series de corridas para dos configuraciones diferentes. A continuación se presentan los mapas comparativos de esfuerzo de las series normal, neutral simple y neutral avanzada de la configuración con agente condicionado.

La Figura 7.3 muestra el mapa comparativo de esfuerzo de la serie neutral simple contra la serie normal para la configuración con agente condicionado. Se utiliza la misma dinámica de relación que la descrita para la Figura 7.1.

Al igual que lo sucedido con la configuración base, en esta figura se puede identificar una zona en la que la serie neutral simple requiere un esfuerzo mayor que la serie normal para encontrar una solución óptima. Sin embargo, en este caso la zona pareciera seguir el comportamiento de una parábola inversa que tiene su origen en probabilidad de cruza 100 % y probabilidad de mutación cuatro por ciento. En la sección 7.1.4 se da una explicación a este fenómeno.

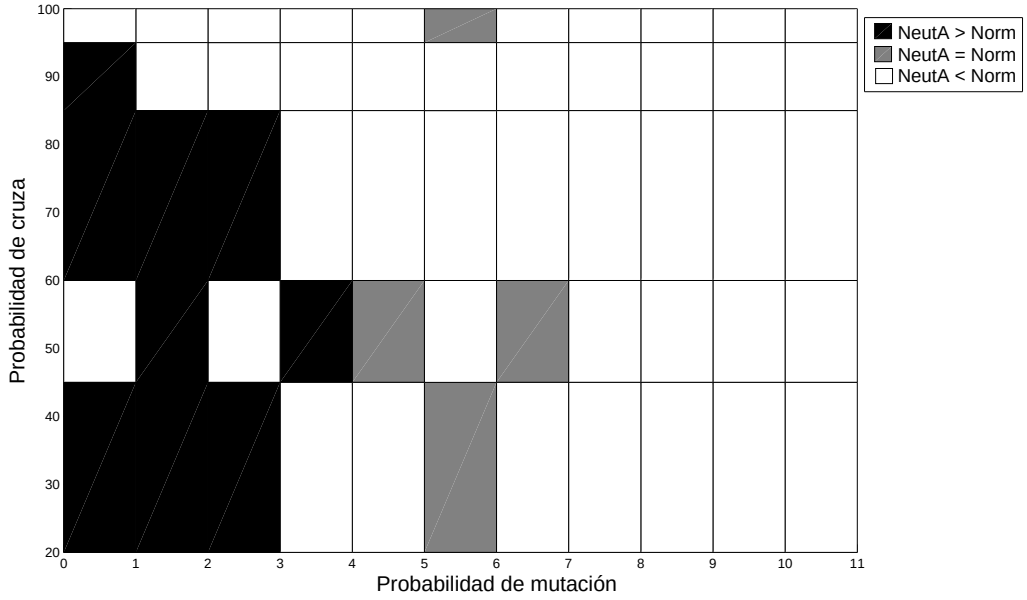

Figura 7.2: *Comparación de esfuerzo entre la serie normal y la neutral avanzada para configuración base.*

Otro rasgo a destacar es la disminución de casos en los que la serie normal se comporta mejor que la serie neutral simple. Mientras que con la configuración base se tenían diez combinaciones en los que la primera se comportó mejor que la segunda, con la configuración con agente condicionado fueron 6 los casos en los que la serie normal tuvo un valor de esfuerzo menor a la serie neutral simple. Esta situación parece indicar que la función neutral proporciona un mayor beneficio cuando se limita la capacidad perceptiva del agente.

La Figura 7.4 muestra el mapa comparativo de esfuerzo de la serie neutral avanzada contra la serie normal para la configuración con agente condicionado. Se utiliza la misma dinámica de relación que la descrita para la Figura 7.1.

Dicha figura no presenta claramente el patrón parabólico encontrado en la Figura 7.3. Sin embargo, varios de los puntos con probabilidades de mutación entre tres y ocho por ciento muestran un desempeño igual para la serie neutral avanzada y la serie normal o incluso puntos en los que esta última se comporta mejor. De nuevo se puede identificar una zona de conflicto.

En la sección 7.1.4 se plantea una hipótesis sobre la existencia de las zonas de conflicto identificadas en las figuras 7.1, 7.2, 7.3 y 7.4. Con base en dicha hipótesis se dará una explicación a la dinámica de evolución de la Programación Genética en la presencia y ausencia de la función neutral.

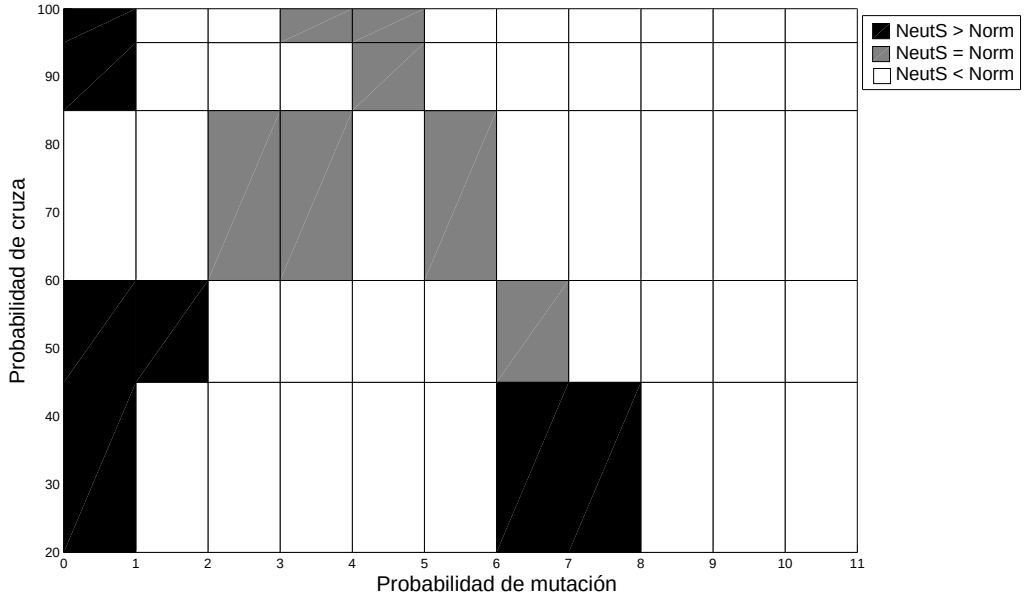

Figura 7.3: *Comparación de esfuerzo entre la serie normal y la neutral simple para configuración con agente condicionado.*

Aptitud promedio del mejor individuo

Con la métrica del promedio de aptitud del mejor individuo de cada corrida se presentó un problema parecido al planteado para el esfuerzo. Cómo representar los resultados en conjunto sin que se perdiera la información arrojada por el promedio de aptitud del mejor individuo.

El esfuerzo se calcula con base a un valor entero positivo (número mínimo de corridas para alcanzar una probabilidad de éxito dada). Esto permitió plantear el tipo de mapas comparativos presentados en las páginas anteriores. Sin embargo, el promedio de aptitud del mejor individuo es un número racional y la diferencia entre un promedio A y otro B puede ser muy pequeña. Por estas razones no se pueden utilizar los mismos mapas comparativos para el esfuerzo y para el promedio de aptitud del mejor individuo.

Se diseñó un mapa comparativo sobre la resta del promedio de aptitud del mejor individuo encontrado por la serie neutral para cada combinación al semejante encontrado por la serie normal. En dicho mapa, cada punto se presenta en un tono de gris. La tonalidad de gris asignada corresponde al resultado de la resta. En dicha escala el blanco indica que el resultado de la resta fue igual a 12 unidades de aptitud mientras que el negro indica que el resultado de la resta fue igual a -18 unidades de aptitud.

De este modo, si una celda tiene un tono de gris obscuro quiere decir que el valor del promedio de aptitud del mejor individuo obtenido por la serie neutral es mayor que el pro-

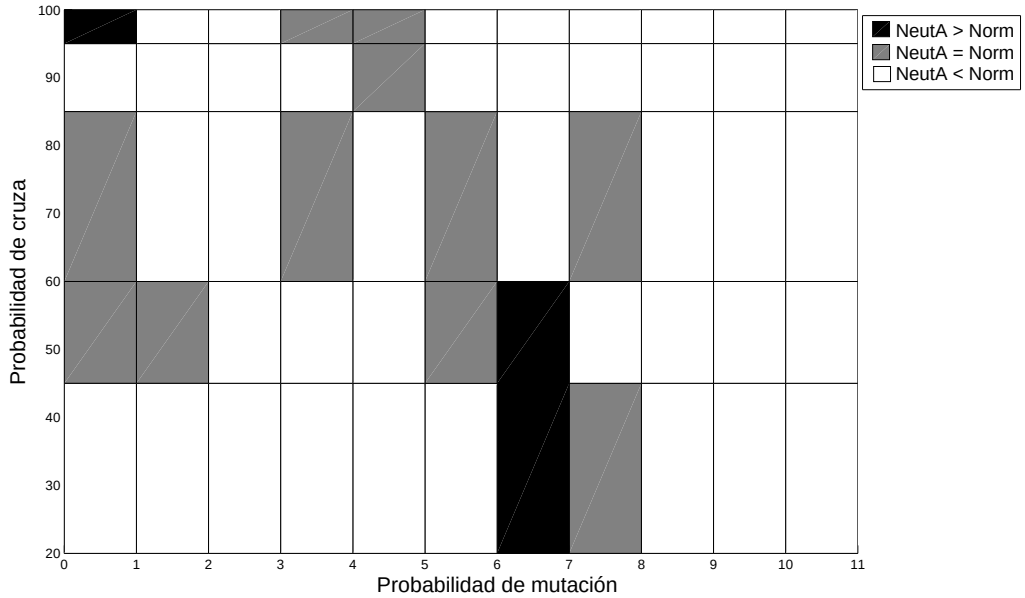

Figura 7.4: *Comparación de esfuerzo entre la serie normal y la neutral avanzada para configuración con agente condicionado.*

medio de aptitud del mejor individuo obtenido por la serie normal. Por el contrario, si la celda tiene un tono de gris claro quiere decir que el valor del promedio de aptitud del mejor individuo obtenido por la serie neutral es menor que el promedio de aptitud del mejor individuo obtenido por la serie normal.

Como puede verse este tipo de mapa comparativo no sólo aporta información sobre cuál de las series obtuvo un menor promedio de aptitud sino que permite identificar la magnitud de la diferencia de los valores encontrados en ambas series. En el apéndice B se presentan gráficas y tablas con el promedio de aptitud del mejor individuo y la desviación estándar de aptitud del mejor individuo obtenidos para cada serie de ambas configuraciones.

La Figura 7.5 muestra el mapa comparativo de promedio de aptitud del mejor individuo de la serie neutral simple contra la serie normal para la configuración base. Como se acaba de explicar, en dicho mapa mientras sea más claro el tono de gris el promedio obtenido por la serie normal será mayor que el obtenido por la serie neutral simple. Un tono obscuro de gris indicará que el valor obtenido por la serie normal es menor que el obtenido por la serie neutral simple.

En dicha figura se puede observar que el tono de gris de las diferentes celdas varía poco entre unas y otras. Ciertos elementos en la zona central del mapa tienden a ser más obscuros que el resto, pero es una diferencia que difícilmente se puede percibir con la técnica de comparación utilizada. En general, el mapa tiende a tonos claros de gris. Esta tendencia

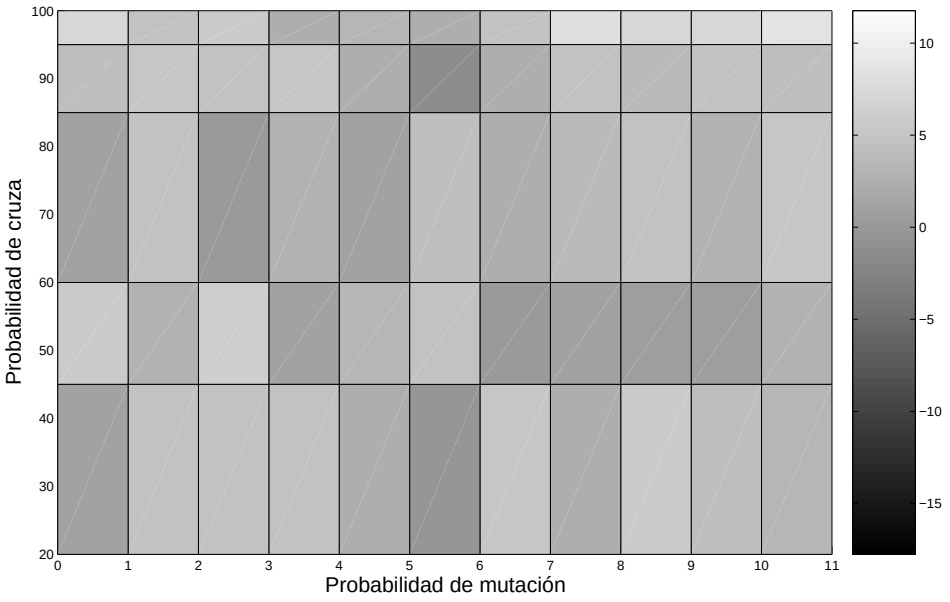

Figura 7.5: *Comparación de promedio de aptitud del mejor individuo entre la serie normal y la neutral simple para configuración base.*

indica que la serie neutral simple obtuvo valores de promedio de aptitud del mejor individuo más bajos que la serie normal con la configuración base.

La Figura 7.6 muestra el mapa comparativo de promedio de aptitud del mejor individuo de la serie neutral avanzada contra la serie normal para la configuración base. Se utiliza la misma dinámica de relación que la descrita para la Figura 7.5.

Según el mapa mostrado en dicha figura el resultado obtenido con la serie neutral avanzada difiere poco del obtenido por la serie neutral simple. Los tonos de gris son ligeramente más claros en la Figura 7.6 con respecto a la Figura 7.5. Se observa que las celdas con tonos de gris más obscuros en la figura anterior se mantienen como las celdas más obscuras en la figura actual.

Estos resultados hacen pensar que las superficies de la aptitud promedio del mejor individuo para las series neutral simple y neutral avanzada mantiene un curvatura semejante entre sí. Por el contrario la serie normal presenta picos (valores altos en el promedio de aptitud del mejor individuo) en celdas en las que las otras dos series se mantienen aproximadamente equidistantes.

Esta semejanza en el comportamiento de las series neutrales y su diferencia con la serie normal se discutirá a fondo en la sección 7.1.4.

Como se comentó en sección 7.1.2, se realizaron las tres series de corridas para dos configuraciones diferentes. A continuación se presentan los mapas comparativos de promedio de

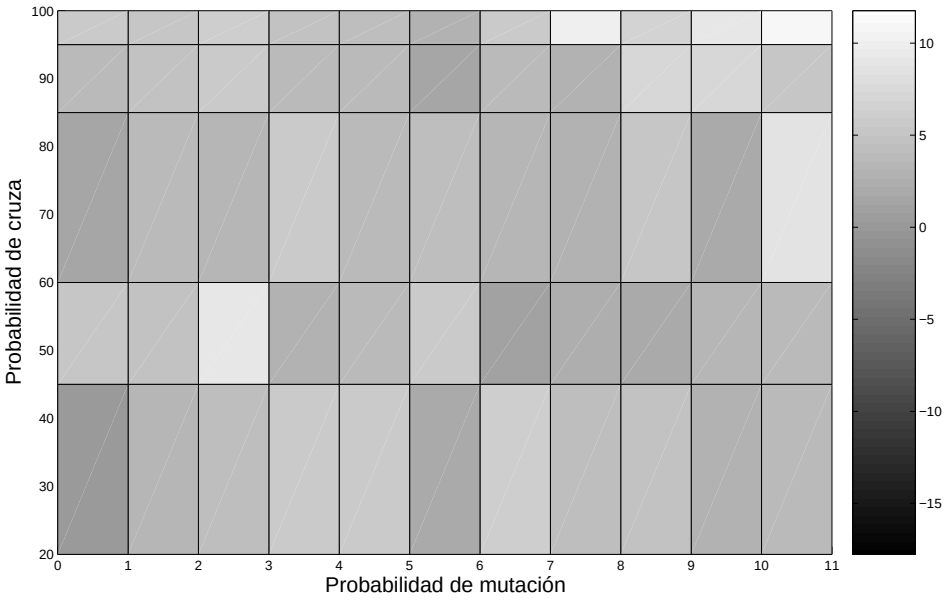

Figura 7.6: *Comparación de promedio de aptitud del mejor individuo entre la serie normal y la neutral avanzada para configuración base.*

aptitud del mejor individuo de las series normal, neutral simple y neutral avanzada de la configuración con agente condicionado.

La Figura 7.7 muestra el mapa comparativo de promedio de aptitud del mejor individuo de la serie neutral simple contra la serie normal para la configuración con agente condicionado. Se utiliza la misma dinámica de relación que la descrita para la Figura 7.5.

Los resultados observados en dicha figura difieren de aquellos identificados en la Figura 7.5. Se reconocen zonas con mucho contraste (casillas con tonos claros al lado de casillas obscuras) cuando la probabilidad de cruza es baja. Ocurre lo mismo cuando la probabilidad de mutación es nula. En la sección 7.1.4 se dará una explicación a esta diferencia en el comportamiento del promedio de aptitud del mejor individuo para las distintas configuraciones.

La Figura 7.8 muestra el mapa comparativo de promedio de aptitud del mejor individuo de la serie neutral avanzada contra la serie normal para la configuración con agente condicionado. Se utiliza la misma dinámica de relación que la descrita para la Figura 7.5.

En dicha figura nuevamente el margen izquierdo y el margen inferior son zonas con mucho contraste. Sin embargo, este comportamiento cambiante no se presenta con la frecuencia con que lo hace en el mapa comparativo de la Figura 7.5. Esta diferencia sugiere que la serie neutral avanzada se comporta con mayor uniformidad para diferentes probabilidades de cruza y mutación para la configuración con agente condicionado que la serie neutral simple.

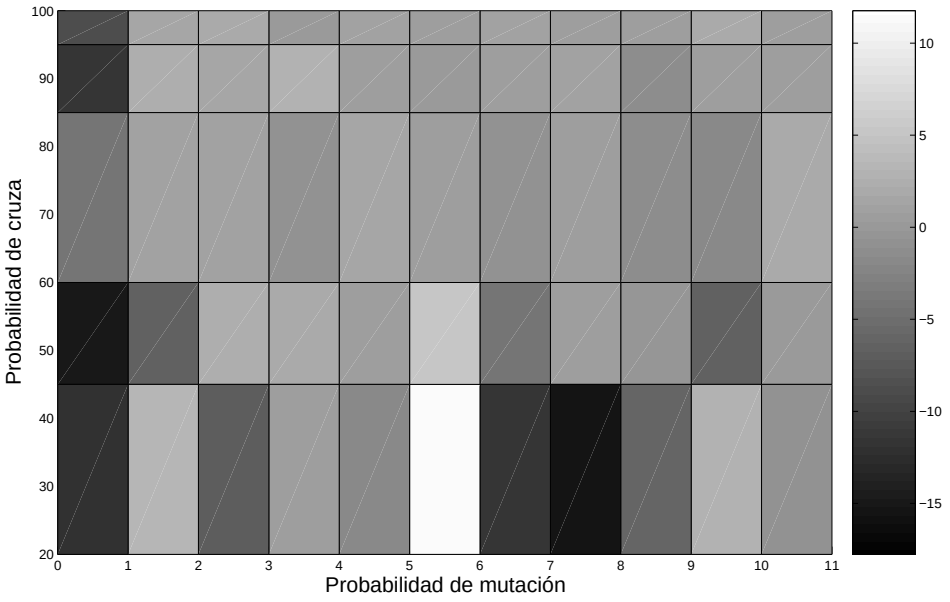

Figura 7.7: *Comparación del promedio de aptitud del mejor individuo entre la serie normal y la neutral simple para configuración con agente condicionado.*

En la siguiente sección se discutirán a detalle las diferencias presentes en las tres series bajo las distintas configuraciones tomando en cuenta las dos métricas propuestas.

7.1.4. Discusión

En ambas configuraciones se detectó la presencia de zonas de los mapas comparativos de esfuerzo en los que la serie normal se comporta igual o mejor que las series neutrales. Fuera de dichas zonas las series neutrales requieren menos esfuerzo que la serie normal para encontrar un programa de control óptimo.

Como se comentó en el capítulo 4 el código inactivo presenta dos características deseables para el proceso evolutivo de la PG. Sirve como *reserva génica* para la creación de bloques de código que posteriormente podrían ser útiles en algún individuo. Brindan protección global a los individuos contra cruzas y mutaciones destructivas.

Aplicando dicho principio al caso de estudio surge la siguiente explicación a la diferencia del comportamiento de las series neutrales y la serie normal:

El activar y desactivar código inactivo provocado por la función neutral mediante el operador de mutación o realizar cruzas constructivas sobre éste, permite que la PG encuentre buenas soluciones en las series neutrales siempre que las probabilidades de mutación y cruza no sean muy bajas. Por el contrario, la serie normal presenta mayores dificultades para

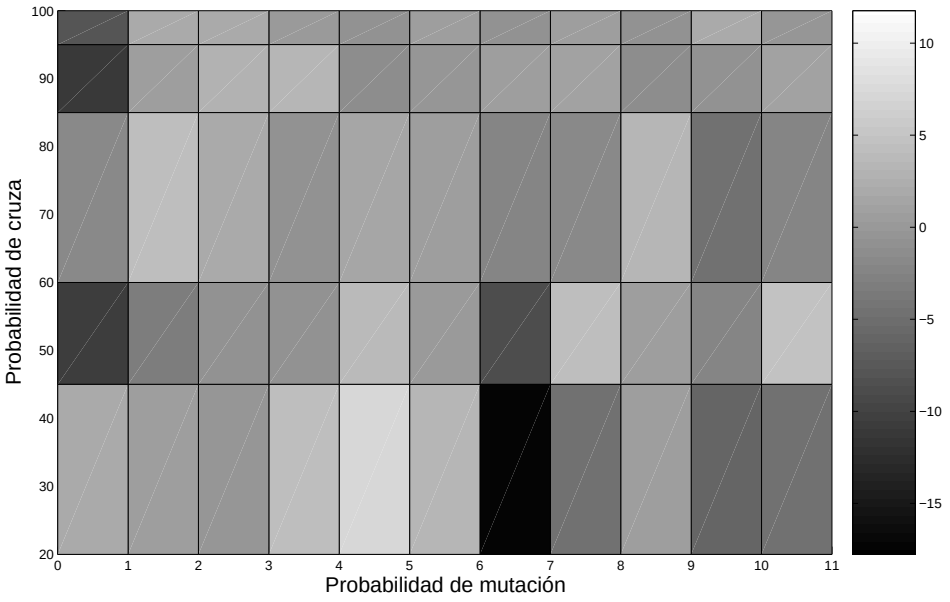

Figura 7.8: *Comparación del promedio de aptitud del mejor individuo entre la serie normal y la neutral avanzada para configuración con agente condicionado.*

encontrar una buena solución dadas diferentes probabilidades de cruza y mutación.

Tras observar las diferentes figuras del apéndice B se plantea que para cada configuración existe una probabilidad de mutación p_{m_o} con la que la serie normal obtiene un valor de esfuerzo menor. Mientras la probabilidad de mutación escogida esté cerca de p_{m_o} los resultados de la serie normal mejorarán con respecto a aquellos obtenidos por las series neutrales.

A modo de ejemplo se presentan las figuras 7.9 y 7.10 (tomadas del apéndice B). Dichas figuras muestran un valle muy pronunciado para la serie normal y algo más parecido a llanuras para las series neutrales. En los puntos bajos de los valles es cuando la serie normal se comporta mejor que las neutrales.

La función neutral aumenta las posibilidades de obtener un programa de solución óptimo en un menor número de corridas sin necesidad de buscar p_{m_o}.

Tal como se puede ver en las figuras del apéndice B la serie neutral avanzada mantiene para muchas de las combinaciones mapeadas un esfuerzo por debajo del producido por la serie neutral simple con la configuración base. La diferencia entre ambas series es la forma de resolver las mutaciones que ocurren en *Neut_progn* y *Progn3*. La serie neutral avanzada permite el intercambio de ambas funciones. Este tipo de mutación provoca que el código inactivo de la función neutral se vuelva activo con gran facilidad. Con la misma facilidad permite que el código activo de *Progn3* se vuelva código inactivo. Este tipo de intercambios parecieran ser útiles frente al problema de la Hormiga Artificial con el trazo Santa Fe.

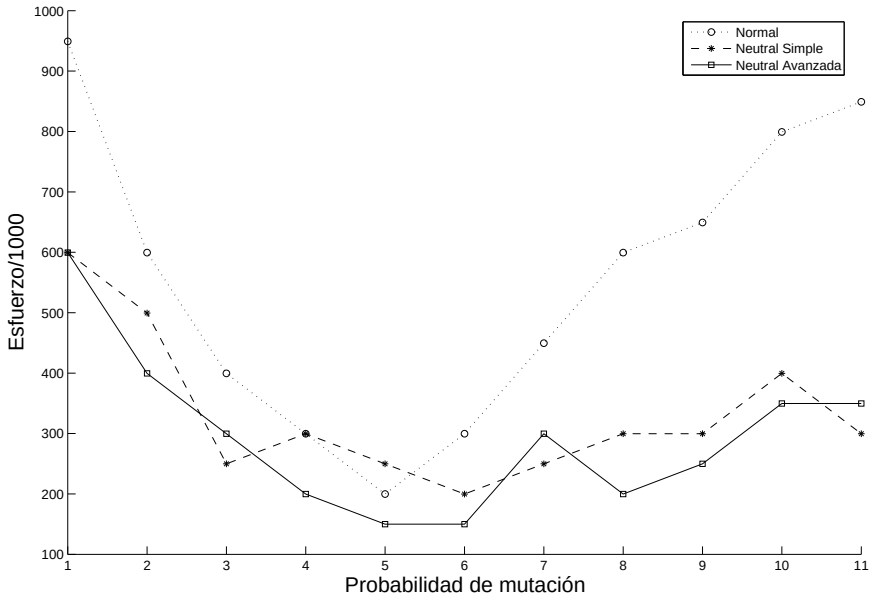

Figura 7.9: *Comparación de esfuerzo/1000 con 85 % de probabilidad de cruza para configuración base.*

Sin embargo, para la configuración con agente condicionado no sucede lo mismo. En dicha configuración ninguna de las dos series neutrales se mantiene arriba de la otra para una mayoría del total de combinaciones mapeadas. Esta diferencia posiblemente se deba al cambio en el paisaje de aptitud ocasionado por el límite en la capacidad perceptiva del agente. Eliminar algunos óptimos locales provoca que la ventaja producida por el método neutral avanzado desaparezca. Con los resultados de este experimento no se pudo identificar el motivo de este cambio en el comportamiento de las series neutrales frente a una modificación en el paisaje de aptitud.

El resultado obtenido al utilizar la configuración con agente condicionado es bueno ya que para varias de las combinaciones de probabilidades de cruza y mutación seleccionadas; todas las corridas ejecutadas encontraron una solución óptima. Este resultado indica que para dichas combinaciones sólo se necesita una corrida para asegurar que se obtiene un programa de control óptimo.

Cabe mencionar que son necesarios experimentos posteriores con un mayor número de corridas por combinación para validar esta afirmación. Sin embargo, aún usando conjuntos de corridas pequeños nadie ha reportado un resultado como el presentado en esta tesis para el problema en cuestión usando la PG como heurística de solución. La Tabla 7.2 compara los mejores valores de esfuerzo encontrados con cada una de las dos configuraciones contra los diferentes resultados reportados en la literatura para el problema de la Hormiga Artificial con el trazo Santa Fe.

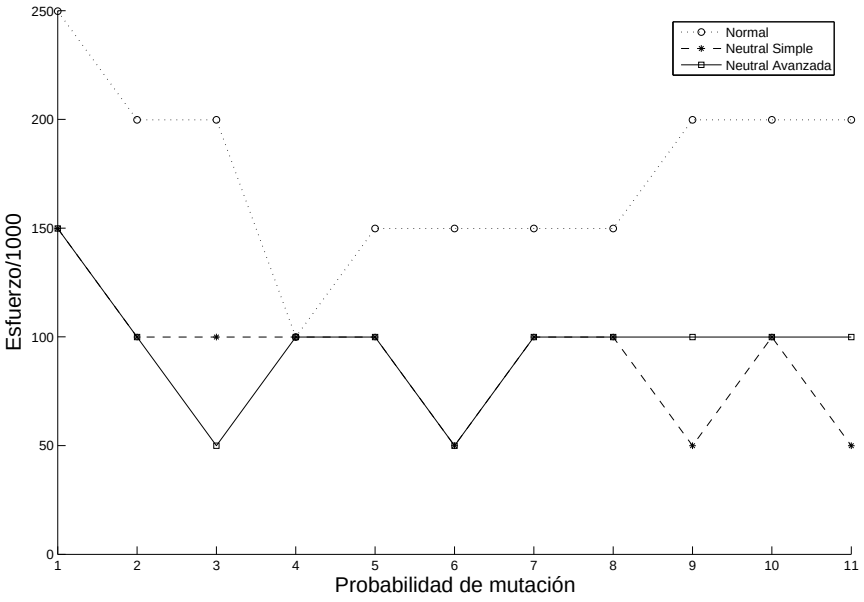

Figura 7.10: *Comparación de esfuerzo/1000 con 85 % de probabilidad de cruza para configuración con agente condicionado.*

Los mapas comparativos de promedio de aptitud del mejor individuo que se presentaron en la sección anterior muestran resultados diferentes para la configuración base y la configuración con agente condicionado.

Las dos figuras de la configuración base muestran mapas casi homogéneos. En dichos mapas se observa que las series neutrales produjeron promedios de aptitud del mejor individuo ligeramente menores a aquellos producidos por la serie normal. Algunas de las celdas para probabilidades de mutación entre tres y ocho por ciento muestran tonos de gris más obscuros. Esta diferencia en el tono indica una menor distancia entre el promedio de aptitud del mejor individuo obtenido por las series neutrales y aquel obtenido por la serie normal. Estos resultados son coherentes con el comportamiento identificado para el esfuerzo.

Las diferentes figuras de comparación del promedio de aptitud del mejor individuo para la configuración base muestran un comportamiento parecido al observado con la métrica de esfuerzo. Por ejemplo compare la Figura B.16 con la Figura B.4 y la Figura B.18 con la Figura B.6. La explicación dada a la mejora del esfuerzo de las series neutrales sobre la normal es consistente con los resultados obtenidos para el promedio de aptitud del mejor individuo.

Cabe destacar que la desviación estándar de aptitud del mejor individuo obtenida para cada combinación por las series neutrales de la configuración base tiende a ser menor que aquella obtenida por la serie normal para la misma combinación y misma configuración.

Tabla 7.2: *Esfuerzo necesario para resolver el trazo Santa Fe.*

Método	**Esfuerzo/1000**	**Probabilidad acumulada de éxito** $P(M,i)$
PG de Koza [58]	450	48/111
Búsqueda aleatoria [65]	450	-/-
PDGP [65]	336	-/-
Escalado de Colinas [43]	188	8/50
CGP neutral [75]	173	32/100
PE restringida [12]	136	47/50
Limite de comida visible 1 [64]	120	19/50
Evolución Gramatical [79]	102	-/-
Búsqueda mediante árboles aleatorios [17]	**21**	-/-
Neutral simple, cruza 60 %, mutación 7 %	100	45/50
Neutral avanzado limitado a 1 sola posición del trazo, cruza 100 %, mutación 6 %	50	**50/50**

Los resultados son claros para la configuración base. Agregar la función neutral no sólo aumenta las posibilidades de obtener un programa de solución óptimo con un menor número de corridas sin requerir encontrar una probabilidad de mutación específica. Dicha adición también provoca que en caso de no encontrarse una solución óptima; aquella encontrada tenga una aptitud menor que la solución que se obtendría con el conjunto de funciones estándar del problema de la Hormiga Artificial.

Los resultados del promedio de aptitud del mejor individuo para la configuración con agente condicionado muestran algunas diferencias con los resultados de esfuerzo para la misma configuración. Cuando la probabilidad de mutación es nula las series neutrales obtienen promedios de aptitud mayores al obtenido con la serie normal. Sin embargo, se observó que con dicha probabilidad de mutación el esfuerzo requerido por las series neutrales es menor al requerido por la serie normal.

La diferencia de resultados entre estas dos métricas indica que cuando:

- Se limita la capacidad perceptiva del agente.
- La probabilidad de mutación es cero.
- Se incluye la función neutral.
- La PG no encuentra una solución óptima.

entonces la solución encontrada tendrá peor aptitud que la hallada por una corrida de PG bajo las mismas condiciones pero sin la función neutral.

Esto ocurre porque al simplificar el paisaje de aptitud del problema, la serie normal puede encontrar con mayor facilidad una solución óptima sin requerir del operador de mutación. En cambio, las series neutrales necesitan del operador de mutación para activar secciones de código que le permitan a la PG construir una mejor solución. En otras palabras, cuando la probabilidad de mutación es nula en la configuración con agente condicionado el usar la función neutral se vuelve un impedimento en la exploración realizada por la Programación Genética.

Es necesario analizar con mayor detenimiento los cambios provocados en el paisaje de aptitud al limitar la capacidad perceptiva del agente. Sin embargo, dicho análisis queda fuera de los objetivos planteados en la presente tesis. Por esta razón, los siguientes experimentos se enfocarán únicamente al efecto de la función neutral en el paisaje de aptitud del problema de la Hormiga Artificial con el trazo Santa Fe.

7.2. Conteo de nodos activos e inactivos

Con el experimento planteado en la sección 7.1 se demostró que agregar la función neutral al conjunto de funciones del problema de la Hormiga Artificial le permite a la Programación Genética mejorar su desempeño en el trazo Santa Fe.

En la misma sección se planteó que la mejora en el comportamiento de la PG se debe a las mutaciones y cruzas que ocurren sobre el código inactivo generado por la función neutral.

Para sustentar dicha afirmación el primer paso es comprobar si la función neutral efectivamente aumenta la cantidad de código inactivo.

El segundo experimento se planteó para comparar la cantidad promedio de código activo e inactivo generado por la Programación Genética en la presencia y ausencia de la función neutral dada una sola combinación de probabilidades de cruza y mutación.

7.2.1. Planteamiento

Para el segundo experimento se utilizaron los parámetros para la PG definidos en la Tabla 7.3. Se puede observar que dichos parámetros difieren poco de los presentados en la sección 7.1.2; salvo la siguiente modificación: solamente se seleccionó una combinación de las probabilidades de cruza y mutación ocupadas en el primer experimento.

El proceso para escoger la combinación a utilizar fue el siguiente:

1. Se sacaron promedios de las tres métricas utilizadas (esfuerzo, promedio de aptitud del mejor individuo y desviación estándar de aptitud del mejor individuo) sobre las tres series de ambas configuraciones de los resultados del primer experimento.

2. Se seleccionó la combinación que obtuvo el menor valor para las tres métricas.

Tabla 7.3: *Parámetros para la PG en segundo y tercer experimento.*

Parámetro	Valor
Tamaño de población	150
Número de generaciones	333
Selección	Torneo (con tamaño de 3 individuos).
Profundidad inicial	3
Profundidad final	8
Probabilidad de cruza	85 %.
Probabilidad de mutación (por nodo)	7 %.
Función de Aptitud	Comida remanente del trazo.

La probabilidad de cruza a utilizar es 85 %. La probabilidad de mutación a ocupar es siete por ciento.

Nodo activo es todo nodo del árbol que se ejecuta en al menos un momento de su evaluación; no importando si dicha ejecución tiene o no efecto en el comportamiento del agente.

Por el contrario, nodo inactivo es aquel que no cumple esta característica. Es decir, *nodo inactivo* es aquel nodo que no se ejecuta para ninguna posible evaluación del árbol. Observar que todo nodo inactivo es intrón pero que no todo intrón es nodo inactivo.

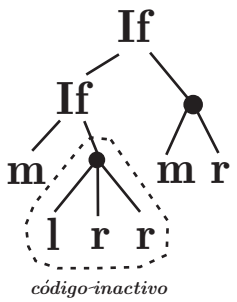

Figura 7.11: *Árbol con código inactivo.*

El *código activo* de un árbol es el número de nodos activos de éste. De igual manera, el *código inactivo* de un árbol es el número de nodos inactivos de éste. El *Total de código* de un árbol es el número de nodos de éste.

Para el caso particular del conjunto de funciones estándar del problema de la Hormiga Artificial sólo se puede tener código inactivo cuando se anidan dos funciones *IfFoodAhead*. Si se presenta dicha combinación entonces la rama izquierda o derecha del segundo *IfFoodAhead* (dependiendo en que rama de la primera función se anide la segunda ocurrencia) no se ejecutará para ninguna posible situación del agente. El árbol presentado en la Figura 7.11 tiene seis nodos de código activo y cuatro nodos de código inactivo. El árbol tiene un total de código igual a diez nodos.

Al adicionar la función *Neut_progn* se agrega un tipo especial de código inactivo, aquel que está controlado por el valor del selector tal como se explicó en la sección 6.2. Para diferenciarlo de dos *IfFoodAhead* anidados a este tipo de código inactivo se le llamará *código inactivo causado por la función neutral*.

En el presente experimento se realizaron tres series de corridas. En la primera de dichas series se utilizó el conjunto de funciones tradicional y se llama **serie normal**. La segunda incluye la función *Neut_progn* con el método simple de mutación (ver sección 6.3.2) y se llama **serie neutral simple**. La última incluye la función *Neut_progn* con el método neutral avanzado de mutación y se llama **serie neutral avanzada**.

En cada serie se realizaron 50 corridas independientes de la PG con los parámetros descritos en esta sección. Para cada generación se calculó el promedio de código activo, código inactivo, código inactivo por la función neutral y el promedio de aptitud de todos los individuos de todas las corridas. En la siguiente sección se presentan las ecuaciones utilizadas para calcular dichos valores.

7.2.2. Métricas

En este experimento se revisó lo que ocurre en el transcurso del proceso evolutivo de la Programación Genética. Por esta razón todas las métricas utilizadas están definidas con base en la variable t que indica el número de la generación que se desea vigilar.

Para calcular el promedio de código activo, inactivo e inactivo por la función neutral de todos los individuos de todas las corridas en la generación t se utilizó:

$$\overline{C}(t) = \frac{\sum_{j=1}^{N} \sum_{i=1}^{n} code(I_i^t)_j}{Nn} \tag{7.4}$$

donde n representa el tamaño de la población, N el número de corridas independientes realizadas, I_i^t corresponde al código del i-ésimo individuo en la generación t y $code(I_i^t)_j$ es una función que cuenta respectivamente el código activo, inactivo e inactivo por la función neutral del i-ésimo individuo en la generación t para la j-ésima corrida.

El promedio del total de código de todos los individuos de todas las corridas en la generación t se define como:

$$\overline{C_{Tot}}(t) = \overline{C_{Act}}(t) + \overline{C_{Ina}}(t) \tag{7.5}$$

donde $\overline{C_{Act}}(t)$ es el promedio de código activo de todos los individuos de todas las corridas en la generación t y $\overline{C_{Ina}}(t)$ es el promedio de código inactivo de todos los individuos de todas las corridas en la generación t.

Tener en cuenta que los nodos inactivos por la función neutral de un individuo también se cuentan como código inactivo de dicho individuo.

El promedio de aptitud de todos los individuos de todas las corridas en la generación t se calcula mediante (7.6). Antes de calcularlo es necesario encontrar el promedio de aptitud de todos los individuos $\overline{f_j}(t)$ para cada una de las corridas realizadas (7.7).

$$\overline{f_N}(t) = \frac{\sum_{j=1}^{N} \overline{f_j}(t)}{N} \tag{7.6}$$

$$\overline{f_j}(t) = \frac{\sum_{i=1}^{n} f_{I_i}^t}{n} \tag{7.7}$$

donde N representa el número de corridas independientes realizadas, n el tamaño de la población y $f_{I_i}^t$ corresponde al valor de aptitud del i-ésimo individuo en la generación t.

Adicionalmente, el promedio de la aptitud del mejor individuo se calculó para todas las generaciones ($0 \geq t \geq 333$) mediante la ecuación (7.3).

7.2.3. Resultados

Los resultados se presentan en tres figuras. Cada figura muestra los resultados obtenidos para una de las tres series del experimento (normal, neutral simple y neutral avanzada).

El eje x de cada figura corresponde con el número de la generación. En el eje izquierdo y se muestran los promedios de código activo, inactivo, inactivo por la función neutral (sólo para las series neutrales) y del total de código de todos los individuos de todas las corridas por generación. Sobre esta información se presenta el promedio de aptitud de todos los individuos de todas las corridas; siendo el eje derecho y' (en color rojo) el encargado de dar información sobre dicha métrica.

La Figura 7.12 presenta los resultados obtenidos en la serie normal.

Para tener una idea de cuánto podrían llegar a crecer los individuos se calcularon los límites máximos permitidos para el número de nodos por individuo con base en la profundidad inicial y la profundidad final. Dado que se cuenta con funciones de aridad dos y tres no es posible calcular límites exactos. Sin embargo, se pueden calcular rangos límite. Para la generación cero (profundidad inicial) el límite del número de nodos está en el rango $15 \geq l_{p_0} \geq 40$; mientras que el límite máximo de número de nodos (profundidad final) está en el rango $511 \geq l_{p_i} \geq 9841$ donde i es diferente de 0.

En la Figura 7.12 se puede observar que el total de código promedio alcanzó los 180 nodos alrededor de la generación 325. Aproximadamente 75 % de dicho código es activo. Nótese que el total de código activo promedio se encuentra bastante lejos del límite máxima de número de nodos. Esta diferencia indica que muchas de las corridas mantuvieron a los individuos lejos de la profundidad final máxima.

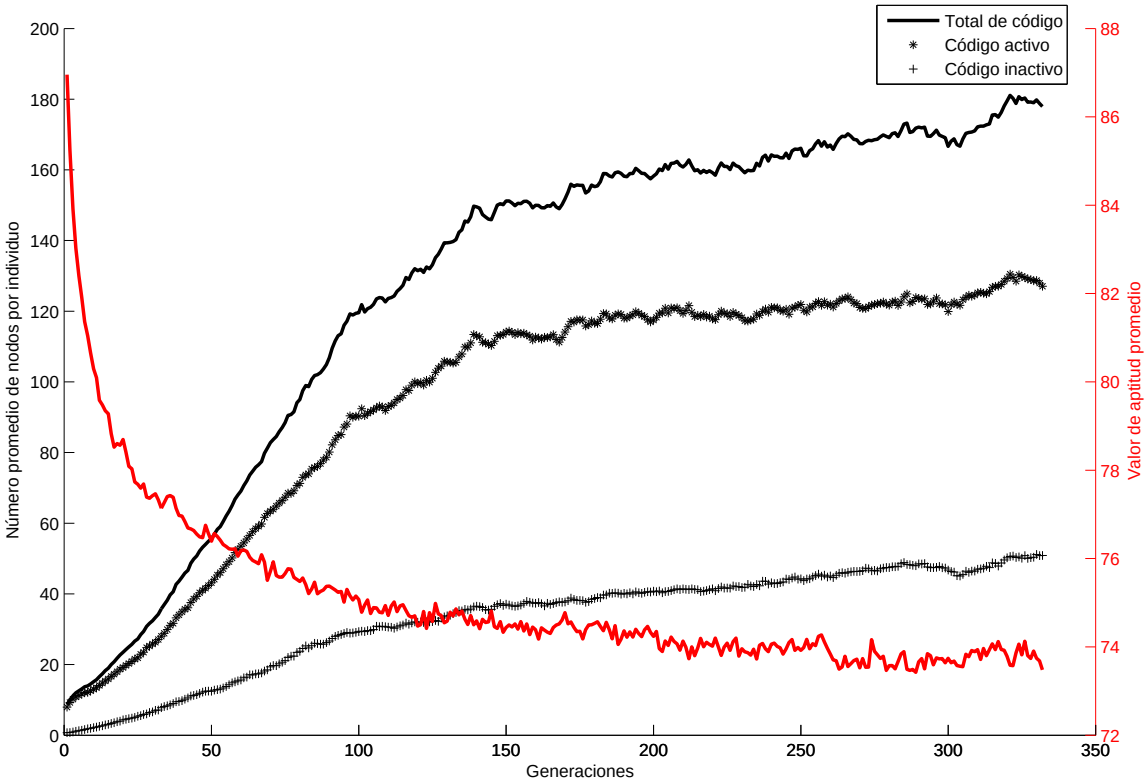

Figura 7.12: *Nodos activos e inactivos de la serie normal.*

El código inactivo pasa de tener poca o nula presencia en los individuos durante la generación cero a ocupar un 25 % del total de código de los individuos durante las últimas generaciones del proceso evolutivo. En las primeras 100 generaciones el código inactivo crece de manera proporcional al crecimiento presentado en el código activo. Tras este intervalo de tiempo mantiene una tasa de aumento constante hasta el final del proceso.

En las últimas generaciones el valor de aptitud promedio se encuentra alrededor de 73.5 puntos de aptitud. Este valor puede parecer muy alto. Sin embargo, se debe tener en cuenta que el tamaño del torneo seleccionado (tres individuos) ocasiona una baja presión selectiva en el método de selección por torneo. Debido a esto, no es de extrañarse que en general la población mantengan valores de aptitud altos aún cuando un pequeño grupo de ellos tengan un valor de aptitud cercano al óptimo.

El valor de aptitud promedio sigue un comportamiento parecido a un logaritmo invertido. Este comportamiento muestra que al final del proceso evolutivo la tasa de mejora es menor que al inicio. En otras palabras, los cambios (o saltos) que generan un beneficio son mas frecuentes al inicio del proceso evolutivo que al final de éste.

El proceso evolutivo representado en la Figura 7.12 se puede dividir en tres etapas:

1. En la primera de ellas (primeras 100 generaciones) se observa descenso en el valor de aptitud promedio y crecimiento en el tamaño de los individuos. En dicha etapa tanto el código activo como el inactivo crecen proporcionalmente. La explicación a este proceso es que la PG está logrando avanzar con facilidad en el paisaje de aptitud del trazo Santa Fe mediante el aumento del tamaño de los individuos.

2. Entre las generaciones 100 y 150 se identifica una segunda etapa en la que el código activo crece más que el inactivo. Sin embargo, la tasa de crecimiento tanto de código activo como inactivo es menor a la identificada durante las primeras generaciones. Es probable que varias de las corridas que no encontraron una solución óptima en la etapa anterior la encuentren en esta etapa. Sin embargo, la PG está comenzando a tener problemas para encontrar valores más bajos de aptitud y para propagar los mejores individuos a través de toda la población.

3. De la generación 150 al final del proceso evolutivo la cantidad de código activo oscila alrededor de los 110 nodos mientras que el código inactivo aumenta ligera y constantemente. El valor de aptitud promedio desciende únicamente 2 puntos de aptitud en el transcurso de 200 generaciones. En esta etapa posiblemente la PG se encuentre protegiendo las soluciones encontradas mediante la adición de código inactivo y tratando de propagar los mejores individuos a través de toda la población.

La Figura 7.13 presenta los resultados obtenidos en la serie neutral simple.

El primer elemento a destacar en dicha figura es la tenue presencia del código inactivo por la función neutral. Contrario a los resultados esperados, el promedio de código inactivo por la función neutral alcanza 15 nodos por individuo de un total de código de casi 200 nodos.

El comportamiento de la serie neutral se puede descomponer en dos etapas:

1. Durante las primeras 80 generaciones se identifica una etapa de crecimiento en el tamaño de los individuos y descenso en el valor de aptitud promedio. Durante esta etapa la PG sigue un proceso parecido al identificado en la primera etapa de la serie normal.

2. De la generación 80 al final del proceso evolutivo la cantidad de código activo continua creciendo. Dicho crecimiento se da con un ritmo menor al presentado durante la primera etapa pero con mayor celeridad que el aumento del código inactivo. Al igual que lo ocurrido durante la tercera etapa de la serie normal, el valor de actitud promedio desciende lentamente mediante un intrincado recorrido.

Se identificaron una serie de diferencias entre el comportamiento de la serie normal y el comportamiento de la serie neutral simple:

- La etapa de descenso en el valor de aptitud promedio ocupa menos generaciones en la serie neutral simple que en la serie normal.

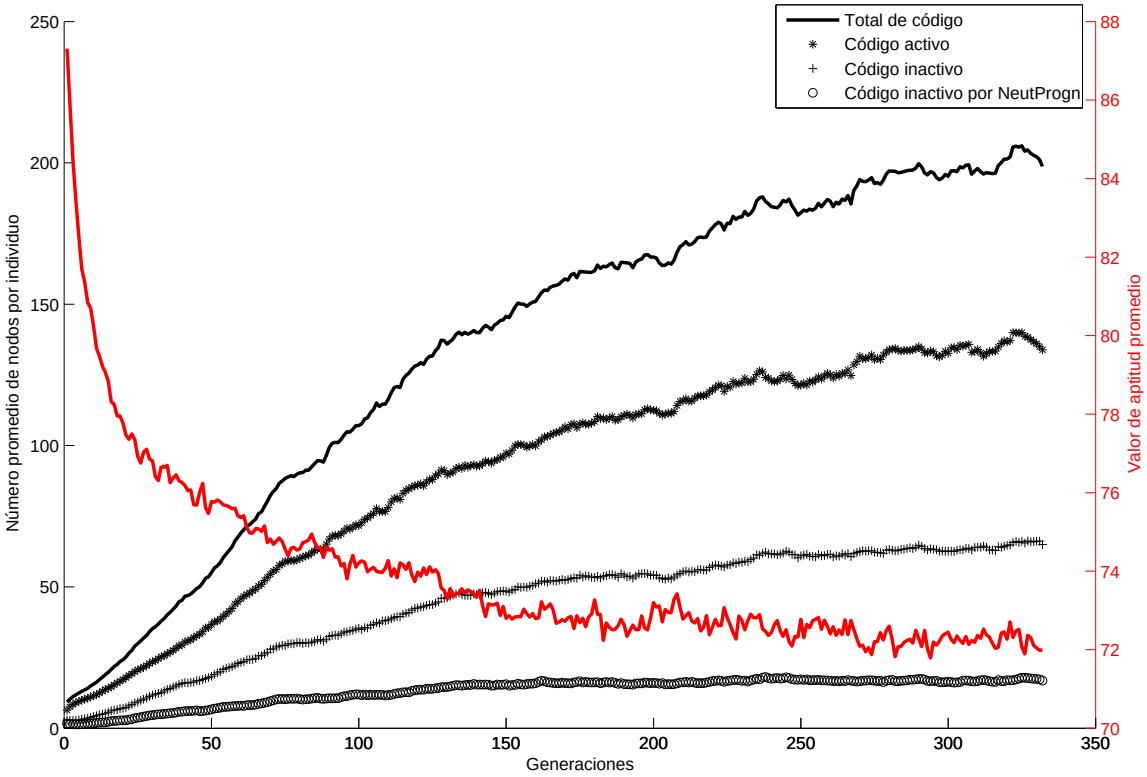

Figura 7.13: *Nodos activos, inactivos y neutrales de la serie neutral simple.*

- El valor promedio de aptitud alcanzado en la última generación es menor en la serie neutral simple que en la serie normal.
- Para la serie neutral simple en ningún momento la cantidad de código activo tiende a descender o a mantener un mismo valor.
- Al final de la primera etapa en la serie neutral simple la cantidad de código inactivo ocupa el 50 % del total de código. Esta proporción de código inactivo no se presenta en ningún momento en la serie normal.

A pesar de la escasa presencia de código inactivo por la función neutral se observaron diferencias entre la serie normal y la serie neutral simple. El continuo aumento de la cantidad de código activo se puede explicar por el efecto de los operadores de cruza y mutación sobre las secciones de código inactivo.

Los resultados de la Figura 7.13 no muestran un aumento significativo del código inactivo cuando se agrega la función neutral al conjunto de funciones de la Hormiga Artificial y se

usa el método simple de mutación. Tampoco explican la mejora observada en el experimento anterior al comparar la serie neutral simple contra la serie normal.

Sería conveniente medir la cantidad de código activo, inactivo e inactivo por la función neutral para corridas con diferentes probabilidades de cruza y mutación con el método de mutación simple para la función neutral. El análisis de estas nuevas corridas permitirá determinar si el comportamiento detectado en la Figura 7.13 es constante para toda combinación de probabilidades de cruza y mutación.

La Figura 7.14 presenta los resultados obtenidos en la serie neutral avanzada.

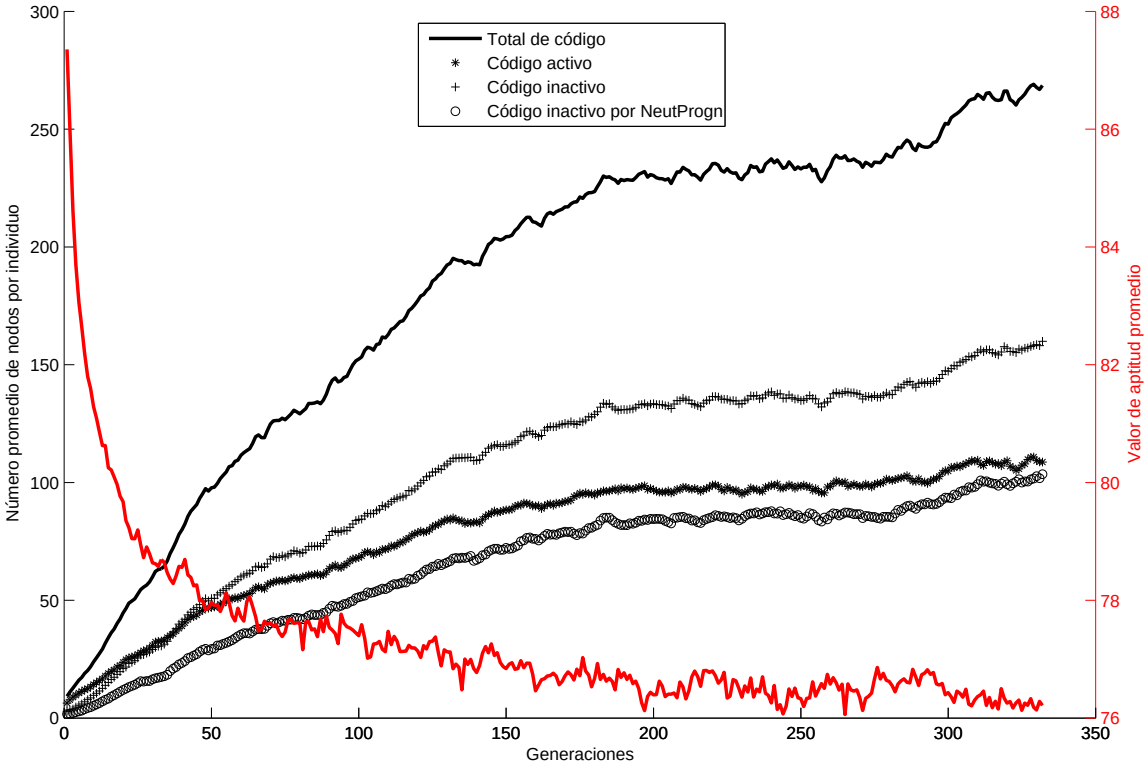

Figura 7.14: *Nodos activos, inactivos y neutrales de la serie neutral avanzada.*

Esta figura muestra un comportamiento diferente para la serie neutral avanzada que el observado para las series normal y neutral simple. La presencia de código inactivo por la función neutral aumenta considerablemente con respecto a la serie neutral simple.

Hasta la generación 50 la cantidad de código activo se mantuvo creciendo como lo hacia en las dos series anteriores. Sin embargo, después de dicha generación la tasa de crecimiento del código activo bajó. En la misma generación la cantidad de código inactivo por la función

neutral comienza a aumentar a tal grado que en las últimas generaciones cubre casi el mismo número de nodos que el código activo.

La cantidad total de código también aumentó con respecto al valor obtenido en las dos series anteriores. Dicho aumento corresponde a la cantidad de código inactivo por la función neutral.

El valor de aptitud promedio desciende durante las primeras 50 generaciones como lo hace para las series anteriores. Posteriormente dicho valor continua descendiendo zigzagueantemente con una tasa menor. A pesar de que el valor de aptitud promedio alcanzado es mayor que el obtenido por las series normal y neutral simple, los resultados obtenidos para el primer experimento indican que tanto el promedio de aptitud del mejor individuo como el esfuerzo de la serie neutral avanzada son menores que los equivalentes de las series normal y neutral simple.

Si se observa la cantidad de código inactivo con detenimiento se descubrirá que gran parte de éste está provocado por la función neutral. De hecho si se resta el código inactivo por la función neutral al código inactivo total se tendrá que la cantidad obtenida es un valor semejante al código inactivo presente en la serie normal. Esto indica que las diferencias observadas en el comportamiento de la serie neutral avanzada con respecto a la serie normal se deben a la inclusión de la función neutral y al uso del método avanzado de mutación.

7.2.4. Discusión

Los resultados sobre la serie neutral avanzada llevan a la conclusión de que la función neutral con el método avanzado de mutación modifican el funcionamiento de la Programación Genética sobre el problema de la Hormiga Artificial con el trazo Santa Fe. Dicha función aumenta la cantidad de código inactivo promedio de los individuos.

La Figura 7.15 muestra el promedio de aptitud del mejor individuo obtenido por las tres series del presente experimento para todas las generaciones.

El aumento de código inactivo promedio causado por la función neutral con el método avanzado de mutación parece no tener un efecto positivo para el promedio de aptitud de toda la población. Sin embargo, tal como se observa en la Figura 7.15 dicha función provoca que el promedio de aptitud del mejor individuo sea menor con respecto al valor obtenido para el conjunto de funciones normalmente usado en el problema de la Hormiga Artificial.

En las figuras de la sección anterior se puede observar que las tres series mantuvieron un número total de nodos promedio por debajo del límite máximo de número de nodos. La serie neutral avanzada fue la que obtuvo el total de código más alto de las tres series. Dicho valor apenas se aproxima en su punto más alto a la mitad del rango más bajo definido para l_{p_i}. Dado que ninguna de las tres series alcanza el límite máximo de número de nodos, el crecimiento de los individuos no se puede considerar como hinchamiento o el número de generaciones no fue lo suficientemente grande para que l_{p_i} se alcanzara.

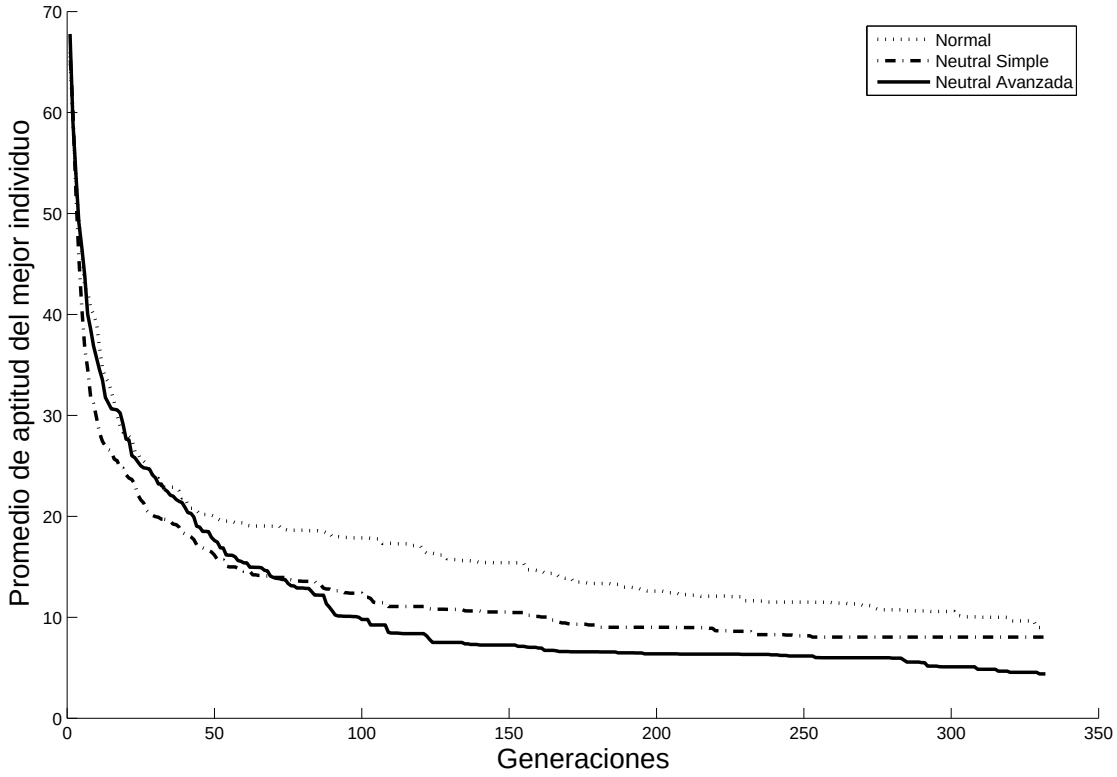

Figura 7.15: *Promedio de aptitud del mejor individuo para todas las series.*

Queda claro que la función neutral tiene poco efecto en el proceso de la Programación Genética si no se ocupa el método de mutación avanzado. El proceso de activación y desactivación mediante mutación de *Neut_progn* a *Progn3* y viceversa es más efectivo que la mutación del selector o el cruzamiento de una rama inactiva de *Neut_progn* con una rama activa. Esta afirmación tiene sustento en las diferencias presentadas entre las series neutral simple y neutral avanzada.

El presente experimento no muestra claramente la forma en la que la función neutral con el método de mutación avanzada mejora el desempeño de la Programación Genética para el problema de la Hormiga Artificial con el trazo Santa Fe. Por esta razón se planteó un tercer experimento en el que se busca determinar cómo opera una sola corrida de la PG con dichas características.

7.3. Conteo de funciones

Con el experimento planteado en la sección 7.2 se demostró que la serie neutral avanzada tiene un mayor promedio de código inactivo de los individuos. También se observó que dicha serie no mejora la aptitud promedio de toda la población. Sin embargo, los resultados del experimento planteado en la sección 7.1 indican que la serie neutral avanzada presenta un promedio de aptitud del mejor individuo y un esfuerzo menores a los obtenidos por las otras dos series.

En el tercer experimento se busca identificar con mayor grado de detalle lo que ocurre en el transcurso de una corrida de la Programación Genética cuando el conjunto de funciones incluye a la función neutral y se usa el método avanzado para resolver la mutación de dicha función.

7.3.1. Planteamiento

Para el tercer experimento se utilizaron los parámetros para la PG definidos en la Tabla 7.3. Solamente se realizó una corrida incluyendo la función *Neut_progn* en el conjunto de funciones del problema de la Hormiga Artificial y resolviendo la mutación de dicha función mediante el método avanzado (ver sección 6.3.2).

En cada generación se calculó el promedio sobre todos los individuos de código activo, inactivo, inactivo por la función neutral y total. También se calculó la cantidad promedio de nodos para cada elemento del conjunto de funciones utilizado sobre todos los individuos. Además se contabilizaron el promedio de aptitud de todos los individuos y la aptitud del mejor individuo de la generación.

Adicionalmente, se almacenó en un archivo temporal el código de los mejores 15 individuos de cada generación.

7.3.2. Métricas

En cada generación t se calculó el promedio de código activo, inactivo e inactivo por la función neutral de todos los individuos mediante:

$$C(t) = \frac{\sum_{i=1}^{n} code(I_i^t)}{n} \tag{7.8}$$

donde n representa el tamaño de la población, I_i^t corresponde al código del i-ésimo individuo en la generación t y $code(I_i^t)$ es una función que cuenta respectivamente el código activo, inactivo e inactivo por la función neutral del i-ésimo individuo en la generación t.

De manera semejante se calculó la cantidad promedio de nodos de la generación t para cada elemento del conjunto de funciones utilizado (*IfFoodAhead*, *Progn2*, *Progn3*, *Neut_Progn*) tal como se presenta en (7.9).

$$F(t) = \frac{\sum_{i=1}^{n} code_{func}(I_i^t)}{n} \tag{7.9}$$

donde n representa el tamaño de la población, I_i^t corresponde al código del i-ésimo individuo en la generación t y $code_{func}(I_i^t)$ es una función que cuenta el número de nodos para cada elemento del conjunto de funciones del i-ésimo individuo en la generación t.

El promedio del total de código de todos los individuos en la generación t se define como:

$$C_{Tot}(t) = C_{Act}(t) + C_{Ina}(t) \tag{7.10}$$

donde $C_{Act}(t)$ es el promedio de código activo de todos los individuos en la generación t y $C_{Ina}(t)$ es el promedio de código inactivo de todos los individuos en la generación t.

Para calcular el promedio de aptitud de todos los individuos $\overline{f}(t)$ se utilizó la ecuación (7.7). Mientras que la aptitud del mejor individuo en la generación t se computó mediante:

$$f_{min}(t) = min(f_{I_1}^t, f_{I_2}^t, \ldots f_{I_n}^t) \tag{7.11}$$

donde n representa el tamaño de la población y $f_{I_i}^t$ corresponde al valor de aptitud del i-ésimo individuo en la generación t.

7.3.3. Resultados

Los resultados se presentan en dos figuras.

La Figura 7.16 muestra los conteos de nodos activos e inactivos de la misma manera que se utilizó en las figuras de la sección 7.2.3.

El eje x corresponde con el número de la generación. En el eje izquierdo y se muestran los promedios de código activo, inactivo, inactivo por la función neutral y del total de código de todos los individuos por generación. Sobre esta información se presenta el promedio de aptitud de todos los individuos por generación; siendo el eje derecho y' (en color rojo) el encargado de dar información sobre dicha métrica.

Al tratarse de la información recopilada para una sola corrida en lugar del promedio de la información de múltiples corridas, dicha figura presenta algunas diferencias con respecto a la Figura 7.14:

- El promedio de aptitud de todos los individuos muestra un comportamiento más desordenado en el tercer experimento que el presentado para el segundo experimento. Se

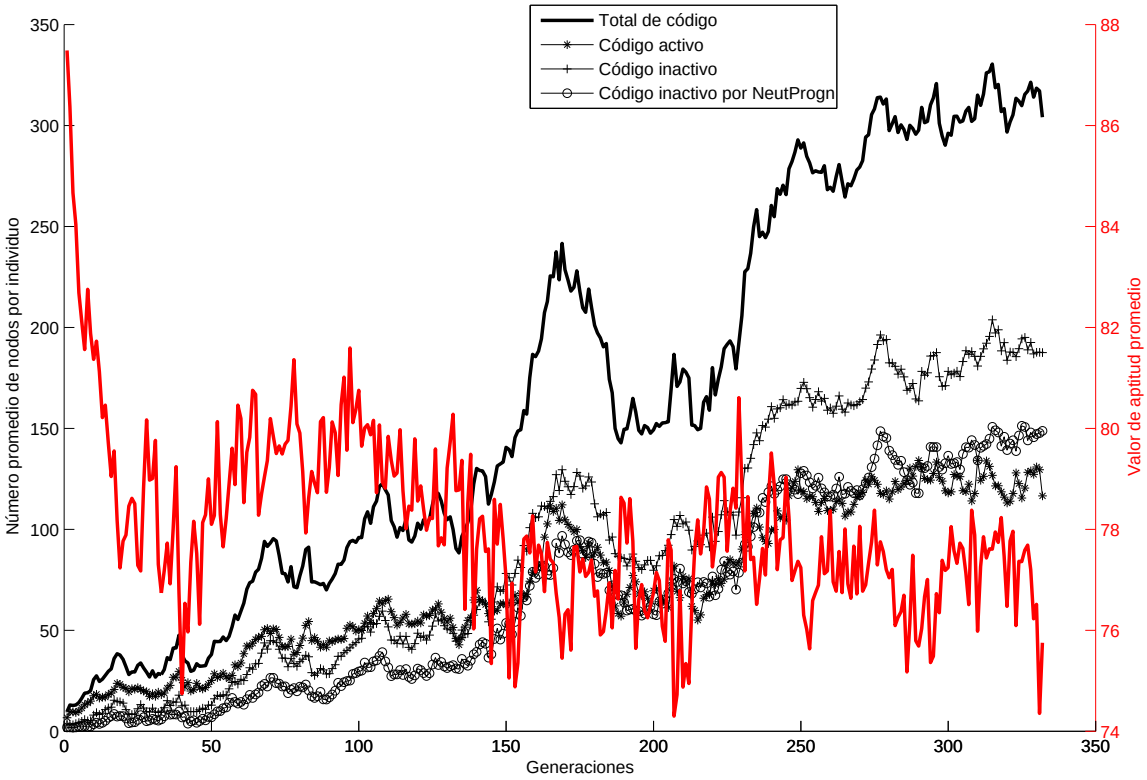

Figura 7.16: *Nodos activos e inactivos del tercer experimento.*

identifica una etapa temprana de descenso claro durante las primeras 20 generaciones. Tras dicha etapa el promedio de aptitud de todos los individuos presenta un comportamiento oscilante con mucho ruido.

- El crecimiento continuo de los promedios de código identificado en la Figura 7.14 no se presenta en la Figura 7.16. En su lugar se identifican etapas de crecimiento excesivo de los individuos seguidas de etapas de reducción de la cantidad de nodos.

- La cantidad de código inactivo que no es provocado por la función neutral es menor en el tercer experimento con respecto a lo observado en la serie neutral avanzada del segundo experimento.

- El total de código alcanzado en la corrida del tercer experimento está aproximadamente 70 unidades arriba del total de código promedio de 50 corridas del segundo experimento. El tamaño de total de código del segundo experimento se rebasa en el tercer experimento alrededor de la generación 225 cuando inicia un proceso de crecimiento que continua hasta la última generación.

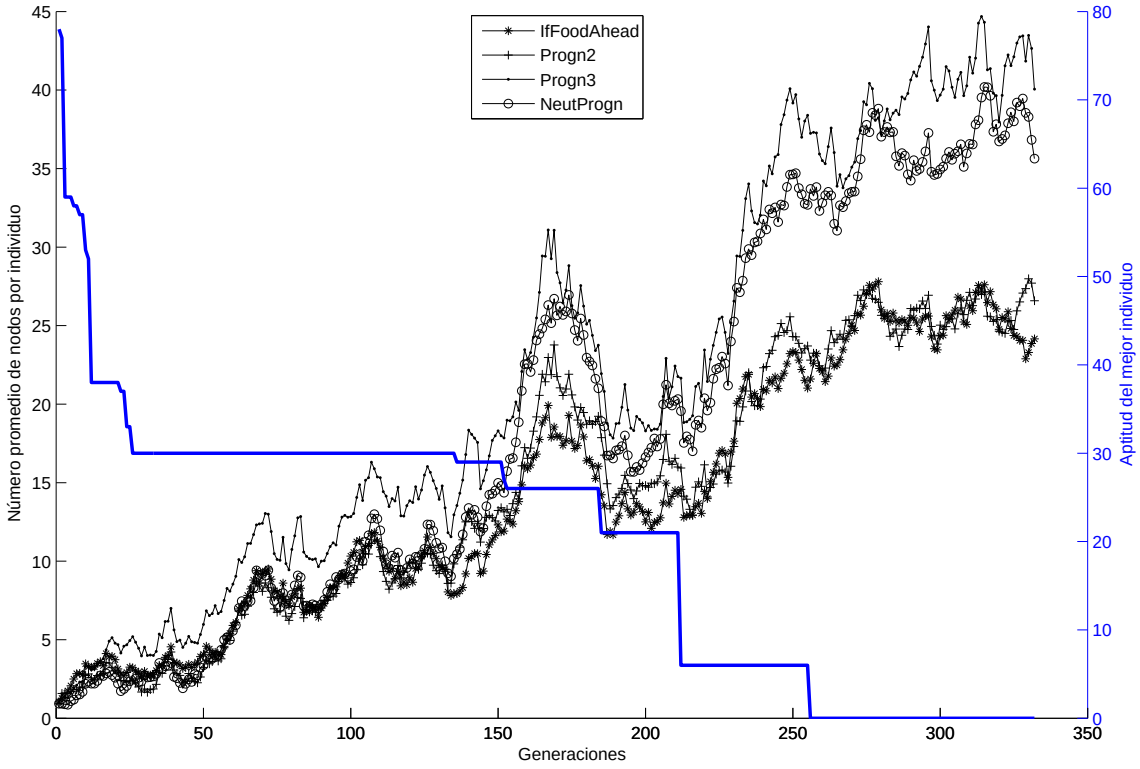

Figura 7.17: *Conteo de funciones para el tercer experimento.*

La Figura 7.17 muestra los conteos por función.

En dicha figura el eje x corresponde con el número de la generación. En el eje y se muestran los promedios de nodos para cada función (*IfFoodAhead*, *Progn2*, *Progn3* y *Neut_progn*). Sobre esta información se presenta la aptitud del mejor individuo por generación; siendo el eje derecho y' (en color azul) el encargado de dar información sobre dicha métrica.

Se observa que la cantidad promedio de nodos de la función neutral guarda una relación con la cantidad promedio de nodos de *Progn3*. Cuando este último valor se dispara, también lo hace la cantidad promedio de nodos de *Neut_progn*. El mismo tipo de relación se identifica entre *IfFoodAhead* y *Progn2*.

Estas relaciones entre funciones con la misma aridad posiblemente estén vinculadas con el operador de mutación.

A pesar que hacia el final de la corrida se presenta un claro aumento en la proporción de las funciones *Progn3* y *Neut_progn* con respecto a las funciones *IfFoodAhead* y *Progn2*; en ningún momento el proceso de la Programación Genética tiende a eliminar la presencia de alguno de los elementos del conjunto de funciones.

La aptitud del mejor individuo por generación da una idea sobre el mecanismo de funcionamiento de la PG para la solución al problema de la Hormiga Artificial con el trazo Santa Fe.

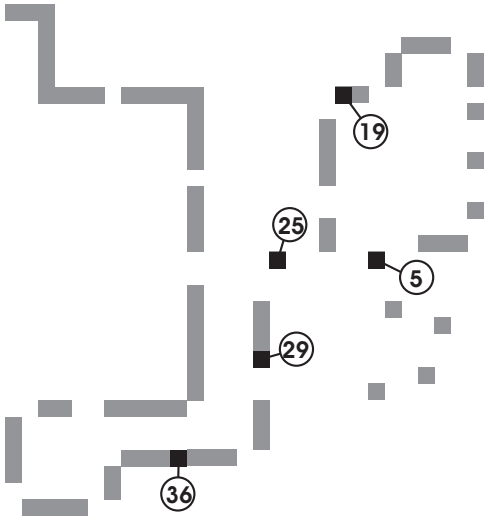

Figura 7.18: *Posiciones de comida difíciles de alcanzar del trazo Santa Fe.*

Se puede encontrar una correspondencia entre los valores de aptitud que le costaron mejorar a la Programación Genética y ciertos *gaps* del trazo Santa Fe. La figura 7.18 muestra en color negro las posiciones de comida del trazo Santa Fe en las que la corrida de este experimento presentó problemas; indicando en un círculo el valor de aptitud estandarizada que corresponde a dichas posiciones de comida.

La dificultad para resolver dichos *gaps* es específica de la corrida efectuada. Posiblemente otra ejecución independiente presentaría dificultades para resolver otras secciones del trazo.

7.3.4. Discusión

A continuación se discutirán las observaciones realizadas a los resultados del tercer experimento y se explicará el proceso seguido por la Programación Genética en la corrida ejecutada. Dicha explicación permitirá esclarecer la diferencia en el comportamiento de la serie neutral avanzada del segundo experimento con respecto a las otras dos series del mismo.

La baja presión selectiva en la población reduce la posibilidad de tomar en la generación $t+1$ una gran cantidad copias de los individuos con mejores valores de aptitud de la generación

t. Este mecanismo le permite a la Programación Genética salir con mayor facilidad de óptimos locales. Sin embargo, si no se controla de la manera adecuada puede provocar la pérdida de los mejores individuos de cada generación.

En la sección anterior, se observó que el valor de aptitud promedio sube y baja constantemente durante el transcurso de las generaciones posteriores a las primeras 20. Este comportamiento se debe a que el tamaño del torneo utilizado para el método de selección por torneo provoca que el proceso evolutivo tenga baja presión selectiva.

Se decidió escoger 3 individuos como tamaño de torneo para evitar estancar el proceso de búsqueda de programas solución en óptimos locales. Tras revisar los mejores 15 individuos de cada generación se comprobó que en ocasiones algunos individuos con un buen valor de aptitud se perdían en el transcurso de una generación a la siguiente. Esta pérdida es causada por el tamaño de torneo seleccionado y provoca la oscilación del valor de aptitud promedio.

Por ello, durante todas las generaciones se cumple que la mayor parte de los individuos de la población tienen un valor de aptitud alto. Así pues al momento de que la Programación Genética incrementa la cantidad de código inactivo no está buscando proteger a los individuos de cruzas o mutaciones destructivas; sino más bien lo implementa como un mecanismo para generar *reservas génicas* en busca de la construcción de mejores soluciones.

Este proceso de creación constante de reservas génicas inactivas junto con la baja presión selectiva le permiten a la Programación Genética explorar nuevas rutas al encontrarse estancado en un óptimo local. Un ejemplo de dicha capacidad es lo sucedido en la corrida analizada.

La Programación Genética avanzó con facilidad durante las primeras 25 generaciones hasta llegar a un valor relativamente bajo de 30 puntos de aptitud. El siguiente programa de control representa al individuo con dicho valor de aptitud encontrado en la generación 25:

```
(IF_FOOD_AHEAD (MOVE)(PROGN3 (IF_FOOD_AHEAD (RIGHT)(RIGHT))(MOVE)(PROGN3 (PROGN3 (PROGN2
(MOVE)(NEUT (1)(PROGN3 (RIGHT)(MOVE)(MOVE))(LEFT)))(PROGN3 (PROGN2 (IF_FOOD_AHEAD
(MOVE)(RIGHT))(IF_FOOD_AHEAD (RIGHT)(RIGHT)))(LEFT)(IF_FOOD_AHEAD (PROGN3 (MOVE)(MOVE)(LEFT))(PROGN3
(RIGHT)(MOVE)(RIGHT))))(IF_FOOD_AHEAD (PROGN2 (PROGN2 (RIGHT)(LEFT))(NEUT (3)(MOVE)(LEFT)))(PROGN3
(MOVE)(NEUT (3)(LEFT)(LEFT))(RIGHT))))(PROGN2 (MOVE)(MOVE))(MOVE))))
```

Sin embargo, dicho individuo corresponde a un óptimo local ya que al recorrer el trazo Santa Fe gira a la derecha en el *gap* que se encuentra antes de la casilla indicada con el valor 29 en la Figura 7.18.

En ese punto del proceso evolutivo una gran parte de la población estaba ocupada por copias de los siguientes individuos con valores de aptitud 33, 38 y 38, respectivamente:

```
(IF_FOOD_AHEAD (MOVE)(PROGN3 (LEFT)(MOVE)(PROGN3 (MOVE)(RIGHT)(MOVE))))
```

```
(IF_FOOD_AHEAD (NEUT (1)(IF_FOOD_AHEAD (MOVE)(RIGHT))(PROGN3 (RIGHT)(MOVE)(PROGN3
(LEFT)(LEFT)(MOVE))))(PROGN3 (RIGHT)(MOVE)(PROGN3 (MOVE)(LEFT)(MOVE))))
```

```
(IF_FOOD_AHEAD (MOVE)(PROGN3 (RIGHT)(MOVE)(PROGN3 (MOVE)(LEFT)(MOVE))))
```

Esta situación se puede identificar como un caso en el que la Programación Evolutiva se encuentra estancada en un óptimo local. La forma en la que la PG logró salir de este punto fue mediante la generación de nuevas soluciones a través de la cruza y mutación en secciones inactivas de código, algunas de ellas generadas por la función neutral.

Durante dichas etapas de exploración neutral se puede identificar aumento en el total de código promedio de los individuos.

En el transcurso de las siguientes 110 generaciones la baja presión selectiva permitió eliminar algunas copias de estos individuos mientras que la cruza y mutación sobre las secciones de código inactivo lograron construir un individuo que resolvía el *gap* en cuestión. Dicho individuo obtuvo un valor de aptitud de 29 y se representa mediante el programa de control:

```
(IF_FOOD_AHEAD (PROGN3 (PROGN2 (MOVE)(NEUT (0)(NEUT (0)(MOVE)(NEUT (0)(LEFT)(RIGHT)))(NEUT
(1)(MOVE)(MOVE))))(NEUT (1)(PROGN3 (MOVE)(PROGN3 (PROGN2 (PROGN2 (LEFT)(PROGN2
(RIGHT)(MOVE)))(MOVE))(LEFT)(PROGN3 (RIGHT)(LEFT)(LEFT)))(PROGN3 (MOVE)(MOVE)(RIGHT)))(PROGN3
(PROGN3 (RIGHT)(PROGN2 (IF_FOOD_AHEAD (PROGN3 (MOVE)(MOVE)(RIGHT))(NEUT
(1)(LEFT)(RIGHT)))(MOVE))(PROGN2 (PROGN2 (PROGN2 (RIGHT)(LEFT))(PROGN3 (LEFT)(LEFT)(MOVE)))(NEUT
(1)(MOVE)(LEFT))))(NEUT (2)(LEFT)(PROGN2 (NEUT (2)(MOVE)(LEFT))(PROGN3 (RIGHT)(LEFT)(MOVE))))(PROGN2
(PROGN2 (PROGN2 (LEFT)(LEFT))(PROGN3 (RIGHT)(MOVE)(RIGHT)))(NEUT (1)(PROGN3 (PROGN3
(LEFT)(MOVE)(RIGHT))(RIGHT)(MOVE))(LEFT)))))(PROGN3 (IF_FOOD_AHEAD (RIGHT)(MOVE))(MOVE)(NEUT
(1)(PROGN2 (RIGHT)(MOVE))(LEFT))))(PROGN3 (PROGN2 (PROGN2 (RIGHT)(MOVE))(IF_FOOD_AHEAD
(MOVE)(LEFT)))(MOVE)(IF_FOOD_AHEAD (MOVE)(NEUT (1)(MOVE)(LEFT)))))
```

Es en este proceso de creación de reservas génicas para la generación de mejores individuos que la función neutral con el método avanzado de mutación le permite a la Programación Genética salir con mayor facilidad de óptimos locales. Esta afirmación sólo se pudo comprobar para el problema de la Hormiga Artificial con el trazo Santa Fe y dados los parámetros definidos en la Tabla 7.3.

El proceso de búsqueda de mejores soluciones continua funcionando en la corrida analizada incluso en las últimas generaciones. Por ejemplo en la generación 330 se cuenta con cuatro individuos óptimos representados por los siguientes programas de control:

```
(IF_FOOD_AHEAD (NEUT (1)(NEUT (1)(PROGN3 (PROGN2 (PROGN2 (RIGHT)(RIGHT))(IF_FOOD_AHEAD
(MOVE)(RIGHT)))(PROGN3 (PROGN2 (PROGN2 (RIGHT)(PROGN2 (RIGHT)(RIGHT)))(RIGHT))(RIGHT)(NEUT
(1)(LEFT)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD (LEFT)(LEFT))(LEFT)(RIGHT)))(NEUT (1)(IF_FOOD_AHEAD
(NEUT (1)(PROGN3 (PROGN3 (LEFT)(MOVE)(RIGHT))(PROGN2 (RIGHT)(RIGHT))(LEFT))(LEFT))(PROGN3
(IF_FOOD_AHEAD (IF_FOOD_AHEAD (MOVE)(NEUT (1)(LEFT)(LEFT)))(PROGN2 (IF_FOOD_AHEAD
(LEFT)(MOVE))(IF_FOOD_AHEAD (RIGHT)(RIGHT))))(PROGN3 (LEFT)(IF_FOOD_AHEAD (NEUT
(0)(LEFT)(RIGHT))(MOVE))(PROGN3 (LEFT)(RIGHT)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD
(MOVE)(PROGN3 (LEFT)(MOVE)(RIGHT)))(MOVE)(IF_FOOD_AHEAD (MOVE)(LEFT))))))(NEUT (3)(PROGN3
(NEUT (3)(NEUT (2)(LEFT)(LEFT))(LEFT))(PROGN3 (PROGN2 (IF_FOOD_AHEAD (LEFT)(PROGN2
(MOVE)(MOVE)))(LEFT))(MOVE)(PROGN3 (RIGHT)(LEFT)(MOVE)))(NEUT (2)(MOVE)(RIGHT)))(PROGN3
(IF_FOOD_AHEAD (PROGN3 (IF_FOOD_AHEAD (PROGN2 (RIGHT)(MOVE))(NEUT (1)(RIGHT)(RIGHT)))(MOVE)(PROGN3
(PROGN3 (RIGHT)(LEFT)(MOVE))(PROGN2 (RIGHT)(RIGHT))(MOVE)))(MOVE))(PROGN3 (IF_FOOD_AHEAD
(MOVE)(NEUT (3)(IF_FOOD_AHEAD (MOVE)(MOVE))(IF_FOOD_AHEAD (RIGHT)(MOVE))))(LEFT)(IF_FOOD_AHEAD
(NEUT (2)(MOVE)(RIGHT))(NEUT (2)(LEFT)(LEFT))))(IF_FOOD_AHEAD (PROGN2 (PROGN2 (MOVE)(RIGHT))(PROGN3
(PROGN3 (RIGHT)(MOVE)(RIGHT))(LEFT)(MOVE)))(NEUT (2)(NEUT (2)(MOVE)(MOVE))(LEFT))))))(PROGN3 (PROGN2
(PROGN2 (RIGHT)(RIGHT))(IF_FOOD_AHEAD (MOVE)(RIGHT)))(MOVE)(RIGHT)))
```

```
(IF_FOOD_AHEAD (NEUT (1)(NEUT (1)(PROGN3 (MOVE)(PROGN3 (PROGN2 (PROGN2 (RIGHT)(PROGN2
(RIGHT)(MOVE)))(RIGHT))(RIGHT)(NEUT (2)(LEFT)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD
(MOVE)(LEFT))(MOVE)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD (PROGN3 (LEFT)(IF_FOOD_AHEAD
(RIGHT)(PROGN2 (RIGHT)(MOVE)))(IF_FOOD_AHEAD (NEUT (0)(RIGHT)(MOVE))(IF_FOOD_AHEAD
(LEFT)(LEFT))))(RIGHT))(IF_FOOD_AHEAD (NEUT (1)(NEUT (2)(IF_FOOD_AHEAD
(RIGHT)(MOVE))(LEFT))(LEFT))(LEFT))(NEUT (3)(NEUT (2)(IF_FOOD_AHEAD (NEUT
(3)(LEFT)(RIGHT))(MOVE))(PROGN3 (LEFT)(LEFT)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD (RIGHT)(NEUT
(0)(LEFT)(MOVE)))(RIGHT)(IF_FOOD_AHEAD (MOVE)(LEFT))))))(NEUT (3)(PROGN3 (NEUT
(3)(NEUT (2)(RIGHT)(LEFT))(LEFT))(PROGN3 (IF_FOOD_AHEAD (PROGN2 (LEFT)(PROGN2
(MOVE)(MOVE)))(RIGHT))(LEFT)(PROGN3 (MOVE)(LEFT)(MOVE)))(PROGN3 (PROGN3 (PROGN2
(LEFT)(PROGN3 (MOVE)(MOVE)(MOVE)))(LEFT)(MOVE))(MOVE)(RIGHT)))(LEFT)))(PROGN3 (PROGN2 (PROGN2
(RIGHT)(RIGHT))(IF_FOOD_AHEAD (MOVE)(RIGHT)))(MOVE)(RIGHT)))
```

```
(IF_FOOD_AHEAD (NEUT (1)(NEUT (1)(PROGN3 (PROGN2 (PROGN2 (RIGHT)(RIGHT))(IF_FOOD_AHEAD
(MOVE)(RIGHT)))(PROGN3 (PROGN2 (PROGN2 (RIGHT)(PROGN2 (RIGHT)(RIGHT)))(RIGHT))(RIGHT)(NEUT
(1)(LEFT)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD (LEFT)(LEFT))(LEFT)(RIGHT)))(NEUT (1)(IF_FOOD_AHEAD
(NEUT (1)(NEUT (1)(PROGN2 (RIGHT)(LEFT))(LEFT))(LEFT))(RIGHT))(PROGN3 (IF_FOOD_AHEAD
(IF_FOOD_AHEAD (MOVE)(NEUT (1)(LEFT)(LEFT)))(PROGN2 (IF_FOOD_AHEAD (LEFT)(MOVE))(IF_FOOD_AHEAD
(RIGHT)(RIGHT))))(PROGN3 (LEFT)(IF_FOOD_AHEAD (NEUT (0)(LEFT)(RIGHT))(MOVE))(PROGN3
(LEFT)(RIGHT)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD (MOVE)(PROGN3 (LEFT)(MOVE)(RIGHT)))(MOVE)(IF_FOOD_AHEAD
(MOVE)(LEFT))))))(NEUT (3)(PROGN3 (NEUT (3)(NEUT (2)(LEFT)(LEFT))(LEFT))(PROGN3
(IF_FOOD_AHEAD (PROGN2 (LEFT)(IF_FOOD_AHEAD (MOVE)(MOVE)))(RIGHT))(LEFT)(PROGN3
(RIGHT)(LEFT)(MOVE)))(NEUT (2)(MOVE)(RIGHT)))(NEUT (3)(PROGN3 (NEUT (3)(NEUT
(1)(NEUT (1)(RIGHT)(RIGHT))(NEUT (0)(MOVE)(MOVE)))(LEFT))(LEFT)(IF_FOOD_AHEAD (PROGN3
(RIGHT)(MOVE)(RIGHT))(NEUT (2)(LEFT)(LEFT))))(PROGN2 (PROGN2 (PROGN2 (RIGHT)(LEFT))(PROGN3 (PROGN3
(RIGHT)(MOVE)(LEFT))(LEFT)(MOVE)))(NEUT (2)(NEUT (2)(RIGHT)(MOVE))(LEFT))))))(PROGN3 (PROGN2 (PROGN2
(RIGHT)(RIGHT))(IF_FOOD_AHEAD (MOVE)(RIGHT)))(MOVE)(RIGHT)))
```

```
(IF_FOOD_AHEAD (NEUT (1)(NEUT (1)(NEUT (3)(PROGN3 (PROGN2 (IF_FOOD_AHEAD (RIGHT)(PROGN2
(RIGHT)(MOVE)))(RIGHT))(RIGHT)(NEUT (1)(LEFT)(RIGHT)))(PROGN3 (IF_FOOD_AHEAD
(LEFT)(LEFT))(LEFT)(RIGHT)))(NEUT (1)(IF_FOOD_AHEAD (NEUT (1)(NEUT (1)(PROGN2
(RIGHT)(LEFT))(LEFT))(LEFT))(PROGN3 (IF_FOOD_AHEAD (IF_FOOD_AHEAD (MOVE)(NEUT
(1)(LEFT)(LEFT)))(PROGN2 (PROGN2 (LEFT)(MOVE))(IF_FOOD_AHEAD (RIGHT)(RIGHT))))(PROGN3
(LEFT)(IF_FOOD_AHEAD (NEUT (0)(RIGHT)(RIGHT))(MOVE))(PROGN3 (LEFT)(RIGHT)(RIGHT)))(PROGN3
(IF_FOOD_AHEAD (MOVE)(PROGN3 (LEFT)(MOVE)(RIGHT)))(MOVE)(IF_FOOD_AHEAD (MOVE)(LEFT))))))(NEUT
(3)(PROGN3 (NEUT (1)(NEUT (2)(LEFT)(RIGHT))(LEFT))(PROGN3 (IF_FOOD_AHEAD (PROGN2 (LEFT)(PROGN2
(MOVE)(MOVE)))(RIGHT))(LEFT)(PROGN3 (RIGHT)(LEFT)(MOVE)))(NEUT (2)(MOVE)(RIGHT)))(NEUT (3)(NEUT
(1)(LEFT)(IF_FOOD_AHEAD (PROGN3 (RIGHT)(MOVE)(RIGHT))(NEUT (2)(LEFT)(LEFT))))(IF_FOOD_AHEAD
(PROGN2 (PROGN2 (MOVE)(MOVE))(PROGN3 (PROGN3 (RIGHT)(MOVE)(RIGHT))(LEFT)(MOVE)))(NEUT
(2)(NEUT (2)(RIGHT)(MOVE))(LEFT))))))(PROGN3 (PROGN2 (PROGN2 (RIGHT)(RIGHT))(IF_FOOD_AHEAD
(MOVE)(RIGHT)))(MOVE)(RIGHT)))
```

Mientras que el resto de la población está constituido por individuos de gran tamaño, con una gran cantidad de código inactivo y alta aptitud. Por ejemplo el individuo con la doceava mejor aptitud de la población (61) se representa con el programa de control:

```
(PROGN2 (PROGN3 (PROGN2 (LEFT)(MOVE))(PROGN3 (PROGN2 (MOVE)(RIGHT))(IF_FOOD_AHEAD (PROGN2
(RIGHT)(LEFT))(PROGN2 (MOVE)(MOVE)))(PROGN3 (PROGN2 (IF_FOOD_AHEAD (RIGHT)(PROGN2 (PROGN3
(LEFT)(RIGHT)(LEFT))(NEUT (1)(RIGHT)(LEFT)))))(PROGN2 (PROGN3 (IF_FOOD_AHEAD (RIGHT)(RIGHT))(PROGN2
(RIGHT)(RIGHT))(MOVE))(PROGN3 (PROGN2 (MOVE)(LEFT))(PROGN2 (RIGHT)(LEFT))(MOVE))))(PROGN3 (NEUT
(1)(PROGN3 (PROGN2 (MOVE)(RIGHT))(IF_FOOD_AHEAD (LEFT)(RIGHT))(RIGHT))(MOVE))(MOVE)(PROGN2
(PROGN3 (IF_FOOD_AHEAD (MOVE)(LEFT))(LEFT)(LEFT))(PROGN3 (LEFT)(MOVE)(MOVE))))(IF_FOOD_AHEAD
(IF_FOOD_AHEAD (PROGN2 (LEFT)(MOVE))(PROGN3 (RIGHT)(MOVE)(MOVE)))(NEUT (2)(PROGN3 (PROGN2
(LEFT)(RIGHT))(LEFT)(RIGHT))(MOVE))))))(PROGN3 (IF_FOOD_AHEAD (RIGHT)(IF_FOOD_AHEAD (PROGN2 (PROGN2
(MOVE)(IF_FOOD_AHEAD (MOVE)(LEFT)))(LEFT))(IF_FOOD_AHEAD (LEFT)(IF_FOOD_AHEAD (MOVE)(MOVE)))))(PROGN3
(LEFT)(LEFT)(NEUT (1)(LEFT)(RIGHT)))(NEUT (1)(NEUT (0)(PROGN3 (MOVE)(MOVE)(PROGN3 (PROGN2
(MOVE)(MOVE))(RIGHT)(LEFT)))(NEUT (2)(PROGN2 (RIGHT)(LEFT))(MOVE)))(PROGN2 (RIGHT)(RIGHT)))))(PROGN3
(IF_FOOD_AHEAD (IF_FOOD_AHEAD (PROGN3 (NEUT (0)(LEFT)(MOVE))(PROGN2 (PROGN2 (MOVE)(NEUT
(3)(LEFT)(MOVE)))(PROGN2 (IF_FOOD_AHEAD (LEFT)(RIGHT))(LEFT)))(IF_FOOD_AHEAD (RIGHT)(NEUT (2)(PROGN3
(RIGHT)(MOVE)(LEFT))(NEUT (0)(MOVE)(LEFT)))))(NEUT (1)(IF_FOOD_AHEAD (MOVE)(MOVE))(IF_FOOD_AHEAD
(MOVE)(IF_FOOD_AHEAD (NEUT (3)(MOVE)(RIGHT))(IF_FOOD_AHEAD (LEFT)(MOVE))))))(PROGN2 (NEUT
(1)(PROGN2 (NEUT (1)(PROGN2 (RIGHT)(MOVE))(NEUT (1)(RIGHT)(LEFT)))(LEFT))(IF_FOOD_AHEAD
(RIGHT)(RIGHT)))(MOVE)))(NEUT (3)(NEUT (2)(NEUT (2)(RIGHT)(IF_FOOD_AHEAD (MOVE)(MOVE)))(PROGN3
(PROGN3 (PROGN3 (LEFT)(IF_FOOD_AHEAD (RIGHT)(LEFT))(IF_FOOD_AHEAD (LEFT)(MOVE)))(PROGN2
(PROGN3 (MOVE)(MOVE)(LEFT))(IF_FOOD_AHEAD (MOVE)(RIGHT)))(NEUT (2)(PROGN3
(LEFT)(MOVE)(LEFT))(MOVE)))(MOVE)(LEFT)))(PROGN3 (NEUT (2)(RIGHT)(MOVE))(PROGN2 (PROGN3
(NEUT (3)(MOVE)(LEFT))(PROGN3 (LEFT)(IF_FOOD_AHEAD (RIGHT)(LEFT))(LEFT))(LEFT))(LEFT))(NEUT
(1)(PROGN3 (MOVE)(PROGN2 (PROGN3 (MOVE)(MOVE)(RIGHT))(MOVE))(NEUT (2)(RIGHT)(LEFT)))(NEUT
(2)(RIGHT)(IF_FOOD_AHEAD (LEFT)(LEFT))))))(NEUT (3)(NEUT (1)(NEUT (2)(MOVE)(NEUT
(2)(MOVE)(MOVE)))(NEUT (3)(MOVE)(MOVE)))(PROGN3 (IF_FOOD_AHEAD (NEUT (1)(RIGHT)(NEUT
(1)(PROGN3 (RIGHT)(MOVE)(MOVE))(RIGHT)))(PROGN2 (LEFT)(PROGN3 (IF_FOOD_AHEAD
(MOVE)(MOVE))(MOVE)(PROGN3 (MOVE)(LEFT)(LEFT)))))(PROGN3 (RIGHT)(NEUT (2)(MOVE)(MOVE))(PROGN2
(PROGN3 (LEFT)(LEFT)(RIGHT))(PROGN3 (PROGN3 (LEFT)(MOVE)(RIGHT))(RIGHT)(RIGHT))))(IF_FOOD_AHEAD
(PROGN2 (IF_FOOD_AHEAD (LEFT)(RIGHT))(PROGN3 (IF_FOOD_AHEAD (MOVE)(LEFT))(MOVE)(RIGHT)))(NEUT
(1)(NEUT (3)(MOVE)(MOVE))(RIGHT)))))))
```

Capítulo 8

Conclusiones y Trabajo futuro

8.1. Conclusiones

Se debe recordar que el Cómputo Evolutivo es un área que se encuentra en pleno crecimiento. Por esto es natural que día con día surjan una gran cantidad de modelos nuevos como metáforas de procesos biológicos muy diversos.

Al diseñar una modificación que incluya el concepto de neutralidad es importante no perder de vista que éste rebasa el simple hecho de agregar redundancia al proceso evolutivo. Apegándose a la definición de Wagner [107] se deben generar mecanismos que tengan algún efecto fenotípico (a mediano o largo plazo) mediante los elementos agregados.

En el capítulo 4 se presentaron diversos métodos diseñados con este propósito y se analizaron diferentes resultados publicados para cada uno de ellos.

Tomando esta idea como base se propuso agregar una función al conjunto de elementos de funciones de la representación en Programación Genética del problema de la Hormiga Artificial con el trazo Santa Fe. Dicha función fue nombrada función neutral o *Neut_progn* y aumenta la neutralidad explícita específicamente para el problema designado como caso de estudio.

Se realizaron múltiples experimentos en busca de identificar el funcionamiento de la Programación Genética bajo la influencia de la función neutral.

Se detectó que al agregar la función neutral y utilizar el método avanzado de mutación descrito en la sección 6.3.2 se mejora el desempeño de la PG para el problema en cuestión. Esta mejora se debe a que gracias a las reservas génicas provocadas por la función neutral la Programación Genética puede generar individuos que le permitan evitar estancarse en óptimos locales.

Aumentar la cantidad de código inactivo y permitir la activación de éste mediante la mutación de un nodo terminal *Neut_progn* es una adición útil para la representación en Programación Genética del problema de la Hormiga Artificial en el trazo Santa Fe. Esta

afirmación tiene sustento en la evidencia experimental del capítulo 7.

Por todo lo aquí planteado se cumplieron totalmente los objetivos generales y específicos de la tesis.

La función neutral es un elemento específicamente diseñado para el problema de la Hormiga Artificial. Sin embargo, es robusta y fácilmente puede extrapolarse a otros dominios.

Por ejemplo, en un problema de regresión simbólica se podría agregar una función suma neutral, con un selector y dos argumentos estándar. Al igual que para la función propuesta en el capítulo 6, el selector controlaría el número de argumentos que participarían en la suma.

Se propone la neutralidad explícita mediante la adición de elementos al conjunto de funciones como una forma de evitar la convergencia prematura en la Programación Genética. Es necesario realizar experimentos sobre otros paisajes de aptitud para dar soporte a esta afirmación.

8.2. Trabajo futuro

Los distintos caminos a seguir en un futuro son:

- Realizar experimentos con un mayor número de corridas con diferentes probabilidades de cruza y mutación para las series normal, neutral simple y neutral avanzada para brindar mayor exactitud a los cálculos definidos en la sección 7.1.
- Realizar conteos de nodos activos e inactivos para diferentes combinaciones de probabilidades de cruza y mutación con el objetivo de identificar diferencias sutíles entre las series normal, neutral simple y neutral avanzada.
- Realizar un análisis formal sobre el efecto de la función neutral en la representación en Programación Genética del problema de la Hormiga Artificial con el trazo Santa Fe.
- Modelar los recorridos neutrales realizados por la PG sobre el problema de la Hormiga Artificial con los experimentos base y con agente condicionado.
- Probar el trazo Santa Fe con la versión de búsqueda mediante Escalado de Colinas que opera sobre árboles usando el conjunto de funciones del problema de la Hormiga Artificial con la función neutral.
- Diseñar funciones neutrales para agregar a los conjuntos de funciones de otros problemas de la Programación Genética y comparar los resultados con aquellos obtenidos para las versiones originales.
- Construir una definición sin ambigüedad de neutralidad.

Apéndice A

Experimento previo

Antes de realizar los tres experimentos planteados en el capítulo 7 se efectuó un experimento previo para identificar si existe alguna relación entre el límite de profundidad final de los árboles y el comportamiento de la PG cuando se agrega la función neutral al conjunto de funciones del problema de la Hormiga Artificial.

Los resultados obtenidos para dicho experimento previo se presentan y discuten en este apéndice.

A.1. Planteamiento

En el presente experimento se realizaron tres series de corridas. En la primera de dichas series se utilizó el conjunto de funciones tradicional y se llama **serie normal**. La segunda incluye la función *Neut_progn* con el método simple de mutación (ver sección 6.3.2) y se llama **serie neutral simple**. La última incluye la función *Neut_progn* con el método neutral avanzado de mutación y se llama **serie neutral avanzada**.

Tabla A.1: *Parámetros para la PG en experimento previo.*

Parámetro	**Valor**
Tamaño de población	150
Número de generaciones	333
Selección	Torneo (con tamaño de 3 individuos).
Profundidad inicial	3
Profundidad final	Variable
Probabilidad de cruza	85 %.
Probabilidad de mutación (por nodo)	6 %.
Función de Aptitud	Comida remanente del trazo.

Para cada corrida efectuada en el presente experimento se utilizaron los parámetros definidos en la Tabla A.1. En dicha tabla no se definió un límite de profundidad final porque para cada serie se realizaron corridas con cada uno de los elementos del intervalo $[3-13]$ como niveles límite de profundidad final.

Los parámetros de la Tabla A.1 se definieron en base a experimentos anteriores realizados con el conjunto de funciones estándar para el problema de la Hormiga Artificial con el trazo Santa Fe. Se tomó la configuración en la que los diferentes experimentos realizados mostraron mejores resultados promedio.

Para que los resultados del presente experimento fueran significativos, se realizaron 50 corridas independientes para cada elemento del intervalo de profundidad final en cada serie de corridas.

Cuando el límite de la profundidad final se asigna a tres niveles los árboles generados por la PG tendrán un número de nodos máximo dentro del rango $15 \geq l_{p_0} \geq 40$. Cuando el límite de la profundidad final se asigna a 13 niveles los árboles generados por la Programación Genética tendrán un número máximo de nodos dentro del rango $16383 \geq l_{p_0} \geq 2391484$.

En otras palabras, en el transcurso de las diferentes corridas realizadas se estarán utilizando árboles con un límite en el tamaño máximo que va desde 15 nodos hasta 2391484 nodos. Estos datos se mencionan con la intensión de hacer notar la magnitud del muestreo realizado sobre las diferentes profundidades finales.

A.2. Métricas

En el presente experimento se emplearon las dos medidas de desempeño descritas en la sección 7.1.1:

- Esfuerzo, calculado mediante (7.2).
- Promedio de aptitud del mejor individuo para la última generación, que se calculó mediante (7.3) con $t = 333$.

Adicionalmente se calculó la desviación estándar de la aptitud del mejor individuo en la generación t mediante:

$$\delta_{min}(t) = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(min(f_{I_1}^t, f_{I_2}^t, \ldots f_{I_n}^t)_j - \overline{f_{min}}(t))^2} \tag{A.1}$$

donde n representa el tamaño de la población, N el número de corridas independientes realizadas, $f_{I_i}^t$ corresponde al valor de aptitud del i-ésimo individuo en la generación t,

$min(f^t_{I_1}, f^t_{I_2}, \ldots f^t_{I_n})_j$ hace referencia a la aptitud más baja de toda la población en la generación t para la j-ésima corrida y $\overline{f_{min}}(t)$ es el promedio de la aptitud del mejor individuo en la generación t.

A.3. Resultados

La Figura A.1 presenta los resultados de la comparación de esfuerzo para las tres series de corridas con diferentes valores límite de profundidad final.

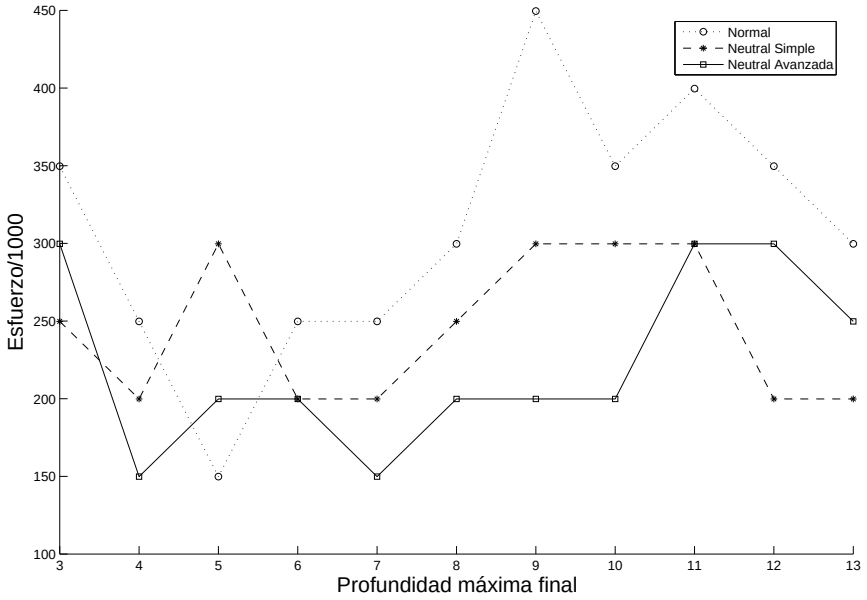

Figura A.1: *Esfuerzo/1000 de diferentes valores de profundidad máxima.*

En dicha figura se puede observar que para límites bajos de profundidad final (menores a seis niveles) las tres series de corridas presentan picos en los que requieren un menor esfuerzo que las otras dos para encontrar un programa de control óptimo.

Sin embargo, para límites de profundidad final mayores a seis las series neutrales requieren un menor valor de esfuerzo que la serie normal para encontrar un programa de control óptimo. Específicamente en el intervalo $[7-11]$ la serie neutral avanzada se mantiene al menos a $50,000$ puntos de esfuerzo (una corrida de diferencia) de las otras dos series.

La Figura A.2 presenta los resultados de la comparación de promedio y desviación estándar de aptitud del mejor individuo para las tres series de corridas con diferentes valores límite de profundidad final.

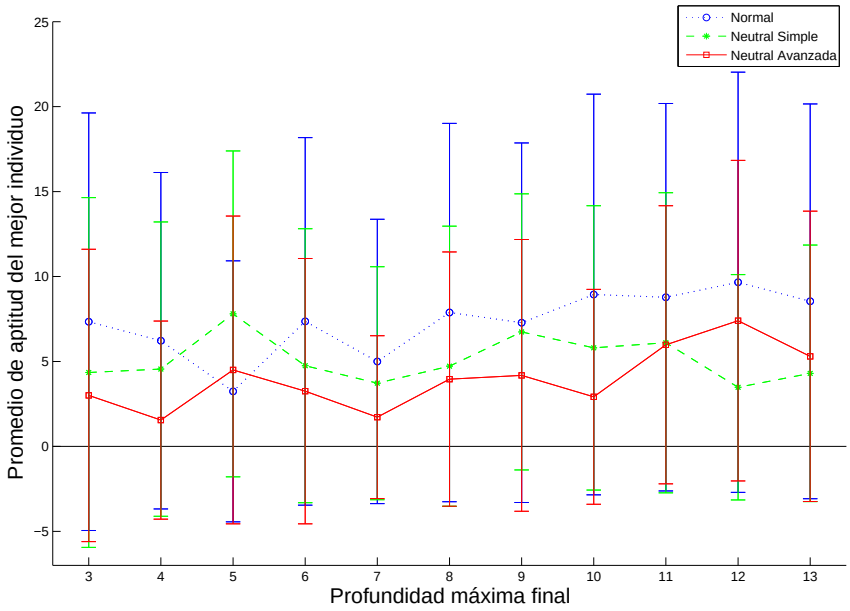

Figura A.2: *Aptitud promedio de diferentes valores de profundidad máxima.*

Dicha figura presenta un comportamiento parecido para el promedio de aptitud del mejor individuo al detectado para la métrica de esfuerzo. La diferencia más notable es el valor obtenido para la serie normal cuando la profundidad final es igual a nueve.

Cabe mencionar que la desviación estándar tiende a ser menor para las series neutrales con respecto a la serie normal. Esto indica que el valor de cada una de las corridas individuales de las series neutrales se mantiene más cercano al promedio de aptitud del mejor individuo de la serie en cuestión que los correspondientes de la serie normal.

Dado que ambas figuras son consistentes se afirma que las series neutrales se comportan mejor que la serie normal cuando la profundidad final es mayor a seis niveles.

Una explicación a este comportamiento es que la Programación Genética requiere que las secciones de código inactivo generadas por la función neutral dispongan de una buena cantidad de nodos para funcionar como *reserva génica* o generar una protección global contra cruzas y mutaciones destructivas en las etapas finales del proceso evolutivo.

La profundidad final se asignó a ocho niveles para que dichas características se presenten en el transcurso de los diferentes experimentos discutidos en el capítulo 7.

Apéndice B

Datos experimentales

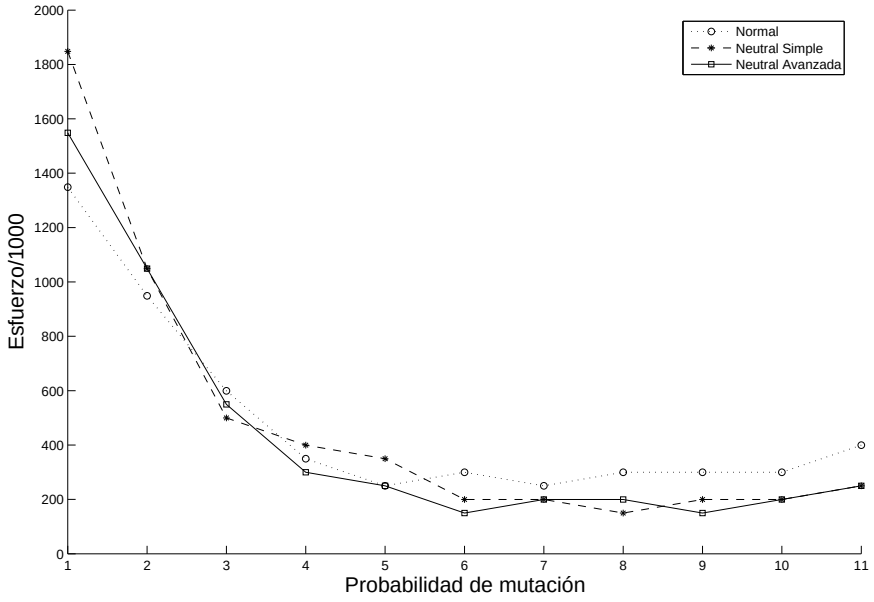

Figura B.1: *Comparación de esfuerzo/1000 con 20 % de probabilidad de cruza para configuración base.*

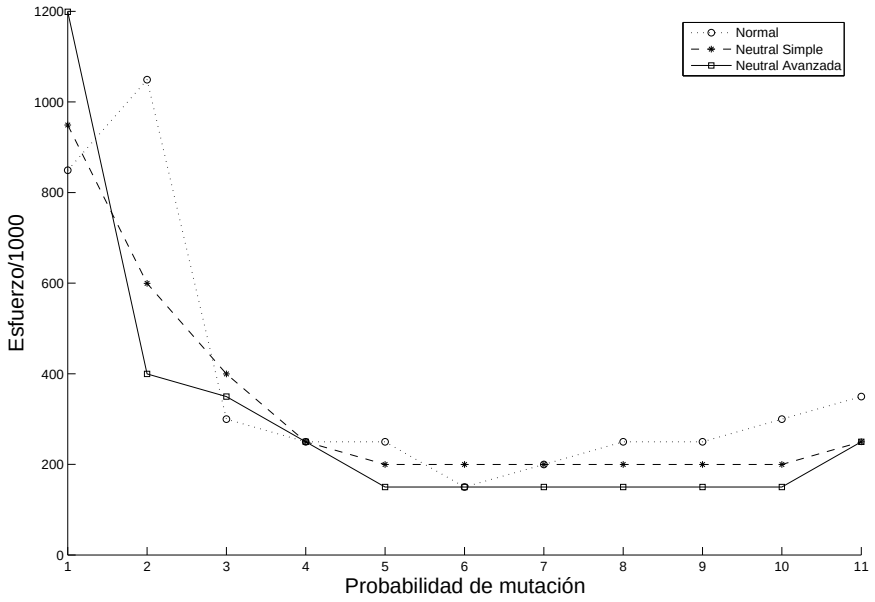

Figura B.2: *Comparación de esfuerzo/1000 con 45 % de probabilidad de cruza para configuración base.*

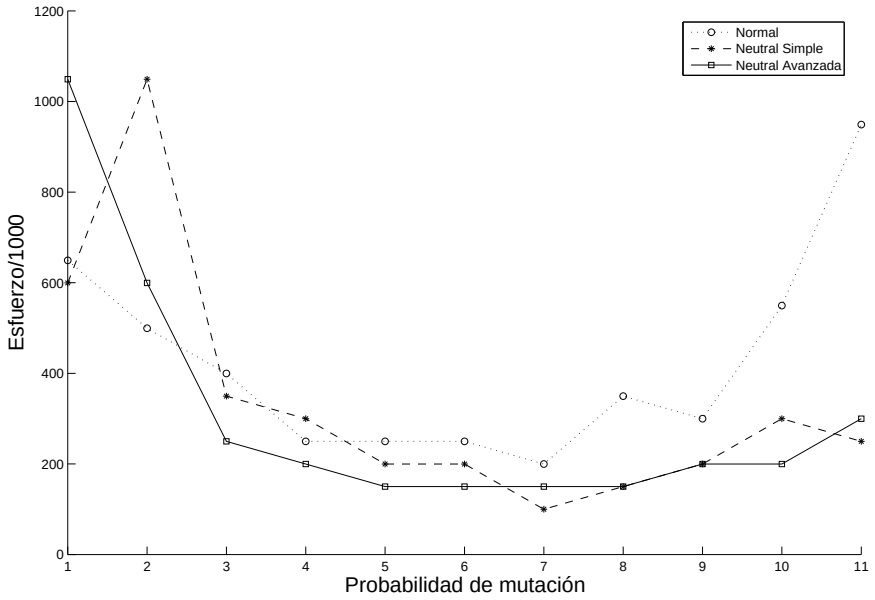

Figura B.3: *Comparación de esfuerzo/1000 con 60 % de probabilidad de cruza para configuración base.*

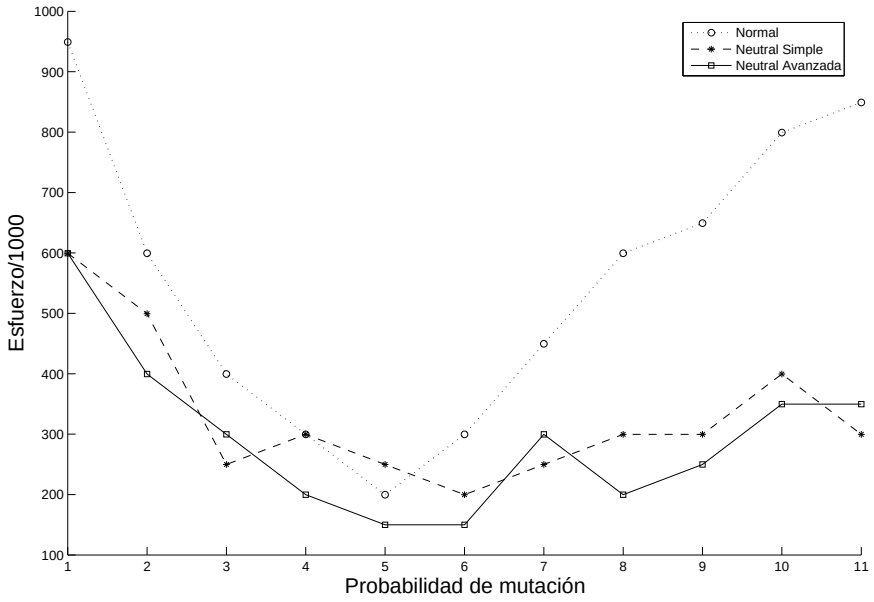

Figura B.4: *Comparación de esfuerzo/1000 con 85 % de probabilidad de cruza para configuración base.*

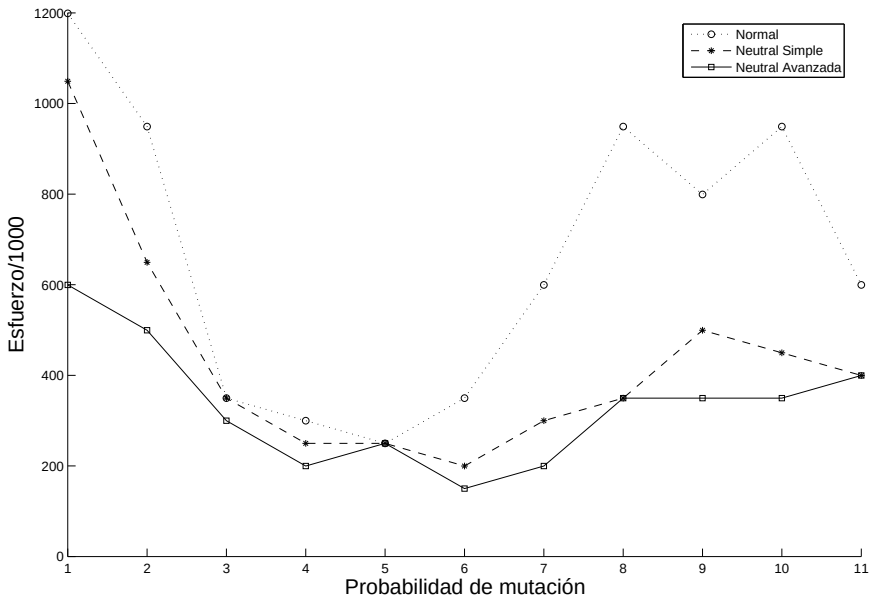

Figura B.5: *Comparación de esfuerzo/1000 con 95 % de probabilidad de cruza para configuración base.*

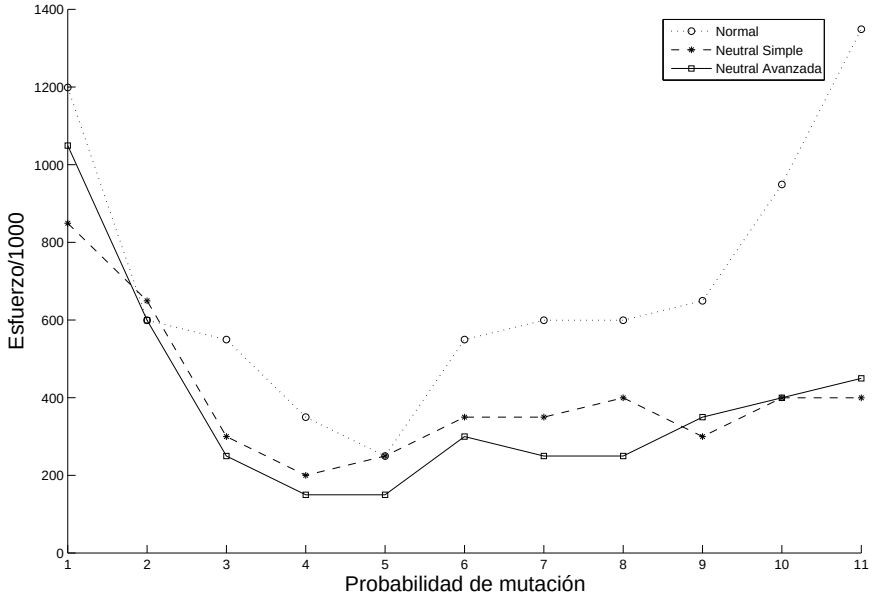

Figura B.6: *Comparación de esfuerzo/1000 con 100 % de probabilidad de cruza para configuración base.*

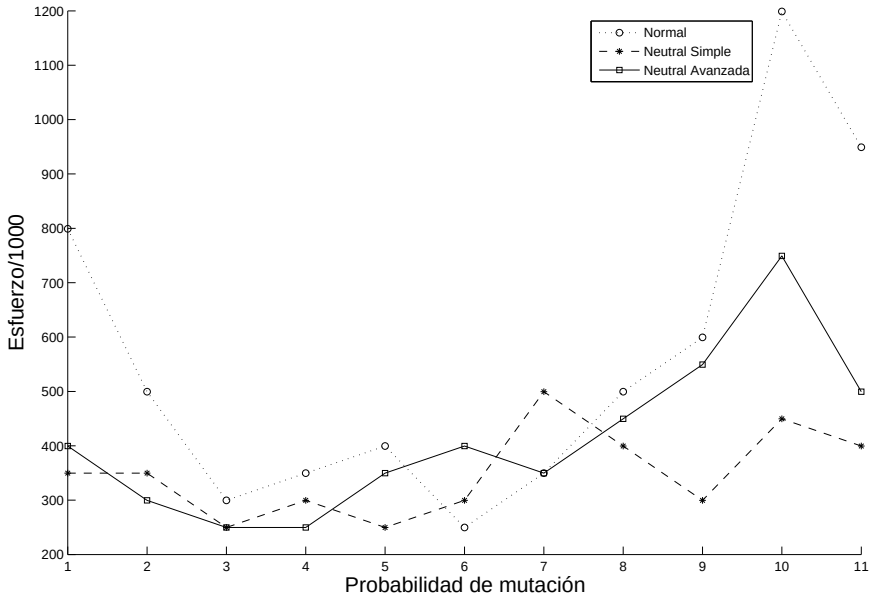

Figura B.7: *Comparación de esfuerzo/1000 con 20 % de probabilidad de cruza para configuración con agente condicionado.*

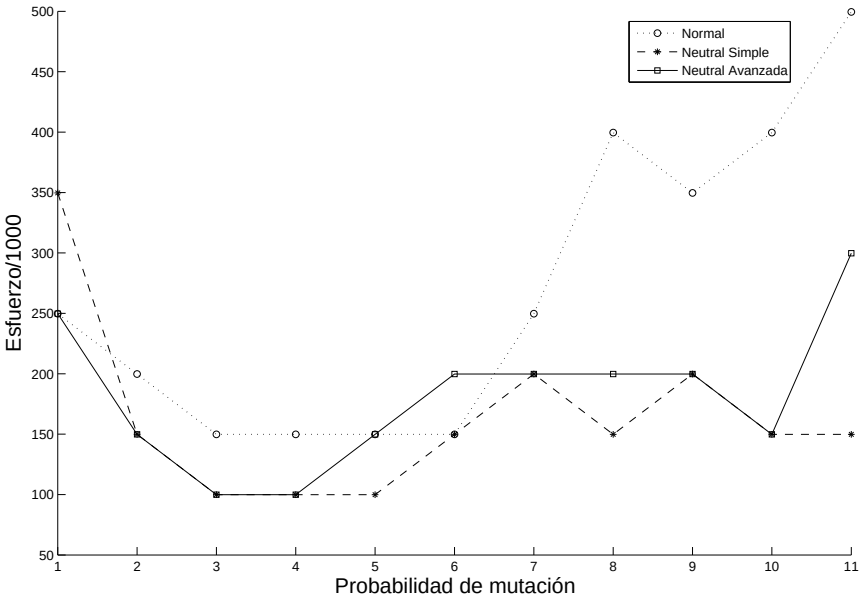

Figura B.8: *Comparación de esfuerzo/1000 con 45 % de probabilidad de cruza para configuración con agente condicionado.*

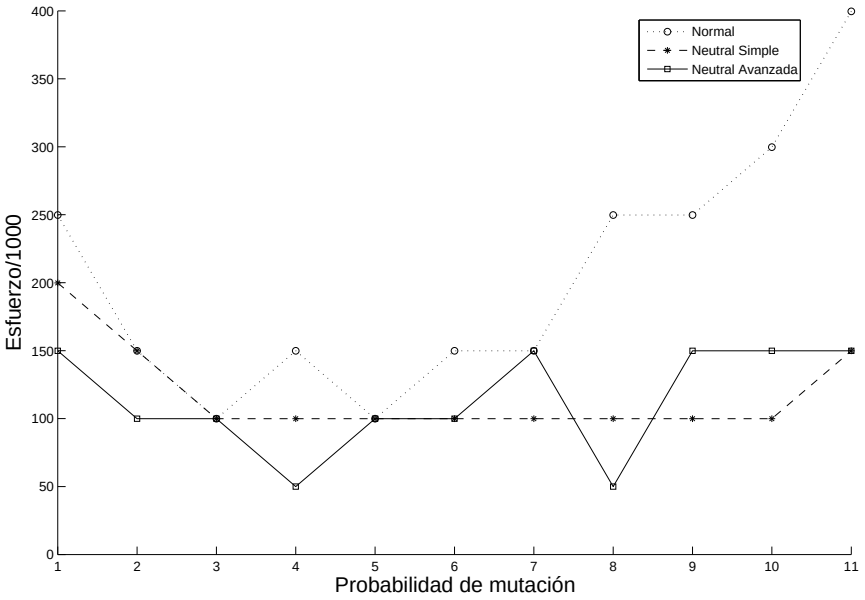

Figura B.9: *Comparación de esfuerzo/1000 con 60 % de probabilidad de cruza para configuración con agente condicionado.*

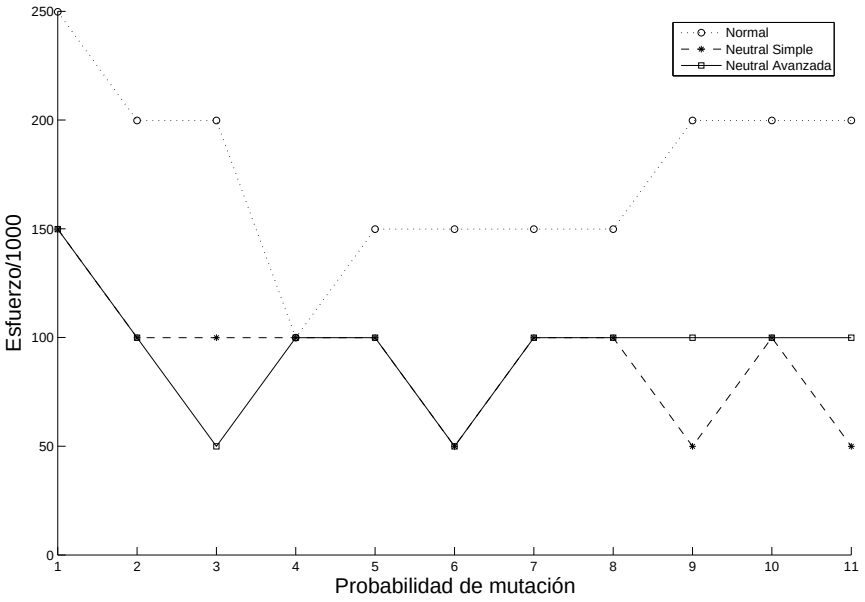

Figura B.10: *Comparación de esfuerzo/1000 con 85 % de probabilidad de cruza para configuración con agente condicionado.*

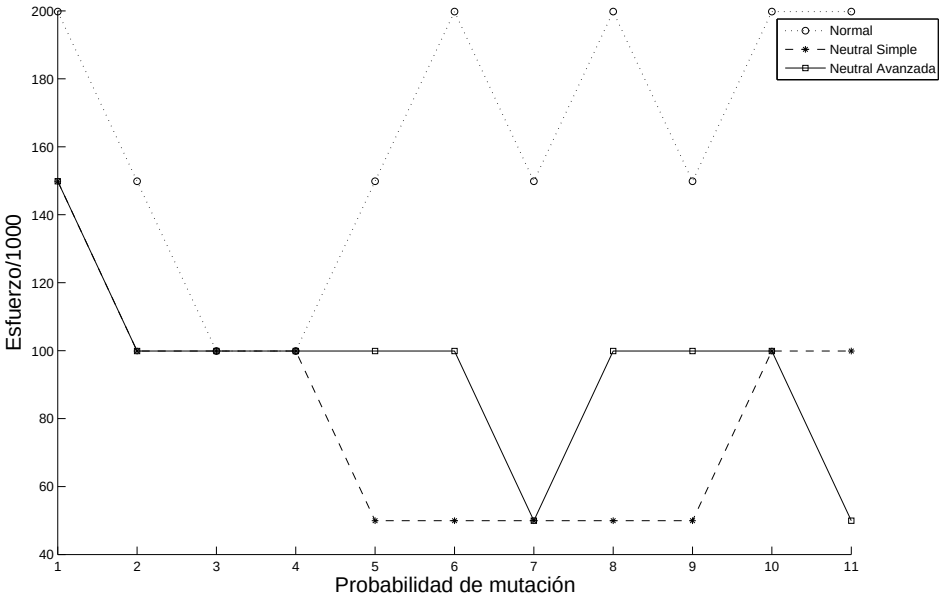

Figura B.11: *Comparación de esfuerzo/1000 con 95 % de probabilidad de cruza para configuración con agente condicionado.*

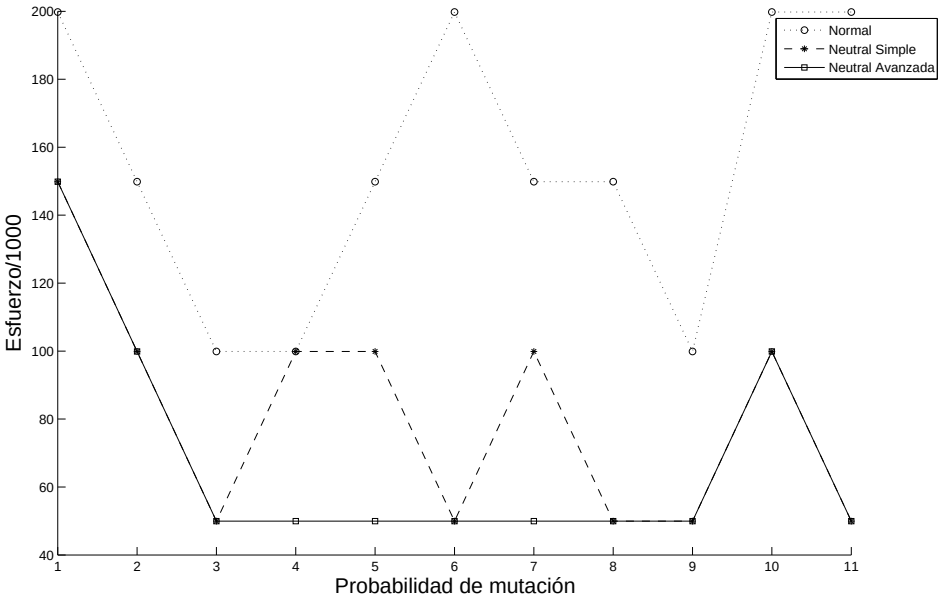

Figura B.12: *Comparación de esfuerzo/1000 con 100 % de probabilidad de cruza para configuración con agente condicionado.*

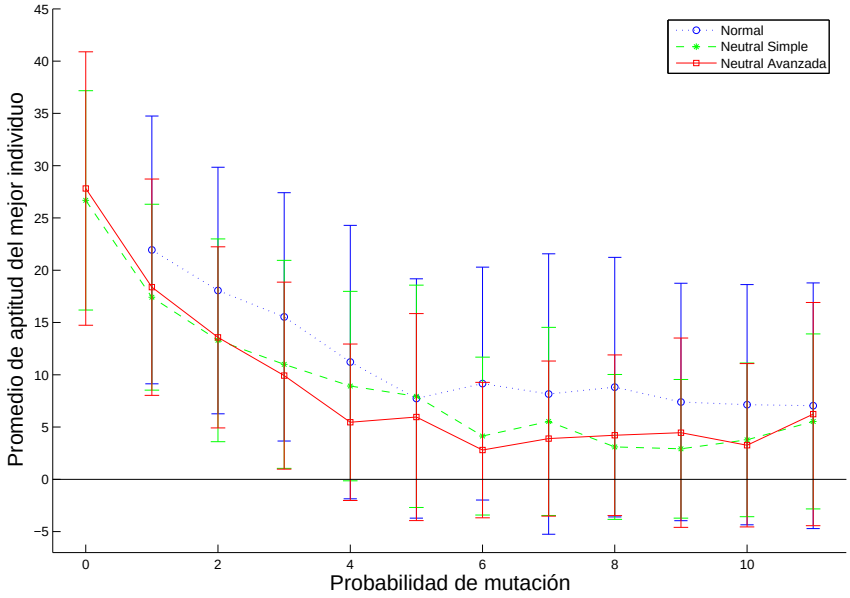

Figura B.13: *Comparación de aptitud promedio con 20 % de probabilidad de cruza para configuración base.*

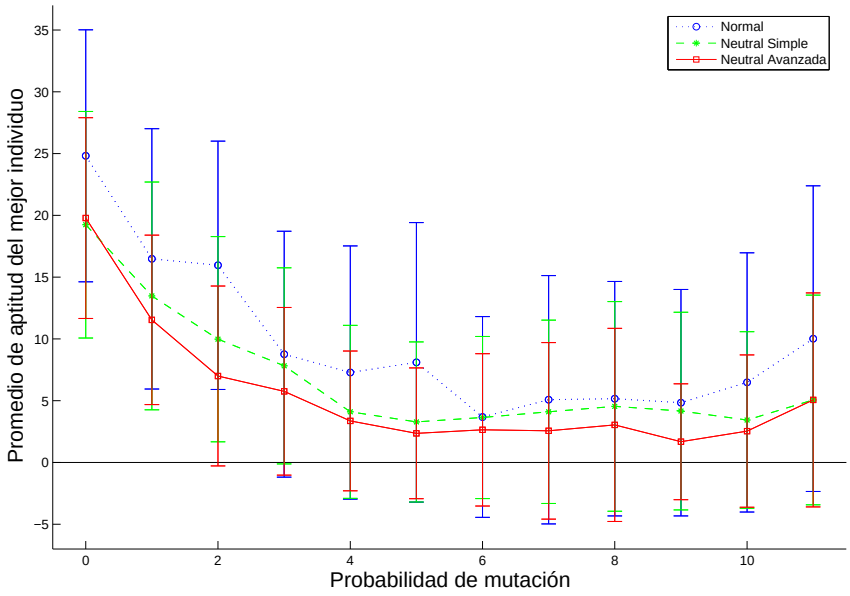

Figura B.14: *Comparación de aptitud promedio con 45 % de probabilidad de cruza para configuración base.*

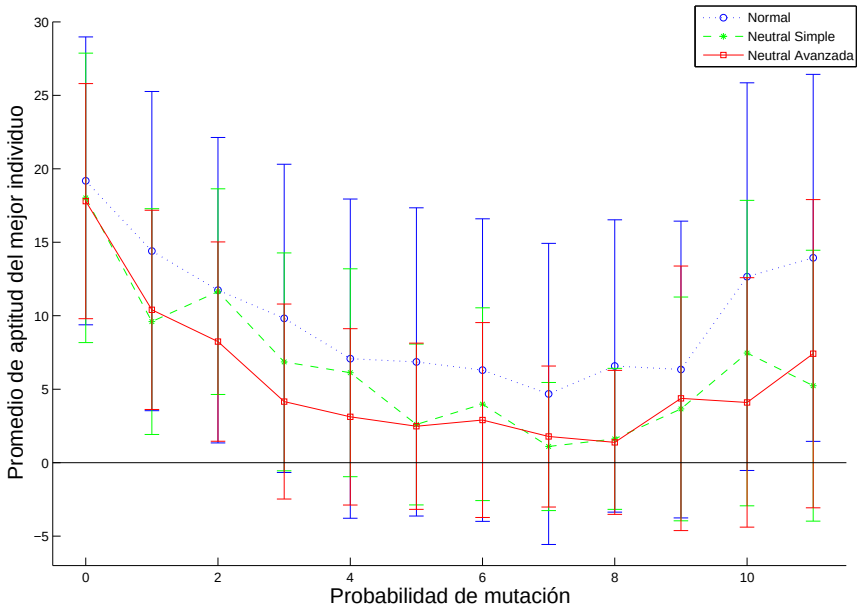

Figura B.15: *Comparación de aptitud promedio con 60 % de probabilidad de cruza para configuración base.*

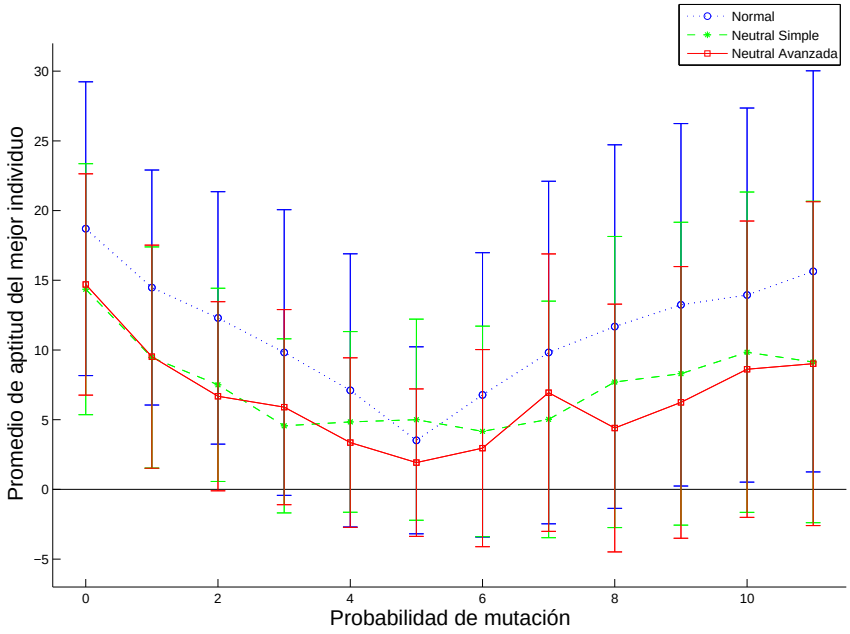

Figura B.16: *Comparación de aptitud promedio con 85 % de probabilidad de cruza para configuración base.*

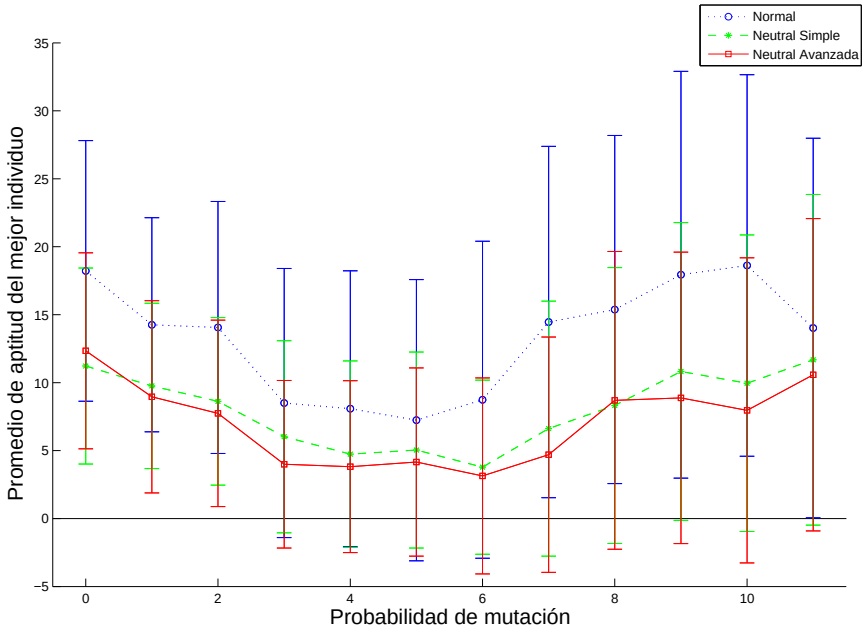

Figura B.17: *Comparación de aptitud promedio con 95 % de probabilidad de cruza para configuración base.*

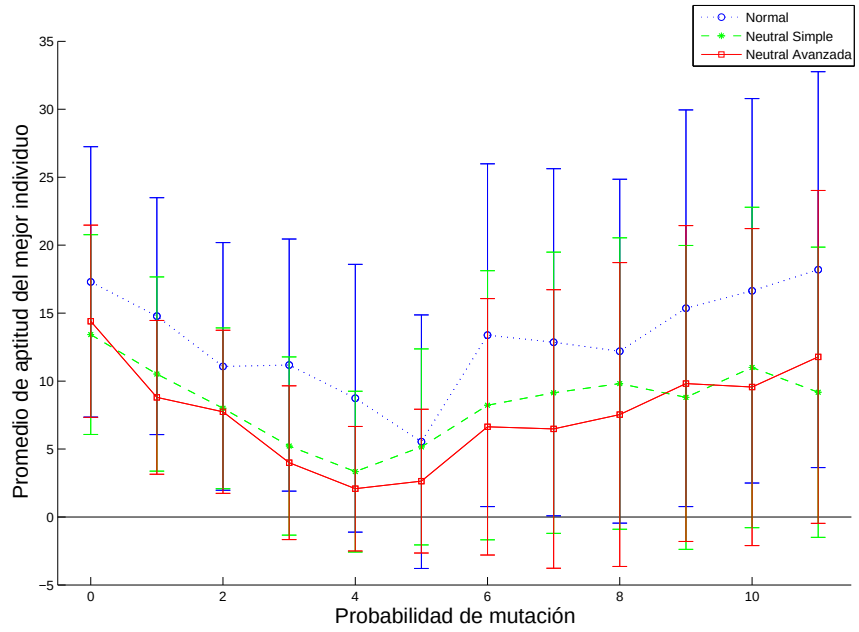

Figura B.18: *Comparación de aptitud promedio con 100 % de probabilidad de cruza para configuración base.*

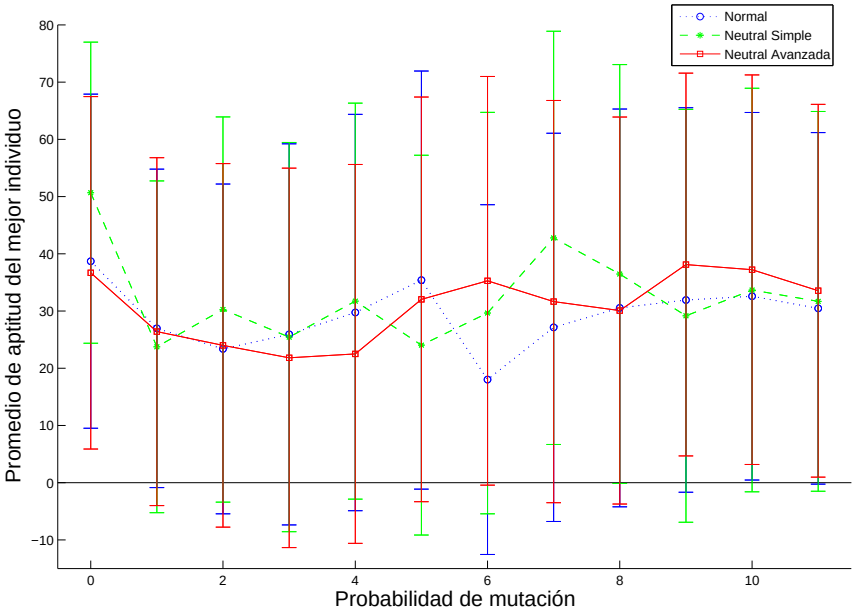

Figura B.19: *Comparación de aptitud promedio con 20 % de probabilidad de cruza para configuración con agente condicionado.*

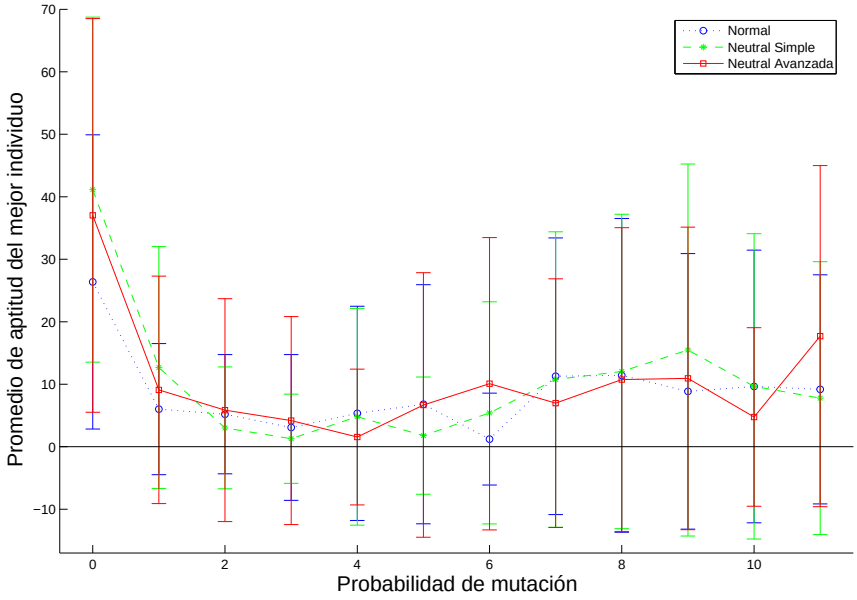

Figura B.20: *Comparación de aptitud promedio con 45 % de probabilidad de cruza para configuración con agente condicionado.*

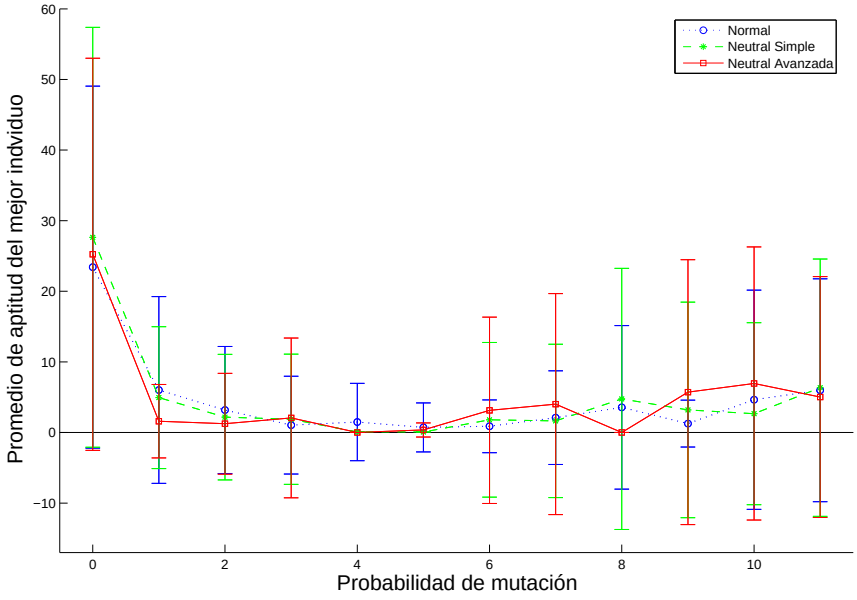

Figura B.21: *Comparación de aptitud promedio con 60 % de probabilidad de cruza para configuración con agente condicionado.*

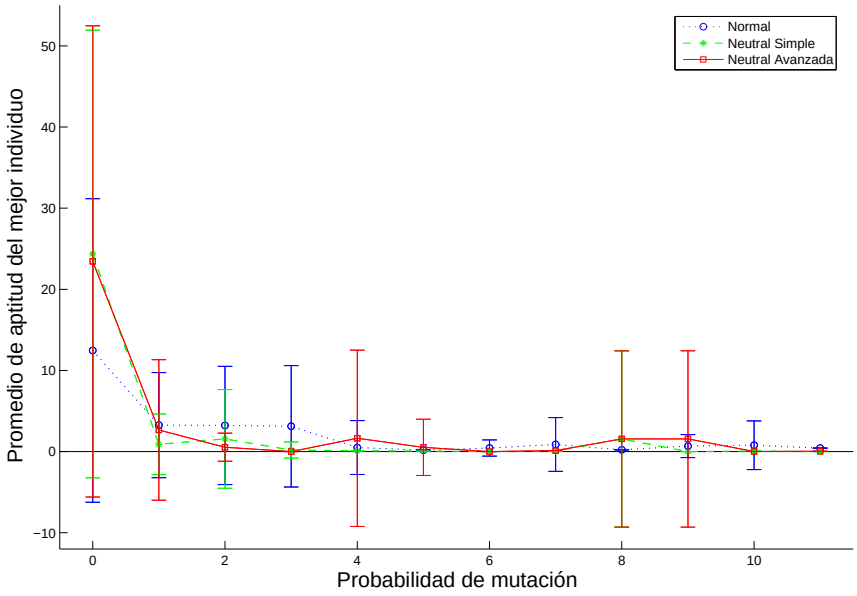

Figura B.22: *Comparación de aptitud promedio con 85 % de probabilidad de cruza para configuración con agente condicionado.*

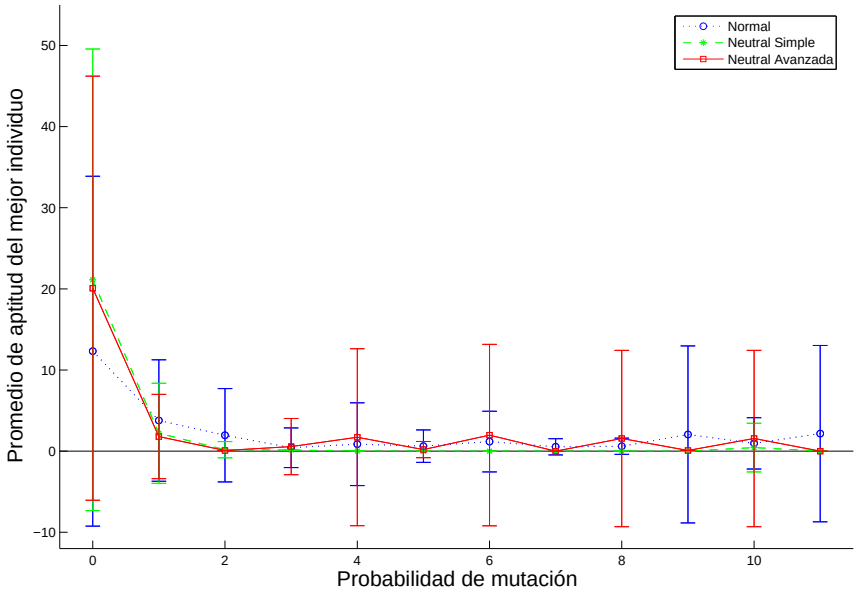

Figura B.23: *Comparación de aptitud promedio con 95 % de probabilidad de cruza para configuración con agente condicionado.*

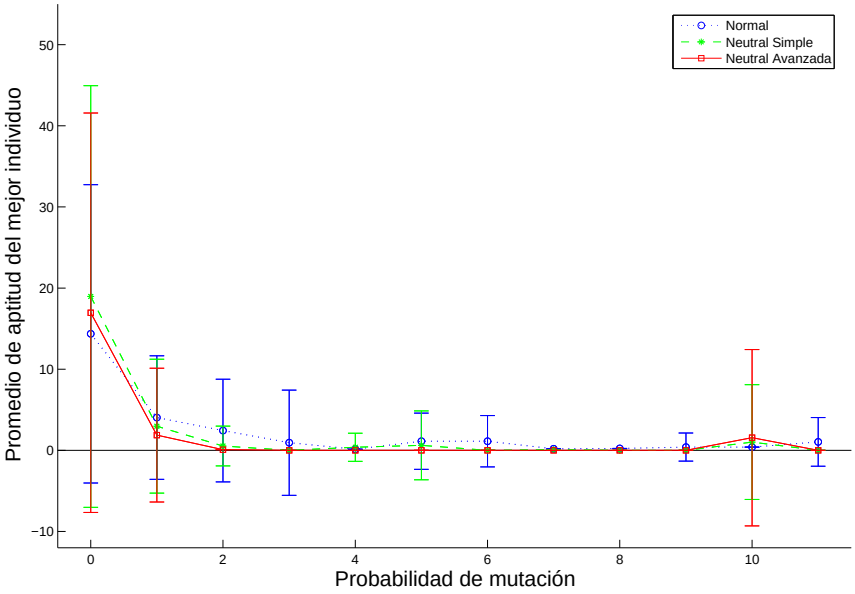

Figura B.24: *Comparación de aptitud promedio con 100 % de probabilidad de cruza para configuración con agente condicionado.*

Tabla B.1: *Esfuerzo/1000 de la serie normal del configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	∞	113	31	24	37	24
1 %	27	17	13	19	24	24
2 %	19	21	10	12	19	12
3 %	12	6	8	8	7	11
4 %	7	5	5	6	6	7
5 %	5	5	5	4	5	5
6 %	6	3	5	6	7	11
7 %	5	4	4	9	12	12
8 %	6	5	7	12	19	12
9 %	6	5	6	13	16	13
10 %	6	6	11	16	19	19
11 %	8	7	19	17	12	27

Tabla B.2: *Esfuerzo/1000 de la serie neutral simple con configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	75	56	31	24	19	24
1 %	37	19	12	12	21	17
2 %	21	12	21	10	13	13
3 %	10	8	7	5	7	6
4 %	8	5	6	6	5	4
5 %	7	4	4	5	5	5
6 %	4	4	4	4	4	7
7 %	4	4	2	5	6	7
8 %	3	4	3	6	7	8
9 %	4	4	4	6	10	6
10 %	4	4	6	8	9	8
11 %	5	5	5	6	8	8

Tabla B.3: *Esfuerzo/1000 de la serie neutral avanzada con configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	228	56	56	44	24	44
1 %	31	24	21	12	12	21
2 %	21	8	12	8	10	12
3 %	11	7	5	6	6	5
4 %	6	5	4	4	4	3
5 %	5	3	3	3	5	3
6 %	3	3	3	3	3	6
7 %	4	3	3	6	4	5
8 %	4	3	3	4	7	5
9 %	3	3	4	5	7	7
10 %	4	3	4	7	7	8
11 %	5	5	6	7	8	9

Tabla B.4: *Esfuerzo/1000 de la serie normal con configuración con agente condicionado.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	31	16	12	8	5	8
1 %	16	5	5	5	4	4
2 %	10	4	3	4	3	3
3 %	6	3	2	4	2	2
4 %	7	3	3	2	2	2
5 %	8	3	2	3	3	3
6 %	5	3	3	3	4	4
7 %	7	5	3	3	3	3
8 %	10	8	5	3	4	3
9 %	12	7	5	4	3	2
10 %	24	8	6	4	4	4
11 %	19	10	8	4	4	4

Tabla B.5: *Esfuerzo/1000 de la serie neutral simple con configuración con agente condicionado.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	75	37	8	9	6	7
1 %	7	7	4	3	3	3
2 %	7	3	3	2	2	2
3 %	5	2	2	2	2	1
4 %	6	2	2	2	2	2
5 %	5	2	2	2	1	2
6 %	6	3	2	1	1	1
7 %	10	4	2	2	1	2
8 %	8	3	2	2	1	1
9 %	6	4	2	1	1	1
10 %	9	3	2	2	2	2
11 %	8	3	3	1	2	1

Tabla B.6: *Esfuerzo/1000 de la serie neutral avanzada con configuración con agente condicionado.*

Mutación \ Cruza	20 %	45 %	60 %	85 %	95 %	100 %
0 %	15	16	12	7	8	7
1 %	8	5	3	3	3	3
2 %	6	3	2	2	2	2
3 %	5	2	2	1	2	1
4 %	5	2	1	2	2	1
5 %	7	3	2	2	2	1
6 %	8	4	2	1	2	1
7 %	7	4	3	2	1	1
8 %	9	4	1	2	2	1
9 %	11	4	3	2	2	1
10 %	15	3	3	2	2	2
11 %	10	6	3	2	1	1

Tabla B.7: *Promedio de la aptitud del mejor individuo de la serie normal con configuración base.*

Mutación \ Cruza	20 %	45 %	60 %	85 %	95 %	100 %
0 %	51.6	24.82	19.18	18.7	18.22	17.3
1 %	21.94	16.48	14.4	14.48	14.26	14.78
2 %	18.06	15.96	11.74	12.3	14.06	11.08
3 %	15.54	8.76	9.82	9.82	8.5	11.18
4 %	11.22	7.28	7.08	7.1	8.08	8.74
5 %	7.74	8.1	6.86	3.52	7.24	5.54
6 %	9.16	3.68	6.3	6.78	8.74	13.38
7 %	8.16	5.08	4.68	9.82	14.46	12.86
8 %	8.82	5.16	6.58	11.68	15.38	12.2
9 %	7.4	4.84	6.34	13.24	17.94	15.36
10 %	7.14	6.48	12.66	13.94	18.62	16.64
11 %	7.04	10.02	13.94	15.64	14.02	18.2

Tabla B.8: *Promedio de la aptitud del mejor individuo de la serie neutral simple con configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	26.68	19.24	18.02	14.36	11.22	13.42
1 %	17.42	13.48	9.6	9.46	9.76	10.52
2 %	13.3	9.98	11.64	7.5	8.62	8
3 %	11	7.82	6.86	4.56	6.02	5.22
4 %	8.92	4.1	6.12	4.84	4.74	3.34
5 %	7.94	3.28	2.6	5	5.04	5.16
6 %	4.14	3.64	3.98	4.16	3.78	8.22
7 %	5.54	4.1	1.1	5.02	6.62	9.14
8 %	3.1	4.54	1.62	7.7	8.32	9.82
9 %	2.92	4.16	3.66	8.3	10.82	8.8
10 %	3.78	3.44	7.46	9.84	9.96	11
11 %	5.54	5.06	5.24	9.14	11.68	9.18

Tabla B.9: *Promedio de la aptitud del mejor individuo de la serie neutral avanzada con configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	27.82	19.78	17.8	14.7	12.34	14.4
1 %	18.38	11.54	10.4	9.52	8.96	8.8
2 %	13.58	7	8.24	6.68	7.74	7.74
3 %	9.92	5.76	4.16	5.9	4	4
4 %	5.46	3.36	3.12	3.36	3.82	2.08
5 %	5.96	2.36	2.48	1.92	4.16	2.64
6 %	2.8	2.64	2.9	2.96	3.14	6.64
7 %	3.9	2.56	1.78	6.94	4.7	6.48
8 %	4.22	3.04	1.38	4.4	8.7	7.54
9 %	4.46	1.68	4.38	6.24	8.88	9.82
10 %	3.26	2.54	4.1	8.62	7.96	9.56
11 %	6.24	5.06	7.42	9.02	10.58	11.78

Tabla B.10: *Promedio de la aptitud del mejor individuo de la serie normal con configuración con agente condicionado.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	38.7	26.38	23.42	12.46	12.32	14.36
1 %	26.96	6.02	6.02	3.26	3.78	4.04
2 %	23.38	5.2	3.18	3.22	1.96	2.44
3 %	25.92	3.08	1.04	3.12	0.42	0.94
4 %	29.74	5.34	1.48	0.5	0.86	0.14
5 %	35.4	6.8	0.72	0.18	0.62	1.12
6 %	18.02	1.22	0.88	0.44	1.18	1.12
7 %	27.14	11.28	2.1	0.88	0.54	0.2
8 %	30.54	11.42	3.56	0.22	0.6	0.24
9 %	31.92	8.86	1.26	0.68	2.06	0.4
10 %	32.58	9.64	4.64	0.78	0.96	0.4
11 %	30.46	9.18	5.98	0.42	2.16	1.04

Tabla B.11: *Promedio de la aptitud del mejor individuo de la serie neutral simple con configuración con agente condicionado.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	50.68	41.14	27.64	24.36	21.12	18.96
1 %	23.74	12.66	4.94	0.9	2.2	2.98
2 %	30.26	3.02	2.18	1.56	0.18	0.54
3 %	25.44	1.28	1.88	0.2	0.14	0
4 %	31.72	4.8	0.08	0.08	0.02	0.38
5 %	24.04	1.78	0.06	0.06	0	0.62
6 %	29.64	5.42	1.8	0	0	0
7 %	42.78	10.76	1.64	0.04	0	0.08
8 %	36.46	12.04	4.76	1.56	0	0
9 %	29.16	15.48	3.2	0	0	0
10 %	33.66	9.66	2.66	0.06	0.44	1.02
11 %	31.68	7.78	6.34	0	0.02	0

Tabla B.12: *Promedio de la aptitud del mejor individuo de la serie neutral avanzada con configuración con agente condicionado.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	36.68	37.02	25.24	23.44	20.08	16.96
1 %	26.4	9.1	1.6	2.66	1.8	1.88
2 %	24	5.86	1.24	0.54	0.08	0.08
3 %	21.82	4.18	2.06	0	0.56	0
4 %	22.5	1.56	0	1.64	1.72	0
5 %	32.04	6.68	0.36	0.52	0.2	0
6 %	35.28	10.08	3.14	0	1.98	0
7 %	31.64	6.98	4.02	0.12	0	0
8 %	30.08	10.74	0	1.56	1.56	0
9 %	38.12	10.94	5.72	1.56	0.08	0
10 %	37.22	4.76	6.94	0.02	1.56	1.56
11 %	33.54	17.7	5.02	0.04	0	0

Tabla B.13: *Desviación estandar de la aptitud del mejor individuo de la serie normal con configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	45.6541	10.19804	9.79796	10.53565	9.59166	9.94987
1 %	12.80625	10.53565	10.86278	8.42615	7.87401	8.7178
2 %	11.78983	10.04988	10.3923	9.05539	9.27362	9.11043
3 %	11.87434	9.94987	10.48809	10.24695	9.8995	9.27362
4 %	13.0767	10.24695	10.86278	9.79796	10.14889	9.84886
5 %	11.44552	11.31371	10.48809	6.7082	10.34408	9.32738
6 %	11.13553	8.12404	10.29563	10.19804	11.6619	12.60952
7 %	13.41641	10.04988	10.24695	12.28821	12.92285	12.76715
8 %	12.40967	9.48683	9.94987	13.0384	12.80625	12.64911
9 %	11.35782	9.16515	10.0995	13	14.96663	14.59452
10 %	11.48913	10.48809	13.19091	13.41641	14.03567	14.14214
11 %	11.74734	12.36932	12.49	14.3875	13.96424	14.56022

Tabla B.14: *Desviación estandar de la aptitud del mejor individuo de la serie neutral simple con configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	10.48809	9.16515	9.84886	9	7.2111	7.34847
1 %	8.88819	9.21954	7.68115	7.93725	6.08276	7.14143
2 %	9.69536	8.30662	7	6.9282	6.16441	5.91608
3 %	9.94987	7.93725	7.4162	6.245	7.07107	6.55744
4 %	9.05539	7	7.07107	6.48074	6.85565	5.91608
5 %	10.63015	6.48074	5.47723	7.2111	7.2111	7.2111
6 %	7.54983	6.55744	6.55744	7.54983	6.40312	9.8995
7 %	9	7.4162	4.3589	8.48528	9.38083	10.34408
8 %	6.9282	8.48528	4.79583	10.44031	10.14889	10.72381
9 %	6.63325	8	7.61577	10.86278	10.95445	11.18034
10 %	7.34847	7.14143	10.3923	11.48913	10.90871	11.78983
11 %	8.3666	8.48528	9.21954	11.53256	12.16553	10.67708

Tabla B.15: *Desviación estandar de la aptitud del mejor individuo de la serie neutral avanzada con configuración base.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	13.0767	8.12404	8	7.93725	7.2111	7.07107
1 %	10.34408	6.85565	6.78233	8	7.07107	5.65685
2 %	8.66025	7.28011	6.78233	6.78233	6.85565	6
3 %	8.94427	6.78233	6.63325	7	6.16441	5.65685
4 %	7.48331	5.65685	6	6.08276	6.32456	4.58258
5 %	9.8995	5.2915	5.65685	5.2915	6.9282	5.2915
6 %	6.48074	6.16441	6.63325	7.07107	7.2111	9.43398
7 %	7.4162	7.14143	4.79583	9.94987	8.66025	10.24695
8 %	7.68115	7.81025	4.89898	8.88819	10.95445	11.18034
9 %	9.05539	4.69042	9	9.74679	10.72381	11.61895
10 %	7.81025	6.16441	8.48528	10.63015	11.22497	11.6619
11 %	10.67708	8.66025	10.48809	11.61895	11.48913	12.24745

Tabla B.16: *Desviación estandar de la aptitud del mejor individuo de la serie normal con configuración con agente condicionado.*

Mutación \ Cruza	20 %	45 %	60 %	85 %	95 %	100 %
0 %	29.18904	23.5372	25.63201	18.70829	21.56386	18.38478
1 %	27.82086	10.48809	13.22876	6.48074	7.48331	7.61577
2 %	28.80972	9.53939	9	7.28011	5.74456	6.32456
3 %	33.30165	11.6619	6.9282	7.48331	2.44949	6.48074
4 %	34.62658	17.14643	5.47723	3.31662	5.09902	0
5 %	36.52396	19.13113	3.4641	0	2	3.4641
6 %	30.56141	7.34847	3.74166	1	3.74166	3.16228
7 %	33.91165	22.13594	6.63325	3.31662	1	0
8 %	34.75629	25.0998	11.57584	0	1	0
9 %	33.60059	22.04541	3.31662	1.41421	10.90871	1.73205
10 %	32.10919	21.81742	15.52417	3	3.16228	0
11 %	30.70831	18.3303	15.77973	0	10.86278	3

Tabla B.17: *Desviación estandar de la aptitud del mejor individuo de la serie neutral simple con configuración con agente condicionado.*

Mutación \ Cruza	20 %	45 %	60 %	85 %	95 %	100 %
0 %	26.30589	27.60435	29.73214	27.58623	28.44292	25.98076
1 %	28.98275	19.36492	10.04988	3.74166	6.16441	8.24621
2 %	33.66006	9.74679	8.88819	6.08276	1	2.44949
3 %	34	7.14143	9.21954	1	0	0
4 %	34.59769	17.34935	0	0	0	1.73205
5 %	33.18132	9.38083	0	0	0	4.24264
6 %	35.07136	17.77639	10.95445	0	0	0
7 %	36.09709	23.64318	10.86278	0	0	0
8 %	36.60601	25.15949	18.49324	10.86278	0	0
9 %	36.06938	29.76575	15.26434	0	0	0
10 %	35.25621	24.43358	12.8841	0	3	7.07107
11 %	33.18132	21.84033	18.22087	0	0	0

Tabla B.18: *Desviación estandar de la aptitud del mejor individuo de la serie neutral avanzada con configuración con agente condicionado.*

Mutación \ Cruza	**20 %**	**45 %**	**60 %**	**85 %**	**95 %**	**100 %**
0 %	30.78961	31.49603	27.76689	29.03446	26.13427	24.61707
1 %	30.39737	18.19341	5.19615	8.66025	5.19615	8.24621
2 %	31.76476	17.83255	7.14143	1.73205	0	0
3 %	33.15117	16.64332	11.31371	0	3.4641	0
4 %	33.10589	10.86278	0	10.86278	10.90871	0
5 %	35.35534	21.16601	1	3.4641	1	0
6 %	35.70714	23.38803	13.19091	0	11.18034	0
7 %	35.14257	19.89975	15.65248	0	0	0
8 %	33.80828	24.31049	0	10.86278	10.86278	0
9 %	33.45146	24.20744	18.76166	10.86278	0	0
10 %	34.0294	14.28286	19.33908	0	10.86278	10.86278
11 %	32.57299	27.29469	17.05872	0	0	0

Bibliografía

[1] A. Agogino and K. Tumer. Evolving distributed agents for managing air traffic. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1888–1895, London, 7-11 July 2007. ACM Press.

[2] S. B. Akers. Binary decision diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.

[3] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, New York, 2002.

[4] J. W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. In *IFIP Congress*, pages 125–131, 1959.

[5] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Publishers, 1987.

[6] W. Banzhaf. Genotype-phenotype mapping and neutral variation - a case study in genetic programming. In *Parallel Problem Solving from Nature III*, volume 866 of *Lecture Notes in Computer Science*, pages 322–332, October 1994.

[7] W. Banzhaf, P.Ñordin, R. Keller, and F. Francone. *Genetic programming - An Introduction*. Morgan Kaufmann, San Francisco, 1998.

[8] L. Barnett. Ruggedness and neutrality – the NKp family of fitness landscapes. In C. Adami, R. K. Belew, H. Kitano, and C. Taylor, editors, *Artificial Life VI: Proc. of the Sixth Int. Conf. on Artificial Life*, pages 18–27, Cambridge, MA, 1998. The MIT Press.

[9] L. Barnett. Netcrawling - optimal evolutionary search with neutral networks. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 30–37, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 2001. IEEE Press.

[10] T. A. Brown. *Genomes*. Oxford, 2002.

[11] T.-M. Chan, K.-S. Leung, and K.-H. Lee. TFBS identification by position- and consensus-led genetic algorithm with local filtering. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 1, pages 377–384, London, 7-11 July 2007. ACM Press.

[12] K. Chellapilla. Evolutionary programming with tree mutations: Evolving computer programs without crossover. In J. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 2007: Proceedings of the Second Annual Conference on Genetic Programming*, pages 432–438. MIT Press, Morgan Kaufmann, July 1997.

[13] C.-H. Chen and Y. ping Chen. Real-coded ECGA for economic dispatch. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1920–1927, London, 7-11 July 2007. ACM Press.

[14] C. S. Choo, C. L. Chua, and S.-H. V. Tay. Automated red teaming: A proposed framework for military application. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1936–1942, London, 7-11 July 2007. ACM Press.

[15] R. Chow. Evolving genotype to phenotype mappings with a multiple-chromosome genetic algorithm. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Part I*, volume 3102 of *Lecture Notes in Computer Science*, pages 1006–1017. Springer, 2004.

[16] S. Christensen and F. Oppacher. An analysis of koza's computational effort statistic for genetic programming. In *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*, pages 182–191, London, UK, 2002. Springer-Verlag.

[17] S. Christensen and F. Oppacher. Solving the artificial ant on the santa fe trail problem in 20,696 fitness evaluations. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A.

Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 1, pages 1574–1579, London, 7-11 July 2007. ACM Press.

[18] V. Ciesielski, D. Mawhinney, and P. Wilson. Genetic programming for robot soccer. In *RoboCup 2001: Robot Soccer World Cup V*, pages 319–324, London, UK, 2002. Springer-Verlag.

[19] M. Collins. Finding needles in haystacks is harder with neutrality. In H.-G. Beyer, U.-M. O'Reilly, D. V. Arnold, W. Banzhaf, D. Dasgupta, K. Deb, J. A. Foster, E. D. de Jong, H. Lipson, X. Llora, S. Mancoridis, A. M. Tyrrell, J.-P. Watson, and E. Zitzler, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and Evolutionary Computation*, volume 2, pages 1613–1618. ACM Press, June 2005.

[20] J. M. Daida. What makes a problem gp-hard? a look at how structure affects content. In R. L. Riolo and B. Worzel, editors, *Genetic Programming Theory and Practice*, chapter 7, pages 99–118. Kluwer, 2003.

[21] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[22] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, Ann Arbor, MI, USA, 1975.

[23] K. Deb and H.-G. Beyer. Self-adaptation in real-parameter genetic algorithms with simulated binary crossover. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 172–179, Orlando, Florida, USA, 1999. Morgan Kaufmann.

[24] D. Doherty and C. O'Riordan. A phenotypic analysis of GP-evolved team behaviours. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1951–1958, London, 7-11 July 2007. ACM Press.

[25] R. Downing. Neutrality and gradualism: Encouraging exploration and exploitation simultaneously with binary decision diagrams. In G. G. Yen, S. M. Lucas, G. Fogel, G. Kendall, R. Salomon, B.-T. Zhang, C. A. C. Coello, and T. P. Runarsson, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 615–622, Vancouver, BC, Canada, 2006. IEEE Press.

[26] R. M. Downing. Evolving binary decision diagrams with emergent variable orderings. In T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. M. Guervós, L. D. Whitley, and

X. Yao, editors, *PPSN*, volume 4193 of *Lecture Notes in Computer Science*, pages 798–807. Springer, 2006.

[27] M. Ebner, P. Langguth, J. Albert, M. Shackleton, and R. Shipman. On neutral networks and evolvability. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1–8, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 2001. IEEE Press.

[28] A. Engelbrecht. *Computational Intelligence: An Introduction*. Halsted Press, New York, NY, USA, 2002.

[29] K. Fan, A. Brabazon, C. O'Sullivan, and M. O'Neill. Option pricing model calibration using a real-valued quantum-inspired evolutionary algorithm. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1983–1990, London, 7-11 July 2007. ACM Press.

[30] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Trans. Neural Networks*, 5(1):3–14, 1994.

[31] C. M. Fonseca and M. B. Correia. Developing redundant binary representations for genetic search. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1675–1682, Edinburgh, UK, 2005. IEEE.

[32] F. D. Francone, L. M. Deschaine, and J. J. Warren. Discrimination of munitions and explosives of concern at F.E. Warren AFB using linear genetic programming. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1999–2006, London, 7-11 July 2007. ACM Press.

[33] R. M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13, 1958.

[34] R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part ii. *IBM Journal of Research and Development*, 3(3):282–287, 1959.

[35] E. Gálvan-López and R. Poli. An empirical investigation of how and why neutrality affects evolutionary search. In M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, F. Rothlauf, C. Ryan, and D. Thierens, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 2, pages 1149–1156, Seattle, Washington, USA, 2006. ACM Press.

[36] E. Galván-López and K. Rodríguez-Vázquez. The importance of neutral mutations in GP. In T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. M. Guervós, L. D. Whitley, and X. Yao, editors, *PPSN*, volume 4193 of *Lecture Notes in Computer Science*, pages 870–879. Springer, 2006.

[37] E. Galván-López, K. Rodríguez-Vázquez, and R. Poli. Beneficial aspects of neutrality in gp. In F. Rothlauf, editor, *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2005)*, Washington, D.C., USA, June 2005.

[38] S. Gerbex, R. Cherkaoui, and A. J. Germond. Optimal location of facts devices to enhance power system security. *IEEE Bologna PowerTec*, 3(3):7–14, 2003.

[39] P. Geremia, M. Poian, and S. Poles. Genetic optimization for yacht design. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2007–2012, London, 7-11 July 2007. ACM Press.

[40] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, (3):493–530, 1989.

[41] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 160–168, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[42] S. Hanna. Defining implicit objective functions for design problems. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2013–2020, London, 7-11 July 2007. ACM Press.

[43] K. Harries and P. Smith. Exploring alternative operators and search strategies in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 147–155, Stanford University, CA, USA, 1997. Morgan Kaufmann.

[44] I. Harvey and A. Thompson. Through the labyrinth evolution finds a way: A silicon ridge. In T. Higuchi, M. Iwata, and L. Weixin, editors, *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96)*, volume 1259 of *LNCS*, pages 406–422. Springer-Verlag, 1996.

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

[46] M. A. Huynen, P. F. Stadler, and W. Fontana. Smoothness within ruggedness: The role of neutrality in adaptation. *Proceedings of the National Academy of Sciences USA*, 93(1):397–401, January 1996.

[47] C. Igel and M. Toussaint. Neutrality and self-adaptation. *Natural Computing*, 2(2):117–132, 2003.

[48] Y. Ikeda and S. Tokinaga. Synthesis of multi-agent systems based on the co-evolutionary genetic programming by considering social learning and its applications to the analysis of artificial stock markets. *Journal of the Japan Society for Management Information*, 12(3):17–35, 2003.

[49] T. Ito, H. Iba, and S. Sato. Non-destructive depth-dependent crossover for genetic programming. In *EuroGP '98: Proceedings of the First European Workshop on Genetic Programming*, pages 71–82, London, UK, 1998. Springer-Verlag.

[50] D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang. Evolution as a theme in artificial life: The genesys/tracker system. In C. G. Langton, C. E. Taylor, D. J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 549–578. Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, 1991.

[51] L. J.Fogel. Autonomous automata. *Industrial Research*, 4(1):14–19, 1962.

[52] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[53] R. Jursa. Variable selection for wind power prediction using particle swarm optimization. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2059–2065, London, 7-11 July 2007. ACM Press.

[54] M. Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, 2004.

[55] R. E. Keller and W. Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 116–122, Stanford University, CA, USA, 1996.

[56] F. Kharbat, L. Bull, and M. Odeh. Mining breast cancer data with XCS. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2066–2073, London, 7-11 July 2007. ACM Press.

[57] M. Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, 1983.

[58] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, 1992.

[59] J. R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Stanford University, Computer Science Departament, California, USA, 1990.

[60] J. R. Koza. Genetic evolution and co-evolution of computer programs. In C. Taylor, C. Langton, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 603–629, Santa Fe Institute, New Mexico, USA, 1991. Addison-Wesley.

[61] A. Kuri-Morales and C. V. Quezada. A universal eclectic genetic algorithm for constrained optimization. In *Proceedings 6th European Congress on Intelligent Techniques Soft Computing, EUFIT'98*, pages 518–522, Aachen, Germany, 1998. Verlag Mainz.

[62] I. Kuscu. Evolving a generalized behaviour: Artificial ant problem revisited. In *EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII*, pages 799–808, London, UK, 1998. Springer-Verlag.

[63] W. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry, R. Roy, and R. K. Pan, editors, *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag, June 1997.

[64] W. Langdon and R. Poli. Better trained ants for genetic programming. Technical Report CSRP-98-12, University of Birmingham, School of Computer Science, April 1998.

[65] W. Langdon and R. Poli. Why ants are hard. In J. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, July 1998.

[66] P. Larranaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[67] X. Llorà, R. Reddy, B. Matesic, and R. Bhargava. Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2098–2105, London, 7-11 July 2007. ACM Press.

[68] A. J. Lockett, C. L. Chen, and R. Miikkulainen. Evolving explicit opponent models in game playing. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2106–2113, London, 7-11 July 2007. ACM Press.

[69] E. G. López and R. Poli. Some steps towards understanding how neutrality affects evolutionary search. In T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. M. Guervós, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, volume 4193 of *Lecture Notes in Computer Science*, pages 778–787. Springer, 2006.

[70] S. Luke and L. Spector. A revised comparison of crossover and mutation in genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 208–213, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[71] A. T. Machwe and I. C. Parmee. Integrating aesthetic criteria with evolutionary processes in complex, free-form design- an initial investigation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 165–172. IEEE Computer Society, 26-29 June 2006.

[72] G. J. Mendel. Experiments in plant hybridization. In C. I. Davern, editor, *Genetics: Readings from Scientific American*, chapter 1, pages 8–17. W. H. Freeman, San Francisco, 1981.

[73] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1992.

[74] B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[75] J. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. Langdon, J. Miller, P.Ñordin, and T. C. Fogarty, editors, *Third European Conference*

on Genetic Programming, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer-Verlag, 2000.

[76] N.Ñedjah and L. de Macedo Mourelle. Evolutionary digital circuit design using genetic programming. In N.Ñedjah, A. Abraham, and L. de Macedo Mourelle, editors, *Genetic Systems Programming: Theory and Experiences*, volume 13 of *Studies in Computational Intelligence*, pages 149–174. Springer, Germany, 2006.

[77] P.Ñordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.

[78] P.Ñordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, Tahoe City, California, USA, 1995.

[79] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Trans. Evolutionary Computation*, 5(4):349–358, 2001.

[80] R. Poli. Exact schema theorem and effective fitness for GP with one-point crossover. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 469–476, Las Vegas, Nevada, USA, 2000. Morgan Kaufmann.

[81] R. Ramirez and A. Hazan. Inducing a generative expressive performance model using a sequential-covering genetic algorithm. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2159–2166, London, 7-11 July 2007. ACM Press.

[82] I. Rechenberg. *Evolutionsstrategie – Optimierung technisher Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, GER, 1973.

[83] G. Reis and F. Fernandez Vega. Electronic synthesis using genetic algorithms for automatic music transcription. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1959–1966, London, 7-11 July 2007. ACM Press.

[84] R. G. Reynolds. An introduction to cultural algorithms. In A. Sebalk and L. Fogel, editors, *Proceedings of the Third Annual conference on Evolutionary Programming*, pages 131–139, River Edge NJ, 1994. World Scientific Publishing.

[85] E. Ricalde and K. Rodríguez-Vázquez. A gp neutral function for the artificial ant problem. In D. Thierens, editor, *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007, Companion Material*, pages 2565–2571. ACM, 2007.

[86] E. Ricalde and K. Rodríguez-Vázquez. Una función neutral en pg para el problema de la hormiga artificial. In F. Padilla, F. Álvarez, K. Rodríguez, A. Padilla, E. Ponce, and H. Sánchez, editors, *Memorias del 3er Congreso Mexicano de Computación Evolutiva (COMCEV'2007)*, pages 67–73. Universidad Autónoma de Aguascalientes, 2007.

[87] S. J. Ross, J. M. Daida, C. M. Doan, T. F. Bersano-Begey, and J. J. McClain. Variations in evolution of subsumption architectures using genetic programming: The wall following robot revisited. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 191–199, Stanford University, CA, USA, 1996. MIT Press.

[88] C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *EuroGP*, volume 1391 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 1998.

[89] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[90] R. Shipman, M. Shackleton, and I. Harvey. The use of neutral genotype-phenotype mappings for improved evolutionary search. *BT Technology Journal*, 18(4):103–111, 2000.

[91] T. Smith, P. Husbands, and M. O'Shea. Neutral networks in an evolutionary robotics search space. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 136–143. IEEE Neural Network Council (NNC), Evolutionary Programming Society (EPS), Institution of Electrical Engineers (IEE), IEEE Press, 27-30 May 2001.

[92] T. Soule and J. A. Foster. Code size and depth flows in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313–320, Stanford University, CA, USA, 1997. Morgan Kaufmann.

[93] C. R. Stephens. Effect of mutation and recombination on the genotype-phenotype map. In W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, pages 1382–1390. Morgan Kaufmann, 1999.

[94] C. R. Stephens. Effective fitness landscapes in evolution. In P. Angeline, editor, *Congress on Evolutionary Computation CEC 99*, pages 703–714. IEEE Press, 1999.

[95] C. R. Stephens, M.Ñicolau, and C. Ryan. Zero is ot a four letter word: Studies in the evolution of language. In M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, editors, *Genetic Programming, 8th European Conference, EuroGP2005*, volume 3447 of *Lecture Notes in Computer Science*, pages 371–380. Springer, 2005.

[96] C. R. Stephens and J. M. Vargas. Effective fitness as an alternative paradigm for evolutionary computation I: General formalism. *Genetic Programming and Evolvable Machines*, 1(4):363–378, 2000.

[97] C. R. Stephens and J. M. Vargas. Effective fitness as an alternative paradigm for evolutionary computation II: Examples and applications. *Genetic Programming and Evolvable Machines*, 2(1):7–32, 2001.

[98] C. R. Stephens and H. Waelbroeck. Effective degrees of freedom of genetic algorithms and the block hypothesis. In T. Bäck, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 34–41. Morgan Kaufmann, 1997.

[99] T. Stewart. Extrema selection: Accelerated evolution on neutral networks. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 25–29, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 2001. IEEE Press.

[100] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, California, USA, 1995.

[101] G. Sywerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[102] G. Tao and Z. Michalewicz. Inver-over operator for the TSP. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 803–812, London, UK, 1998. Springer-Verlag.

[103] C. Teuscher, E. Sanchez, and M. Sipper. Romero's Odyssey to Santa Fe: from Simulation to Real Life. *M. Jamshidi, A. Maciejewski, R. Lumia, S. Nahavandi, (Eds), Robotic and Manufacturing Systems. TSI Press Series, Albuquerque*, 10:262–267, 2000.

[104] M. Tomassini, L. Vanneschi, F. Fernández, and G. Galeano. A study of diversity in multipopulation genetic programming. In P. Liardet, P. Collet, C. Fonlupt, E. Lutton, and M. Schoenauer, editors, *Evolution Artificielle, 6th International Conference*, volume 2936 of *Lecture Notes in Computer Science*, pages 243–255, Marseilles, France, October 2003. Springer-Verlag.

[105] S. R. D. Torres, D. C. B. Romero, L. F.Ñ. Vasquez, and Y. J. P. Ardila. A novel ab-initio genetic-based approach for protein folding prediction. In D. Thierens, H.-G.

Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 1, pages 393–400, London, 7-11 July 2007. ACM Press.

[106] L. Vanneschi, Y. Pirola, P. Collard, M. Tomassini, S. Verel, and G. Mauri. A quantitative study of neutrality in GP boolean landscapes. In M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, F. Rothlauf, C. Ryan, and D. Thierens, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 895–902. ACM SIGEVO (formerly ISGEC), ACM Press, 2006.

[107] A. Wagner. Robustness, evolvability and neutrality. *FEBS Letters - Elsevier*, 579(8):1772–1778, March 2005.

[108] W. Yan and C. D. Clack. Evolving robust GP solutions for hedge fund stock selection in emerging markets. In D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F.Ñeumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 2234–2241, London, 7-11 July 2007. ACM Press.

[109] T. Yu and J. Miller. Neutrality and the evolvability of boolean function landscape. In J. F. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *Lecture Notes in Computer Science*, pages 204–217, Lake Como, Italy, April 2001. Springer-Verlag.

[110] T. Yu and J. Miller. Needles in haystacks are not hard to find with neutrality. In J. A. Foster, E. Lutton, J. F. Miller, C. Ryan, and A. Tettamanzi, editors, *Proceedings of the Fifth European Conference on GP*, volume 2278 of *Lecture Notes in Computer Science*, pages 13–25, Kinsale, Ireland, April 2002. Springer-Verlag.

[111] T. Yu and J. Miller. The role of neutral and adaptive mutation in an evolutionary search on the onemax problem. In E. Cantú-Paz, editor, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 512–519, New York, USA, July 2002. AAAI.

[112] A. Zhou. Enhanced emerging market stock selection. In R. L. Riolo and B. Worzel, editors, *Genetic Programming Theory and Practice*, chapter 18, pages 291–302. Kluwer, 2003.