SEP Tecnológico Nacional de México

Instituto Tecnológico de Tijuana
Departamento de Ingeniería Eléctrica y Electrónica

EVOLUCIÓN PROGRESIVA DE COMPLEJIDAD EN PROGRAMACIÓN GENÉTICA Y SUS EFECTOS EN EL FENÓMENO BLOAT

Trabajo de tesis presentado por:
Luis Muñoz Delgado
para obtener el grado de:
Maestro en Ciencias de la Ingeniería

Director de tesis:
Dr. Leonardo Trujillo Reyes
Co-Director de tesis:
Dr. Edgar Galván López

Tijuana, B.C. Diciembre del 2014

Agradecimientos

A mi familia.

A los buenos amig@s y educadores que en este largo camino que todavía no acabo, me ayudaron a aprender y a seguir adelante.

Y al Consejo Nacional de Ciencias y Tecnología (CONACYT) por brindarme la beca (No.372126) para el programa de maestría en ciencias de la ingeniería, sin la cual no seria posible la terminación de esta tesis.

Resumen

Comparar la Programación Genética (GP) dentro de la comunidad de aprendizaje máquina, es evidente que GP todavía sufre de algunos problemas teóricos y prácticos que han limitado su aceptación como una herramienta de resolución de problemas en general [14]. GP es visto como un método de búsqueda ineficiente, ya que cuenta con una gran sobrecarga de cálculos y produce soluciones complejas.

GP emplea una representación de longitud variable para representar soluciones, que es gravemente obstaculizada por los efectos del fenómeno de bloat, que se produce cuando los árboles de programas tienden a crecer innecesariamente grandes, sin un aumento correspondiente en la aptitud. De hecho, el bloat puede ser visto como la causa principal de los altos costos computacionales y la complejidad innecesaria en las soluciones de GP. Debido a esto, el bloat ha sido el tema de investigaciones que abarcan desde el análisis teórico hasta el control heurístico [25].

En la investigación de esta tesis se estudio el algoritmo de NeuroEvolución de Topologías en Aumento (NEAT NeuroEvolution of Augmenting Topologies), que utiliza la especiación para proteger nuevas topologías emergentes en la evolución de soluciones, y además promueve la evolución gradual de complejidad [27], se puede ejecutar libre de bloat, utilizando una configuración que favorezca la sobrevivencia de topologías pequeñas [28].

Validando las bases teóricas para desarrollar un algoritmo llamado neat-GP que es una versión simplificada del algoritmo NEAT original, que es adaptado para el paradigma de GP, que evoluciona árboles en vez de redes neuronales, diseñado para

inducir dinámicas de búsquedas similares al método de control de bloat Flat Operator Equalization (Flat-OE)[24].

Para validar que neat-GP no presenta bloat se probó el método en 9 problemas benchmark para regresión simbólica y 5 problemas reales de clasificación. Los resultados muestran que una búsqueda basada en neat-GP puede igualar y en casos superar a una búsqueda en GP, basado en el rendimiento de prueba y sobre todo en tamaño y profundidad de la solución.

El método de neat-GP tiene la ventaja de controlar el bloat sin incurrir en ningún coste computacional adicional, como los métodos actuales de control de bloat [4, 24, 26].

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Programación Genética (GP - Genetic Programming) es probablemente el paradigma más ambiciosos en el campo de la computación evolutiva (EC), ya que tiene como objetivo generar programas de cómputo a través de una búsqueda evolutiva, también llamada inducción automática de programas [21].

En su forma común, GP se puede entender como un algoritmo de aprendizaje supervisado que intenta construir una expresión válida sintácticamente usando un conjunto finito de funciones básicas y variables de entrada, guiado por un dominio dependiente de la función objetivo [21].

En la literatura de EC se encuentran varios ejemplos de éxito en las habilidades de generar soluciones a problemas utilizando GP, que ilustran la flexibilidad y robustez del paradigma [8]. De hecho, GP puede ser entendido como una hiper-heurística, con un enfoque algorítmico para la síntesis automática de los enfoques heurísticos, es una perspectiva que tiene impresionantes implicaciones teóricas y prácticas [19].

A pesar de su éxito, GP aún no se utiliza como una metodología de off-the-shelf [14], en la forma en que se utilizan Support Vector Machine (SVM) o Regresión Lineal. Esta falta de aceptación se deriva de algunas limitaciones pragmáticas importantes del enfoque básico de GP, que se discutirán a continuación.

En general, GP se caracteriza por dos rasgos principales que lo diferencia de otras técnicas del EC. En primer lugar, las soluciones evolucionadas representan expresiones sintácticas o programas válidos, que podrían ser utilizados como modelos, predictores, operadores o clasificadores. La capacidad de GP para construir expre-

siones sintácticas directamente, sin asumir un modelo previamente, permite generar soluciones interpretables, que no sólo resuelven el problema, sino que también proporciona una visión de dominio del problema.

En segundo lugar, GP utiliza un esquema de codificación de longitud variable, que se ha aplicado usando una variedad de estructuras de datos, tales como árboles, grafos y listas. Esta segunda característica puede ser vista como la fuente de la robustez y la flexibilidad de GP y también de algunos de sus principales deficiencias.

En particular, la búsqueda sintáctica tiende a ser ineficaz, debido a su pobre estructura local y los altos niveles de neutralidad [18] y [22].

Uno de los mayores temas de estudios en GP es el fenómeno de bloat (hinchazón), donde el tamaño de la población (para GP estandar la población esta compuesta de árboles tambien llamados programas) tienden a crecer innecesariamente sin un aumento de la aptitud [21, 25]. Por esta razón, algunos investigadores están considerando otros espacios que se muestrean simultáneamente durante la busqueda de GP, como el espacio de semántica [33] y espacio de conducta [16, 17, 30].

En cierto sentido, el bloat parece ser una consecuencia inevitable de la naturaleza del espacio de búsqueda en GP y la búsqueda impulsada por aptitud [9, 10].

Por otra parte, el bloat provoca varios efectos secundarios indeseables, como la evaluación de los programas grandes en la población requieren más tiempo de procesamiento, y soluciones de gran tamaño son difíciles de interpretar para los usuarios de GP. Por lo tanto, los investigadores han desarrollado una gran variedad de métodos de control de bloat, una revisión exhaustiva de los enfoques utilizados anteriormente es proporcionada por Silva en [25].

El presente trabajo es un novedoso enfoque hacia el control de bloat, aprovechando los conocimientos de los estudios recientes [24] y un algoritmo desarrollado originalmente para neuroevolución [27].

Silva sugiere en [24] que una estrategia de control de bloat es inducir una distribución uniforme del tamaño de los programas dentro de la población en evolución.

En particular [24], desarrolla el método de control de bloat Flat Operator Equalization (Flat-OE), que obliga explícitamente a la población a seguir evolucionando respetando una distribución específica de tamaño de programas.

Por otro lado, este trabajo utiliza los criterios del algoritmo Neuroevolution of

Augmenting Topologies (NEAT), que utiliza la especiación para proteger las nuevas topologías y promueve la evolución gradual de la complejidad [27].

El algoritmo propuesto es llamado *neat*-GP,que es una versión simplificada de la propuesta original de NEAT parcialmente adaptada al paradigma de GP, diseñada para inducir dinámicas de búsqueda similares a los mostrados por Flat-OE.

Se llevaron a cabo experimentos utilizando una representación basada en árbol y puestos a prueba en varios problemas de referencia de regresión simbólica y problemas reales de clasificación con *neat*-GP, dando resultados competitivos o superiores a GP estándar, basado en pruebas de rendimiento sobre todo en cuanto a tamaño de la solución y bloat.

Estos resultados corresponden con los reportados en [28], con la ventaja de que *neat*-GP no incurre en ningún costo computacional adicional como los métodos OE o Flat-OE [24, 26].

Capítulo 2

Planteamiento del problema

El bloat es considerado uno de los aspectos teóricos y prácticos más importantes en GP [18]. Normalmente se define como el aumento en el tamaño promedio del programa sin un aumento proporcional en la aptitud, es una definición difusa. Pero de una forma general, el bloat se produce cuando la mejor aptitud encontrada durante la búsqueda se estanca, mientras que el tamaño promedio de los programas (o el tamaño de la mejor solución) continúa creciendo, una discusión detallada sobre este tema se proporciona en [25, 26].

Esta tesis tiene como objetivo desarrollar un método de búsqueda en GP libre del fenómeno de bloat, inspirado en las ideas de Flat-OE y los principios algorítmicos generales de NEAT.

Se propone modificar el proceso de búsqueda efectuado en GP, incorporando técnicas del algoritmo NEAT, que enfatiza un proceso de evolución donde la complejidad de las soluciones incrementa gradualmente en base a las características del muestreo en el espacio de búsqueda.

Capítulo 3

Marco Teórico

En la inteligencia artificial, la programación genética es una metodología basada en algoritmos evolutivos, inspirada en la evolución biológica para desarrollar automáticamente programas de cómputo. Es una especialización de los algoritmos genéticos (GA - Genetic Algorithms) donde cada individuo es un programa de cómputo. También es una técnica de aprendizaje automático utilizada para optimizar una población de programas de acuerdo a una función de ajuste o aptitud (fitness function) que evalúa la capacidad de cada programa para llevar a cabo la tarea en cuestión.

3.1. Representación

GP desarrolla programas informáticos, tradicionalmente representados como estructuras de árbol, ver Figura 3.1. Los árboles pueden ser fácilmente evaluados de forma recursiva. Cada nodo del árbol tiene una función como operador y cada terminal tiene un operando, formando expresiones matemáticas, que pueden ser evolucionadas y evaluadas.

Representaciones que no utilizan árboles se han sugerido y aplicado con éxito, tales como programación genética lineal, la cual se adapta a los tradicionales lenguajes imperativos.

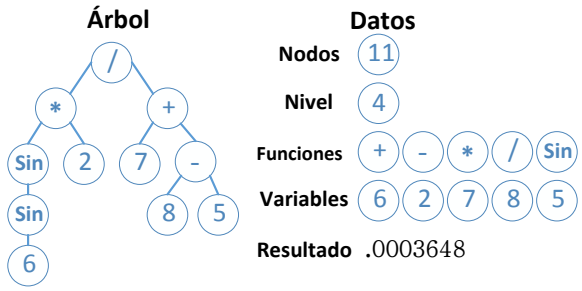


Figura 3.1: Ejemplo de un árbol y sus elementos, el resultado también conocido como aptitud o fitness

3.2. Ciclo evolutivo GP

El ciclo comienza comúnmente con una población aleatoria de programas, y cada programa de manera independiente se le llama individuo (haciendo juego con el termino de población), cada programa o individuo está representado por una estructura de árbol, compuesta por funciones que realizan operaciones y terminales con valores de entrada.

Generando una población aleatoria de programas, estos programas generan resultados en base a los valores de las terminales o variables de entrada para generar un valor de salida o resultado, ver Figura 3.2, que es utilizado para obtener el valor de aptitud o fitness. El valor de aptitud indica el rendimiento del programa y aquellos que tengan el mejor rendimiento seguirán dentro del proceso evolutivo y los que tengan el menor rendimiento serán descartados en los siguientes ciclos. Existen distintos criterios de selección de sobrevivencia de programas, donde la mejor aptitud no es prioridad, pero tales métodos estas fuera del alcance de la tesis.

Los programas con la mejor aptitud son seleccionados (los programas seleccionados son llamados padres), para entrar a la fase de reproducción, compuesta de dos posibles operadores cruce (recombination) y mutación (mutation), ver Figura 3.2. Los nuevos programas generados por el cruce y la mutación, también llamados descendencia o hijos, son anexados con los mejores de la población de padres, para formar una nueva población que repetirá el ciclo hasta encontrar la solución o bien hasta concluir una determinada cantidad de ciclos o generaciones.

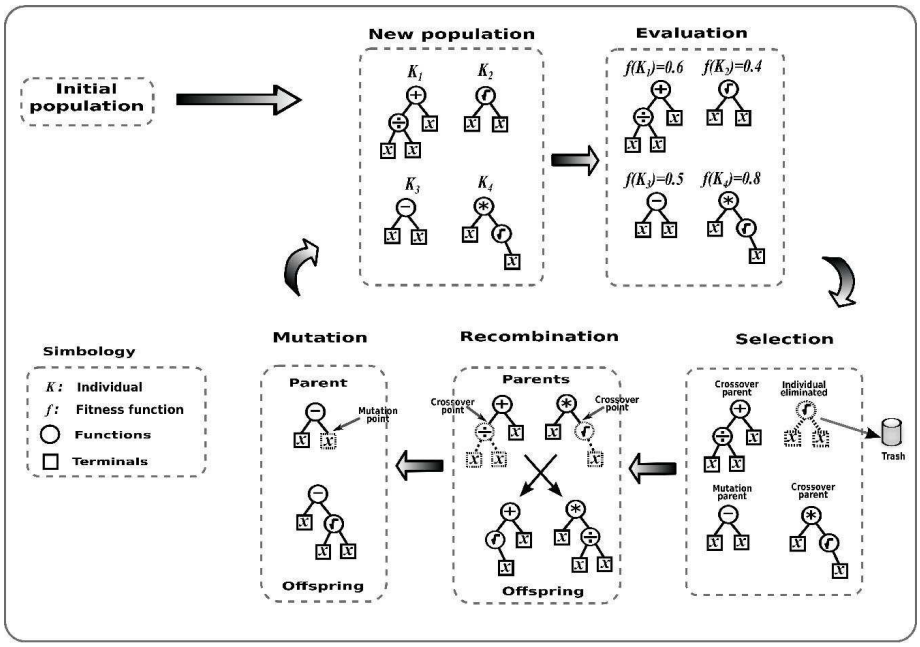


Figura 3.2: Ciclo evolutivo básico que sigue la programación genética, se comienza con una población inicial de programas aleatorios, que es evaluada para calcular la aptitud (fitness function) de cada programa, esto es necesario para la fase de selección donde los mejores programas que son aquellos con mejor aptitud, los cuales son seleccionados para la reproducción de nuevos programas a través de los operadores cruce y mutación. Estos nuevos programas generados por los operadores son introducidos a la población actual para remplazar a los peores programas (aquellos con aptitud baja) la cual forman la nueva población para la siguiente generación (un nuevo ciclo de búsqueda), hasta encontrar la solución o realizar una determinada cantidad de generaciones, ver Figura 3.2.

3.3. Operadores genéticos

Los principales operadores en GP son cruce y mutación.

3.3.1. Cruce

El cruce es aplicado mediante un intercambio de ramas entre dos árboles de la población, el punto de intercambio(nodo) es seleccionado aleatoriamente en cada árbol. Con una representación basada en árboles, la sustitución de un nodo implica la sustitución de toda la rama. Las expresiones resultantes del cruce son llamadas hijos o descendencia y los árboles utilizados en el cruce son llamados padres.

3.3.2. Mutación

En la mutación se selecciona un individuo de la población. Se genera un árbol aleatorio que es insertado al individuo en un nodo aleatorio. Para mantener la integridad, las operaciones deben ser protegidas de errores matemáticos, ciclos infinitos, entre otras posibilidades en el contexto de los operadores utilizados. Por ejemplo, debe tomar en cuenta sino deja operadores incompletos o forma operaciones incalculables.

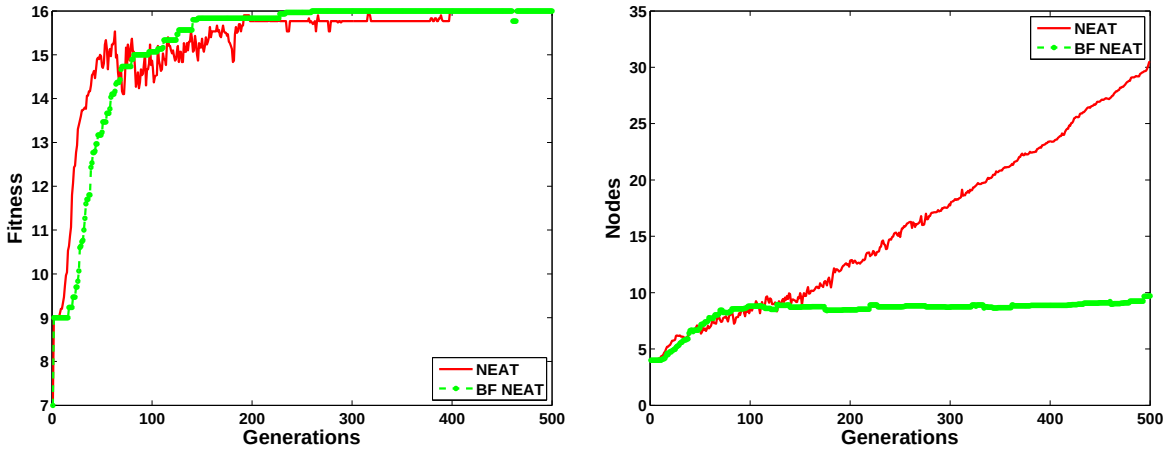
Capítulo 4

Bloat

A partir de la década de 1990, los investigadores comenzaron a notar en GP que además de aumentar progresivamente la aptitud del mejor individuo, la población muestran otras dinámicas. En particular, se detectó que muy a menudo el tamaño promedio (número de nodos) de los programas en una población, después de un cierto número de generaciones de tener una tendencia estática, comenzara a crecer a un ritmo acelerado. Normalmente, el aumento en el tamaño del programa no es acompañado por un aumento correspondiente en la aptitud, a este fenómeno se conoce como bloat.

Tomando en cuenta que el crecimiento del programa es algo esperado como parte del proceso de resolución del problema. Por ejemplo, GP normalmente inicia a partir de pequeños programas al azar, y puede ser necesario el crecimiento para que los programas sean más complejos para poder cumplir con las necesidades del problema y lograr una buena aptitud. Por lo tanto, no debemos comparar el crecimiento con bloat y debemos definir bloat como el crecimiento del programa sin un (significativo) retorno en términos (mejora) de aptitud ver Figura 4.1, la figura muestra dos ejecuciones de GP, una en rojo(o color negro) donde la aptitud (Fitness) alcanza una estabilidad pero los nodos (Nodes, gráfica a la derecha) sigue creciendo (teniendo bloat), en verde (o color gris) se muestra como los nodos se mantienen estables aun con el aumento de aptitud (libre de bloat).

El bloat tiene efectos prácticos importantes: programas grandes son computacionalmente costosos de evolucionar y su uso posterior, puede ser difícil de interpretar



(a) Aptitud (b) Tamaño

Figura 4.1: Ejemplos gráfico del fenómeno de bloat en rojo y en verde libre de bloat (Bloat Free). Del lado izquierdo el comportamiento de la aptitud y del lado derecho el crecimiento de tamaño (en número de nodos), en un problema de maximización

y generar soluciones poco robustas. Por estas razones, bloat ha sido un tema de intenso estudio en GP. Con los años, se han propuesto muchas teorías para explicar diversos aspectos del bloat, se han logrado grandes avances, pero todavía no existe una única teoría unificadora universalmente aceptada, para explicar la amplia gama de observaciones empíricas.

4.1. Teorías sobre bloat

Existen muchas teorías sobre el bloat, pero solo veremos aquellas relacionadas con el objetivo de esta tesis (un método para el control de bloat).

La teoría más establecida con respecto a las causas del bloat es la teoría que la aptitud causa bloat (Fitness Causes Bloat Theory - FCBT), desarrollado por Langdon y Poli [9].

FCBT se basa en el hecho de que:

a) Existe una correlación de muchos a uno desde el espacio sintáctico al espacio semántico o aptitud.

b) Hay exponencialmente más programas grandes con una aptitud deseada (buena) y pocos programas pequeños con la aptitud deseada.

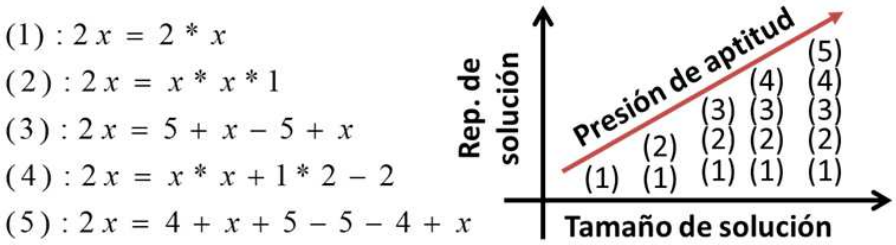


Figura 4.2: Representación de espacio semántico al espacio sintáctico, a la izquierda se muestra la forma semántica para este ejemplo son ecuaciones que puede ser formadas por los árboles y entre mas nodos tenga tenga el árbol mas posibilidades de representar la misma solución de una forma sintáctica, si se ejerce presión de aptitud la selección favorece a ábroles de mayor numero de nodos ya que estos tiene la ventaja de representar un buen resultado de distintas formas.

Por lo tanto, si se desea un valor de aptitud particular, existe una tendencia hacia programas más grandes o con bloat, durante una búsqueda de GP, simplemente porque hay más programas grandes(con respecto al número de nodos en el árbol) con buena aptitud que pequeños con buena aptitud. De hecho, es posible afirmar que la búsqueda de una buena aptitud es la principal causa de bloat, es ya indiscutible, ya que es básicamente el factor que subyace en todas las grandes teorías de bloat [25]. Por otra parte, trabajos recientes sugieren que una búsqueda en GP que no considera explícitamente la aptitud de hecho, puede evitar el bloat mediante la promoción de la soluciones novedosas (novelty search) [11, 30].

En la actualidad, una de las teorías más útiles que describen el proceso de bloat como es la teoría cruce sesgado (Crossover Bias Theory - CBT), desarrollado por Dignum y Poli [20]. Centrándose en GP estándar estilo Koza, el CBT establece que el bloat se produce por el efecto que la mutación de sub árbol tiene en la distribución de tamaño del árbol. Mientras que el tamaño promedio de los árboles no se ve afectada por el cruce, la distribución del tamaño si es afectada. El cruce de sub árbol produce un gran número de árboles pequeños, que para la mayoría de los problemas logra una baja aptitud. Por lo tanto, la selección favorecerá árboles con un numero de nodos mayores al promedio en la población, causando un aumento en el tamaño promedio de árbol dentro de la población Figura 4.3, ciclando efectivamente el efecto de bloat.

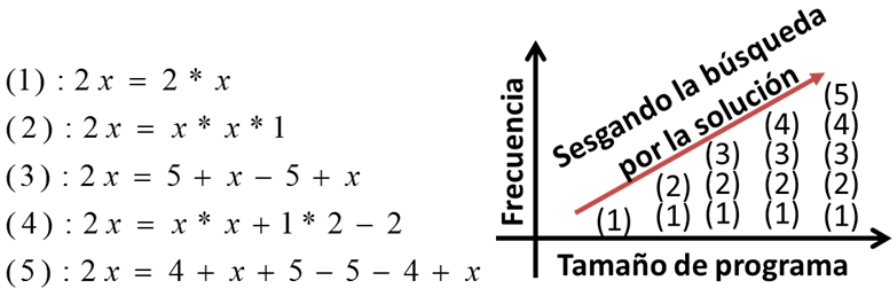


Figura 4.3: Los individuos pequeños generalmente no cuentan con la aptitud necesaria para dar solución al problema. Lo cual favorece a individuos de gran tamaño ya que tiene la complejidad necesaria para obtener mejores resultados o aptitud. Causando la proliferación de individuos con un gran numero de nodos, que al al aplicar el operador de cruce perpetuán el crecimiento de la población.

4.2. Métodos anti bloat

Teniendo en cuenta los conocimientos proporcionados por el CBT, Dignum y Poli [4] propusieron el método de control de bloat ecualización de operador (Operator Equalisation - OE), que se centra en controlar explícitamente la distribución de tamaño de los programas en cada generación. OE ha producido resultados impresionantes en varios problemas de referencia y problemas del mundo real, y la propuesta inicial ha dado lugar a una familia de métodos relacionados [26].

4.2.1. Criterio de OE

- Se define un rango de posibles tamaños que la población podrá tomar y respetar por todas las generaciones.
- El rango de posibles tamaños es dividido en recipientes o contenedores que contendrán los individuos de la población en base al tamaño de cada individuo.
- La cantidad de individuos que cada contenedor o recipiente puede contener es predefinida por el tipo de distribución que se desea seguir Figura 4.4.
- Si se genera un individuo fuera de rango de tamaño aceptado es rechazado.

- Si algún recipiente o contenedor, no tiene más espacio para aceptar un nuevo individuo de tamaño aceptable, solo lo aceptara si tiene la mejor aptitud en todo el contenedor,por lo tanto desecha el individuo de peor aptitud del contenedor o recipiente para aceptar el nuevo individuo.

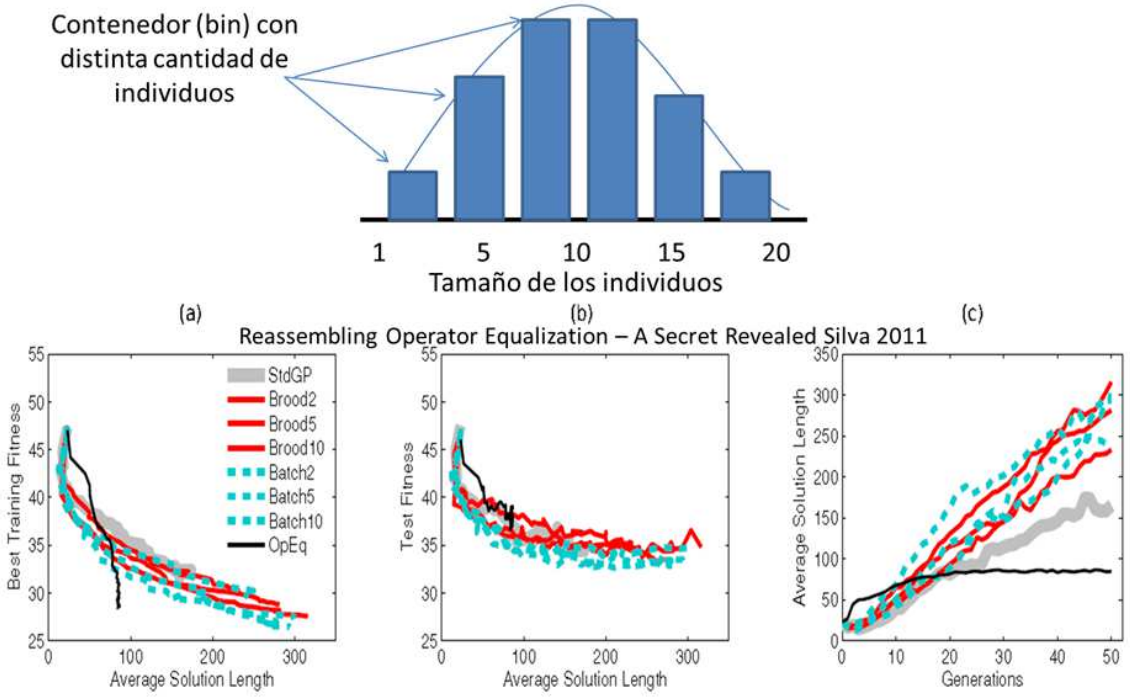


Figura 4.4: Ejemplo de distribución de recipientes y resultados obtenidos por Silva 2011, En la imagen superior se muestra la distribución de contenedores utilizada por OE, en las imágenes inferiores a,b y c, se muestra el método de OE vs otros métodos de GP, donde OE obtiene una aptitud competitiva sin un crecimiento de tamaño promedio en la poblacioñ.

A pesar del éxito de OE, hay varias preocupaciones con el método. Por ejemplo, se basa en un proceso computacionalmente costoso al generar, evaluar y en muchos casos rechazando programas que no encajan en la distribución deseada lo que es un desperdicio de recursos y posibles soluciones. Por otra parte, los resultados de Silva sugieren que algunos de los supuestos que subyacen en OE no pueden justificarse.

Silva intentó desarrollar un método OE simplificado, con un costo computacional menor, que al mismo tiempo que preservara las propiedades principales de OE [24].

Sin embargo, la aproximación simplificada propuesta fracasó en controlar el bloat. La razón de esto parece estar en una contradicción de OE con CBT. Dado que CBT sugiere que los programas pequeños son perjudiciales para el proceso evolutivo, y el método de OE tiende a promover distribuciones de tamaño objetivo que excluyen a dichos programas. Sin embargo, mientras que los métodos están diseñados para promover este tipo de distribuciones, en la práctica OE no encaja realmente. De hecho, [24] muestra que OE tiende a producir distribuciones uniformes o planas, con una proporción similar de árboles de diferentes tamaños, desde pequeños a grandes árboles. Esto motivo a Silva para proponer la variante Flat-OE (Flat Operator Equalisation)[24], donde se busca una distribución plana de tamaño de árboles en la población, y los resultados experimentales sugieren que Flat-OE puede controlar bloat sin comprometer la calidad de las soluciones desarrolladas, ver Figura 4.5.

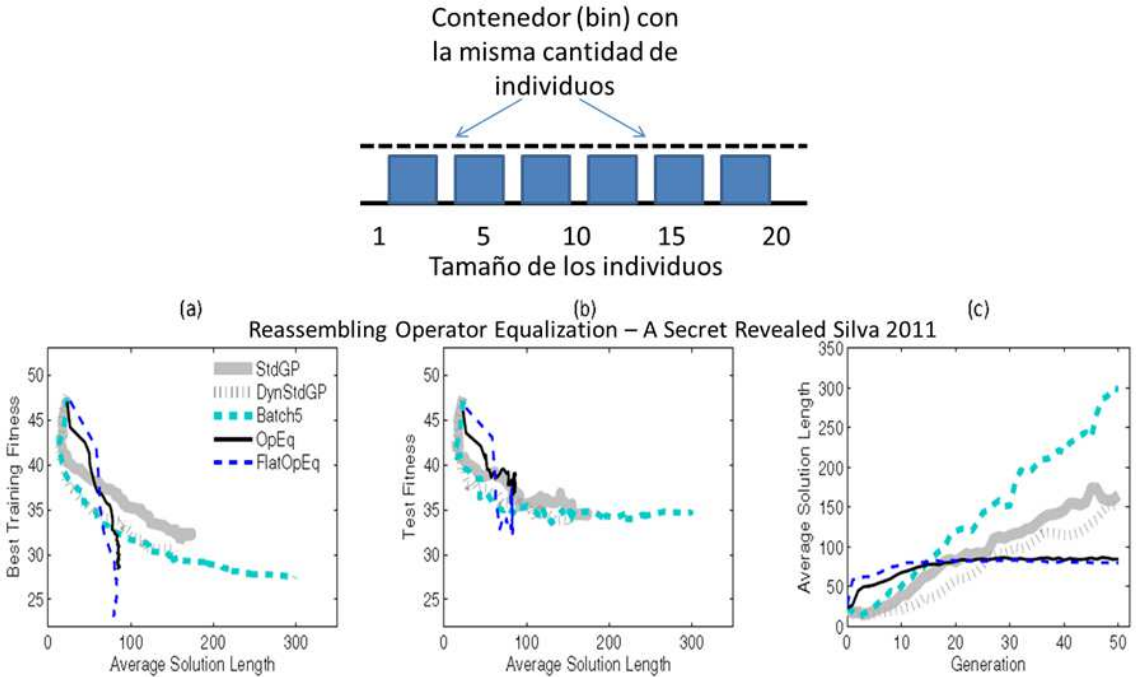


Figura 4.5: La imagen superior muestra a Flat-OE que sigue el mismo criterio de OE, la única diferencia es que los recipientes contiene el mismo número de individuos y tiene la capacidad de generar nuevos recipientes fuera de la distribución preestablecida. Las imágenes inferiores a,b y c, son los resultados obtenidos por Silva 2011, donde la nueva version del OE es papas de general mismos resultados son comprometer la calidad de la solución

La estrategia original de Silva en [24] parece bastante razonable: determinar las propiedades fundamentales de OE y desarrollar un algoritmo aproximado que satisfaga las propiedades. La propuesta consistía en utilizar la recombinación de cría (Brood recombination) [1] y límites dinámicos [25] como los ingredientes generales que forman a OE. Aunque esta combinación fracaso, lo hizo debido a los supuestos que se creía que haciendo OE en realidad estaban equivocados.

Sin embargo, los méritos generales de la estrategia requieren más atención; es decir, desarrollar una versión aproximada de OE que se basa en métodos computacionalmente sencillos.

Capítulo 5

NEAT

En esta sección se trata de reproducir el control de distibución de la población del algoritmo original OE, pero con un objetivo centrado en Flat-OE. Para ello, se propone dar un paso fuera del marco tradicional de GP, y el estudio de un algoritmo que se utiliza ampliamente en otras áreas del EC. Utilizando un método para neuroevolución (NE), que permite la evolución artificial de las redes neuronales utilizando GA llamado Neuro Evolution Augmenting Topologies (NEAT), desarrollado por Stanley y Miikkulainen [27].

Se propone modificar el proceso de búsqueda efectuado en GP, incorporando técnicas del algoritmo NEAT, que enfatiza un proceso de evolución donde la complejidad de las soluciones incrementa gradualmente en base a las características del muestreo en el espacio de búsqueda. La propuesta es una simplificación de la esencia del método de OE que ha demostrado la eliminación del Bloat, a través de un muestreo uniforme del espacio de tamaño de programas, ver Figura 5.1.

5.1. Algoritmo

Es un algoritmo genético de longitud variable que codifica explícitamente la topología y pesos de conexión de redes neuronales (NNets). NEAT puede ser dividido en las siguientes características o componentes principales.

La principal característica de NEAT es una representación especial de longitud variable. El genoma se divide conceptualmente en dos segmentos: el primero contiene

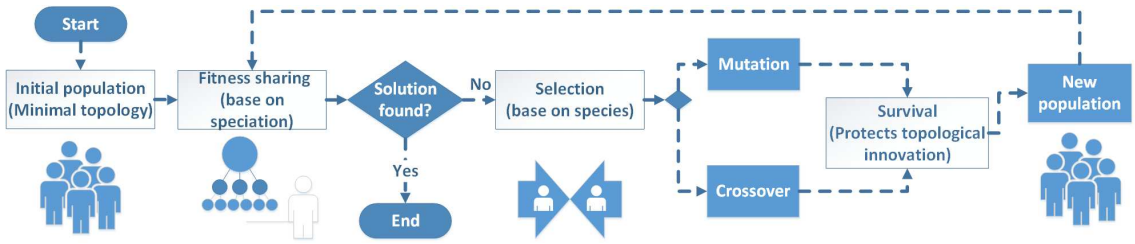


Figura 5.1: Flujo general de un algoritmo genético con NEAT

los genes de nodos que especifican el conjunto de entrada, salida y nodos ocultos de la red; mientras que el segundo segmento contiene el conjunto de genes de conexión o sinapsis que especifican el nodo de entrada y de salida de cada conexión y su respectivo peso.

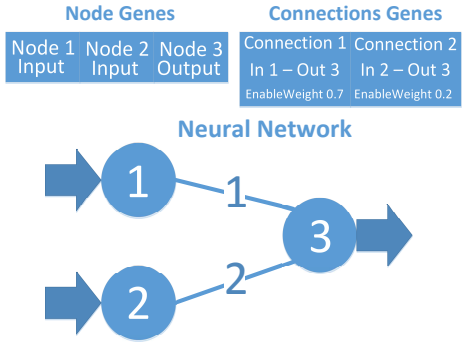


Figura 5.2: Representación de redes neuronales en NEAT

Una cosa que NEAT no evoluciona son las funciones de activación, esto se establecen a priori para todos los nodos ocultos y de salida, la representación de una red neuronal para NEAT se muestra en la Figura 5.2.

La segunda característica de NEAT es que está diseñado para buscar NNets, con representación de longitud variable codificadas en estructuras gráficas.

Para permitir que las operaciones de búsqueda sean coherentes entre NNets de diferentes tamaños, NEAT incluye marcas históricas para cada conexión sináptica, cuando se lleva a cabo el cruce entre dos NNets, pueden tener diferentes topologías de red, los genes de conexión son alineados primero sobre la base de estas marcas históricos, identificando la estructura compartida entre ambos padres (nodos y conexiones). Y poder realizar una operación de cruce entre los genes que coincidan entre

dos padres, se puedan heredar de forma aleatoria de cualquier padre, y los genes disjuntos o en exceso se heredan del más apto de los padres. Por otra parte, los pesos de conexión también se heredan después de una operación de cruce, ver Figura 5.3.

Otros operadores de búsqueda que utiliza es la mutación, que es una mutaciones estructural que puede agregar nodos o conexiones a la NNets, ver Figure 5.4. Este conjunto de operadores de búsqueda es inusual, en el sentido de que producen descendencia que siempre es mayor o igual a los padres, lo que promueve el crecimiento de tamaño.

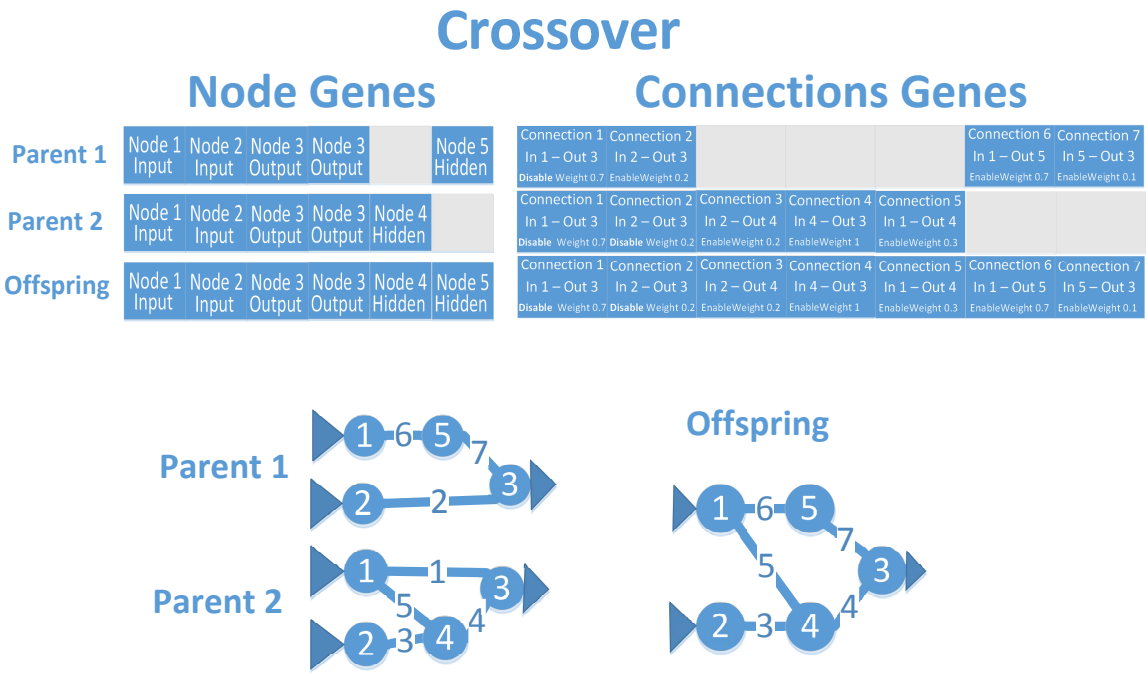
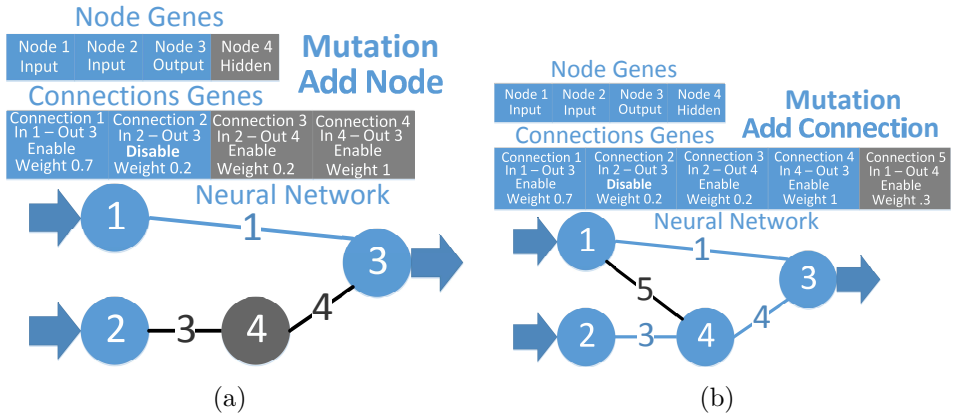


Figura 5.3: Cruce NEAT

La tercera característica de NEAT alienta una evolución gradual de complejidad en la solución (en base a tamaño). Ya que la población inicial sólo contiene NNets que comparten la misma topología mínima. Que en la mayoría de los casos se trata de una NNets feedforward totalmente conectada sin neuronas ocultas y con pesos aleatorios en las conexiones. NEAT asume que la mejor manera de realizar la búsqueda es comenzando con NNets más simples, y progresivamente aumentar la complejidad a través de los operadores de búsqueda. De esta manera, si el problema puede ser resuelto por una NNet simple de tamaño mínimo la búsqueda podría ser capaz de

encontrarla al inicio de la búsqueda, y sólo si no la encuentra, progresivamente buscar en topologías más grandes.



(a) (b)

Figura 5.4: Mutación NEAT para agregar nodos a) y conexiones b)

Por último, NEAT incorpora un plan para proteger la innovación topológica. Al comienzo de la ejecución todos los individuos comparten la misma estructura inicial, y la búsqueda se centra en la mejora de los pesos de conexión de las redes. Y los operadores de búsqueda añaden progresivamente elementos estructurales (nodos o conexiones) a la topología base, y cada vez que se genera un nuevo nodo aleatorio los pesos de conexión preservan los valores anteriores y solo son aleatorios para mutaciones de pesos.

Esto provoca un problema, porque una conexón generada aleatoriamente con un peso fijado al azar, tendrá con alta probabilidad un efecto destructivo sobre la aptitud. Sin embargo, es razonable suponer que un aumento de la complejidad estructural puede ser necesario para resolver problemas difíciles. Por lo tanto, es necesario proteger estas innovaciones estructurales para no ser descartadas por la fase de selección en el algoritmo genético.

NEAT logra proteger la innovación mediante el uso de la especiación en base a similitudes topológicas y aptitud compartida [6], donde la aptitud de cada individuo es penalizado en función de su similitud con otros individuos.

El elemento clave de NEAT es el uso de una medida de distancia δ, basado en las similitudes topológicas entre dos NNets expresadas por el número de genes disjuntos D y genes en exceso G, dado por

$$\delta = \frac{c_1 \cdot G + c_2 \cdot D}{N} + c_3 \cdot \overline{W} \ , \tag{5.1}$$

Donde $\overline{W}$ es la diferencia de peso promedio de los genes semejantes, N es el número de genes en el genoma más grande, y c_x son parámetros lineales para ajustar la importancia de los genes (a cual se le asigna más importancia los genes exceso c_1, en disjuntos c_2 o la diferencia de pesos promedio c_3).

La agrupación de NNets es en base a similitud topológica (todas las NNets que compartan la misma topológia se les puede llamar una especie de la población de NNets) que permita aplicar una penalización de aptitud compartida, de una forma más específica a grupos de NNets que por su buena aptitud pudieran apoderarse de toda la población limitando la posibilidad de optimización de otras topologías emergentes de NNets.

Fue necesario explicar el algoritmo de NEAT y sus características esenciales, ya que al entender su operación se puede apreciar como un programa genético para NNets.

5.1.1. Beneficios

La forma de operar del algoritmo puede promover distribuciones de tamaño en las NNets de la población. Y como lo vimos en la sección de OE y Flat-OE, el control de distribuciones de tamaño en la población puede ser utilizado como un control de bloat. Donde NEAT sea capaz de conservar distintas distribuciones de tamaño dentro de la población sin que algún tamaño sobre poblé la poblacón dejando poco espacio para cualquier otra distribución, por medio de métodos simples y conocidos. Nos lleva a suponer que el algoritmo de NEAT tiene la capacidad de realizar búsquedas sin el fenómeno de bloat.

5.2. Relación NEAT con Flat-OE

Se puede argumentar que la agrupación de NNets basado en la topología no es la mejor manera de promover un conjunto diverso de conductas funcionales [31]. Sin embargo, la especiación basada en similitudes topológicas puede promover una po-

blación diversa de topologías de NNets, es decir, una población de diversas formas de red y lo más importante de tamaños. Por lo tanto, NEAT proporciona una aproximación algorítmica prometedora a la estrategia principal que subyace Flat-OE. Si la observación anterior es correcta, entonces deberíamos esperar NEAT evolucionando libre de bloat.

El enfoque del trabajo experimental, es evaluar si el algoritmo de NEAT está libre de bloat aplicando problemas de referencia estándar (benchmark).

Antes de pasar a la configuración experimental y los resultados, vamos a contextualizar NEAT dentro del paradigma más amplio GP.

GP se puede definir como un algoritmo evolutivo de longitud variable que evoluciona expresiones sintácticas que se interpretan como programas sencillos, funciones, operadores o modelos. Por otro lado, NEAT es un algoritmo evolutivo de longitud variable que evoluciona estructuras gráficas que representan NNets. A partir de esta perspectiva general, se puede afirmar que NEAT es una forma especial de GP, donde las soluciones evolucionadas expresan instancias de una clase limitada de funciones, las que se pueden representar como NNets. Sin embargo, los resultados obtenidos con NEAT pueden proporcionar información útil al paradigma GP, al igual que cualquier otro trabajo experimental hecho con otras variantes comunes de GP.

Adicionalmente, es importante mencionar que anteriormente se a hipotetizado que NEAT podría intrínsecamente controlar bloat,(Por ejemplo, se especuló que el operador de cruce de alguna manera desalienta el bloat, sin embargo, esto es incorrecto, ya que en todos los casos de cruce producirá descendencia que es al menos tan grande como sus padres, que también incurre en una disminución de aptitud debido a las modificaciones estructurales (cruce y mutación) que tienden a ser altamente destructiva en la descendencia. La especiación también se ha identificado como un posible mecanismo de control de la bloat, sin embargo, como se mostrará en el trabajo experimental, la aplicación de NEAT con la configuración original presenta bloat, y se debe tener cuidado para parametrizar adecuadamente NEAT. Por lo tanto, los experimentos realizados son los primeros en identificar una configuración libre de bloat en NEAT, y el primero en evaluar exhaustivamente y explícitamente NEAT, desde la perspectiva del fenómeno de bloat.

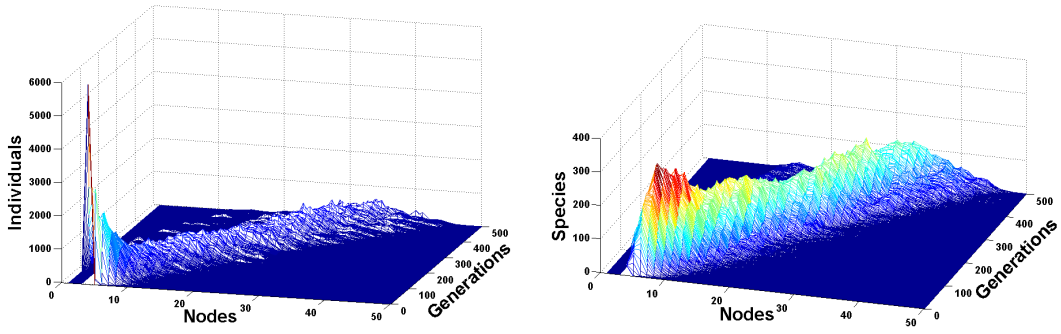
5.3. Experimentos

El objetivo de los siguientes experimentos es determinar si NEAT puede evolucionar sin bloat. Como NEAT es un algoritmo bastante complejo, utilizamos una librería desarrollada en Java con licencia libre (http://nn.cs.utexas.edu/?jneat) que se apega a NEAT original de [27], ya que actualmente existen muchas versiones para distintos lenguajes de programación. Si bien hay muchas implementaciones de NEAT disponibles, muchas de estas son variantes modificadas o simplificadas del algoritmo original, y no es fácil determinar cuáles son las consecuencias de estas modificaciones, no importa cuán pequeños sean, pueden tener efectos sobre la dinámica de búsqueda.

Para evaluar el algoritmo, se utilizan dos problemas de referencia(benchmark), el problema XOR y Parity de 3 bits, que son parte de la librería de NEAT (jneat) utilizada. Se analiza el rendimiento de NEAT utilizando la aptitud y el tamaño (número de nodos) total de las soluciones. Para analizar visualmente el comportamiento NEAT, los resultados se presentan utilizando tres tipos de gráficas. Las primeras gráficas en las Figuras [5.5,5.6,5.7,5.8] muestran cómo la distribución del tamaño a través del tiempo, teniendo en cuenta el número de individuos y de especies de un determinado tamaño que están presentes dentro de la población en cada generación, en el caso de tamaño de las especies, esto es determinado por el promedio de todos los individuos que pertenecen a una especie. En segundo lugar, las gráficas de convergencia clásicas de la mejor solución en base aptitud y tamaño por generación, ver Figuras [5.9,5.10], para comparar diferentes configuraciones del algoritmo. Por último, todos los resultados obtenidos son promedios de treinta corridas independientes.

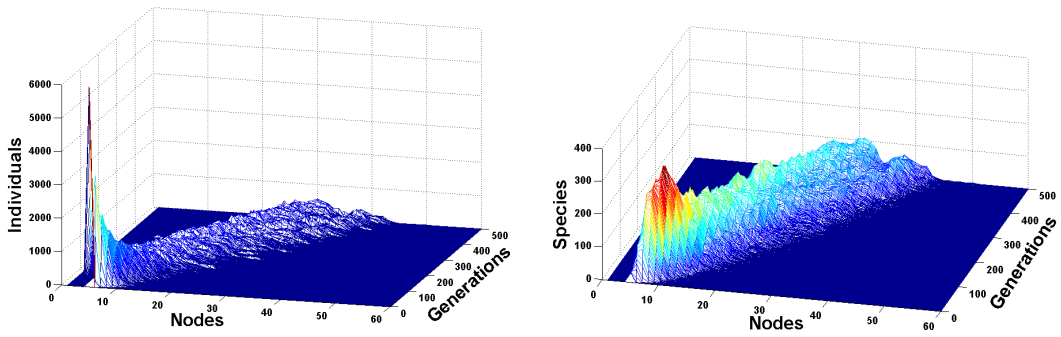
Una de las principales limitaciones de NEAT es que utiliza un amplio conjunto de parámetros. Por lo tanto se prueba utilizando los parámetros propuestos en [27], véase la segunda columna de la Tabla 5.1. La Figura 5.5 presenta gráficas que resumen el comportamiento sobre el problema XOR, teniendo en cuenta la distribución de tamaños en cada generación. La Figura 5.5(a) muestra la distribución de los individuos basado en el número de nodos y la Figura 5.5(b) muestra la distribución de especies. En primer lugar, la distribución del tamaño está sesgada hacia programas más grandes en cada generación ya que en las primeras generaciones se comienza con una población pequeña de tamaño que comienza a crecer sin detenerse o encontrar estabilidad en todo el transcurso de las generaciones, este fenómeno sucede

porque pequeños programas son reemplazados rápidamente por otros más grandes con mejor aptitud, esto genera el efecto de bloat evidente durante la corrida. Del mismo modo, la Figura 5.6 presenta el mismo análisis para el problema de parity, que exhibe tendencias similares al presentar una distribución creciente en el tamaño de la población.



(a) Individuals distribution (b) Species distribution

Figura 5.5: Evolución de tamaño para NEAT en el problema XOR: a) La distribución de tamaño de los individuos basados en el número de nodos; b) La distribución de tamaño de las especies basada en el número de nodos.



(a) Individuals distribution (b) Species distribution

Figura 5.6: Evolución de tamaño para NEAT en el problema de parity: a) La distribución de tamaño de los individuos basados en el número de nodos; b) La distribución de tamaño de las especies basada en el número de nodos.

Al principio, estos resultados sugieren que NEAT no puede promover una distribución de tendencia plana de tamaño de programas; es decir, no puede ser ejecutado libre de bloat. Por lo tanto es necesario reparametrizar algunos parámetros clave de

Parameters	**NEAT**	**BF-NEAT**
Weight mutation power	2.5	9.0
Recurrent probability	1.0	0.1
Disjoint parameter c_1	1.0	0.5
Excess coefficient c_2	1.0	1.0
Weight coefficient c_3	0.4	0.0
Compatibility threshold δ_{min}	3.0	9.0
Age significance	1.0	9.0
Survival threshold	0.2	0.8
Population size	200	200
Drop-off age	50	1000

Tabla 5.1: Parámetros utilizados en los experimentos para NEAT y Bloat-Free NEAT (BF-NEAT).

NEAT, como se muestra en la tercera columna de la Tabla 5.1, esta configuración se llama NEAT libre de bloat o Bloat Free NEAT (BF NEAT).

Los principales cambios se justifican de la siguiente manera.

- En primer lugar,(Weight coefficient) $c_3 = 0$ porque $\overline{W}$ no es relevante para realizar la especiación necesita que promueva la diversidad de tamaño de los individuos.

- En segundo lugar,(Disjoint parameter) c_1=0.5 y (Excess coefficient) c_2=1.0, centrando la condición de especiación en el número de exceso de genes, que son los más relevantes cuando se considera el crecimiento tamaño en nodos.

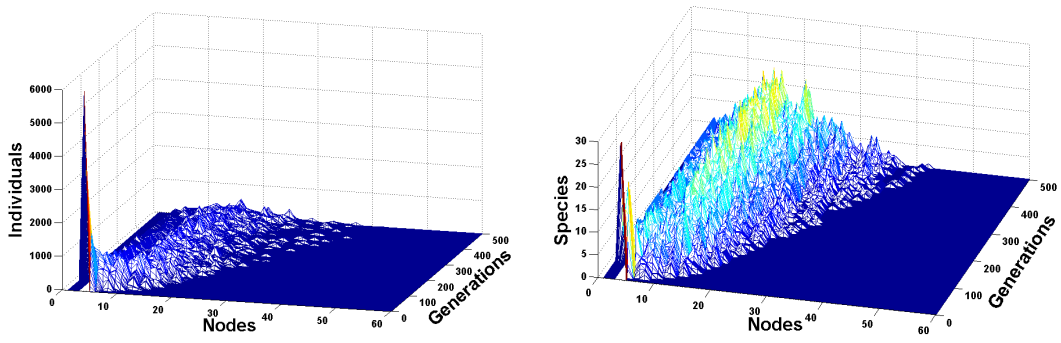
- En tercer lugar, el poder de mutación(Weight mutation power), que controla la rango de posibles valores para el pesos de las conexiones, esto se incrementó a 9, lo que permite una búsqueda a fondo en cada topología de NNet.

- En cuarto lugar, la importancia de la edad(Age significance), se incrementa con respecto a la implementación original, para proteger nuevos individuos y por ende nuevas especies. Teniendo en cuenta que al hacerlo estimula ligeramente el crecimiento del programa, ya que los operadores de búsqueda siempre producirán NNets grandes.

- En quinto lugar, el umbral de supervivencia(Survival threshold) que define el porcentaje de los mejores individuos que pueden reproducirse, utilizando un valor pequeño hace que el proceso de búsqueda sea codicioso.

- En sexto lugar, el parámetro de edad que elimina especies(Drop-off age) de bajo rendimiento o estancados cada cierto número de generaciones; Sin embargo, de acuerdo con Flat-OE la diversidad de tamaño nunca debe perderse durante la corrida.

Con estas modificaciones, los mismos experimentos fueron repetidos para BF-NEAT, evaluando el rendimiento en XOR y parity, los resultados se muestran en las Figuras 5.7 y 5.8. En este caso, los efectos de los nuevos parámetros son claros, se obtiene una distribución que tiende a ser plana de tamaños y el efecto de bloat se vea sustancialmente reducido. Además, las Figuras 5.9 y 5.10 comparan el rendimiento de NEAT y BF-NEAT, en los problemas XOR y parity respectivamente. Las Figuras 5.9(b) y 5.10(b) muestran cómo evoluciona el tamaño de la solución, y las Figuras 5.9(a) y 5.10(a) muestran cómo evoluciona la mejor aptitud. A partir de estos resultados es posible afirmar que el tamaño de la solución se reduce en gran medida en BF-NEAT, sin comprometer la calidad de la solución.

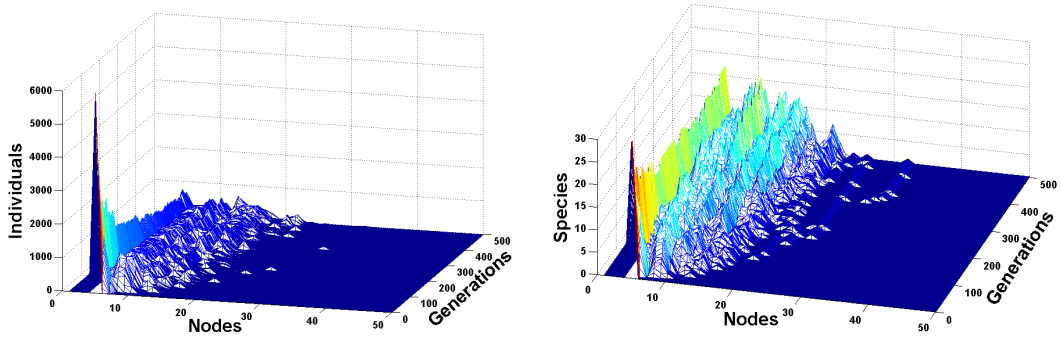


(a) Individuals distribution (b) Species distribution

Figura 5.7: Evolución de tamaño para BF-NEAT en el problema XOR: a) La distribución de tamaño de los individuos basados en el número de nodos; b) La distribución de tamaño de las especies basada en el número de nodos.

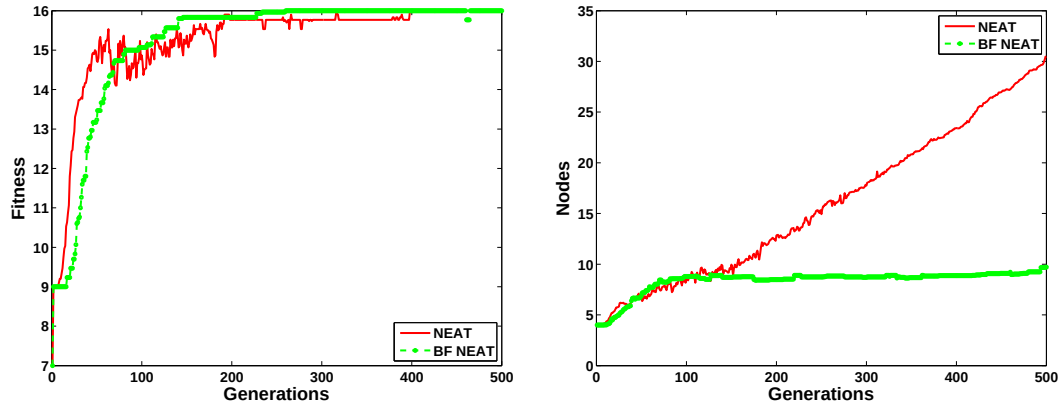
5.3.1. Análisis

En general, los resultados iniciales son alentadores, NEAT se pueden ejecutar sin bloat al favorecer una distribución con una tendencia uniforme en el tamaño en la población. Ahora analicemos cuáles podrían ser las principales causas de estos



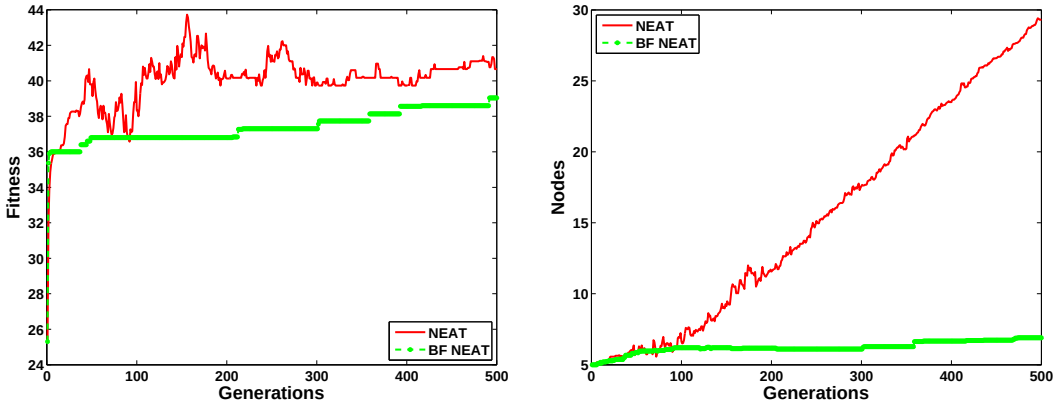
(a) Individuals distribution (b) Species distribution

Figura 5.8: Evolución de tamaño para BF-NEAT en el problema Parity: a) La distribución de tamaño de los individuos basados en el número de nodos; b) La distribución de tamaño de las especies basada en el número de nodos.



(a) Best Fitness (b) Nodes

Figura 5.9: Rendimiento en el problema XOR: a) Mejor aptitud y b) Número de nodos.



(a) Parity NEAT vs. BF NEAT Fintness (b) Parity NEAT vs. BF NEAT Nodes

Figura 5.10: Rendimiento en el problema Parity: a) Mejor aptitud y b) Número de nodos.

resultados. Para ello, consideramos cada una de las características de NEAT que podría ser la causa o inferir en los resultados obtenidos.

En primer lugar, NEAT utiliza una representación gráfica, que por sí misma no tiene propiedades intrínsecas para controlar el bloat.

En segundo lugar, puede que los operadores de búsqueda están diseñados para impedir el crecimiento de tamaño. Sin embargo, el algoritmo original de NEAT utiliza operadores de búsqueda que promueven un aumento de tamaño, que es altamente inusual en GP. No obstante, NEAT se puede configurar para ejecutar libre de bloat.

En tercer lugar, NEAT comienza la evolución con la menor topología posible que considera todos los elementos de entrada o terminales. De acuerdo con la CBT, esto puede fomentar el bloat, ya que las soluciones pequeñas normalmente mostrarán aptitudes por debajo del promedio. Por otra parte, comenzando con el menor tamaño posible de individuos, las poblaciones sólo pueden aumentar de tamaño para que NEAT explore el espacio de búsqueda. Por lo tanto, parece ser que esta característica no es responsable de controlar el bloat.

Esto nos deja con la última característica del algoritmo, la especiación basada en topología o tamaño. Básicamente, la especiación en NEAT se puede configurar para promover la supervivencia de soluciones de diferentes tamaños. En otras palabras, puede promover una distribución uniforme o plana de tamaño de programas. Los resultados experimentales confirman que el reproducir el comportamiento exhibido

por Flat-OE puede contrarrestar el efecto de bloat en variantes de GP como lo es NEAT. Sin embargo, también es crucial entender que este comportamiento no se puede obtener utilizando NEAT con la configuración preestablecida, ya que el conjunto original de parámetros y valores de NEAT no fueron diseñados para controlar de forma explícita el bloat.

5.3.2. Punto clave

En los experimentos previos se estudia el fenómeno de bloat en una variante de GP, que es el método de NEAT, siguiendo las ideas proporcionadas por el método Flat-OE. Donde los resultados anteriores sugieren que mediante la preservación de la diversidad de tamaños de individuos dentro de la población, el bloat puede ser controlado en gran medida o por completo [24]. Los resultados obtenidos brindan la posibilidad de controlar el bloat con un método distinto, utilizando los conocimientos proporcionados por NEAT.

Capítulo 6

neat-GP

En el capitulo anterior se comprobó que NEAT es un método capaz de ejecutarse sin bloat para redes neuronales, pero aun queda la incógnita si sucederá el mismo caso al aplicarlo en programación genética, en especifico de tipo árbol. Para validar que la combinación de los métodos, es necesario desarrollar el algoritmo de Stanley and Miikkulainen [27] en una librería que permita modificaciones (plug and play) y que sea reconocida por la comunidad de GP, la cual es GPlab en matlab [23], por su funcionalidad y adaptabilidad de módulos, así como su aceptación en la comunidad investigadora de GP.

Este trabajo tiene como objetivo desarrollar un método de búsqueda en GP libre del fenómeno de bloat, inspirado en las ideas de Flat-OE y los principios algorítmicos generales de NEAT. Recordemos que NEAT es un algoritmo complejo con una gran variedad de parámetros que afectan el desarrollo de la ejecución y resultado de búsqueda. Este trabajo solo pretende asimilar de NEAT [27] las propiedades que le permitan controlar el bloat y no reproducir el algoritmo como tal.

El método propuesto se llama neat-GP, el nombre no es un acrónimo, es solo utilizado como referencia para los métodos utilizados para su creación. Es importante mencionar que neat-GP no está diseñado para ser una reproducción exacta de NEAT, sólo toma los principios generales del método, que son integrados en un algoritmo estándar de GP estilo Koza.

Las características esenciales de la implementación de neat-GP, las veremos a continuación.

6.1. Características NEAT y neat-GP

6.1.1. Inicia con una población mínima

En NEAT involucra que toda la población generada aleatoriamente, tiene la misma cantidad de entradas y salidas (nodos), así como conexiones, pero con distintos pesos en la NNet.

Para neat-GP

No existen las conexiones o pesos, pero si el concepto de nodos que contienen los operadores y variables, es posible establecer una semejanza donde se comience con una población mínima de tamaño de nodos. Sin olvidar que en la práctica más común GP comienza con una población aleatoria de tamaño de nodos, que permite tener una población con variedad de tamaños que mantiene un ecosistema variado por algunas generaciones hasta que el fenómeno de bloat ocurre. Pero se recomienda comenzar con una población de árboles pequeños, ya que esto permitirá la búsqueda de la solución en estructuras pequeñas que aumentan de tamaño solo en caso de no encontrar una buena solución, lo que lleva al algoritmo a búsquedas de estructuras más grandes dirigido por la mejor solución encontrada.

6.1.2. Especiación de la población

NEAT utiliza la fórmula de distancia δ entre individuos de la población para definir la pertenencia a una especie Ecuación (6.1), esta distancia tiene un límite predeterminado δ_0 en base al criterio humano (el usuario) que establece que tan distinto puede ser un individuo de otro para ser considerado otra especie, pero la fórmula de distancia fue desarrollada solo para NNets que contiene pesos de conexión ($\overline{W}$), genes adjuntos (G) y de exceso (D) y el genoma más grande (N), por lo tanto no puede ser aplicada en estructuras de árbol los cuales no tienen los mismos elementos.

$$\delta = \frac{c_1 \cdot G + c_2 \cdot D}{N} + c_3 \cdot \overline{W}, \tag{6.1}$$

Para neat-GP

Para poder distinguir y agrupar especies de árboles es necesario definir las características con las que contamos, que son los números de nodos, niveles de profundidad,

operadores que contiene el árbol. Recordando los ajustes realizados en la configuración de Bloat Free NEAT (BF-NEAT), no es necesario contemplar los operadores y pesos que contiene las NNets, solo es necesario enfocarnos en el tamaño de la estructura del árbol y es posible descartar los operadores en los nodos del árbol. Y solo enfocarnos en aquellos factores que afectan el tamaño del árbol que es el número de nodos y la profundidad del árbol. Por lo tanto la fórmula a utilizar para realizar la especiación de la población se puede ver en la Ecuación (6.2).

$$\delta_T(T_i, T_j) = \alpha \frac{N_{i,j} - (2n_{S_{i,j}})}{N_{i,j} - 1} + (1 - \alpha) \frac{D_{i,j} - (2d_{S_{i,j}})}{D_{i,j} - 1} \ , \tag{6.2}$$

Donde $N_{i,j}$ es la suma de nodos entre los árboles T_i y T_j, $D_{i,j}$ es la suma de la profundidad(Depth) de los árboles T_i y T_j, $n_{S_{i,j}}$ son los nodos que comparte la estructura similar entre los árboles T_i y T_j, $n_{S_{i,j}}$ es el nivel que profundidad que tiene la estructura similar entre dos árboles T_i y T_j, y $\alpha \in [0, 1]$ de tal forma que $\delta_T \in [0, 1)$; en este trabajo $\alpha = 0{,}5$ dando la misma importancia al tamaño y niveles profundidad de los árboles.

Para poder comprender la Ecuación (6.2) debemos definir cuando los nodos y niveles de profundidad son semejantes, ya que estos son aquellos nodos y niveles entre dos árboles X y Y donde la estructura es semejante sin tomar en cuenta los operadores y aridad, ver Figuras [6.1,6.2,6.3].

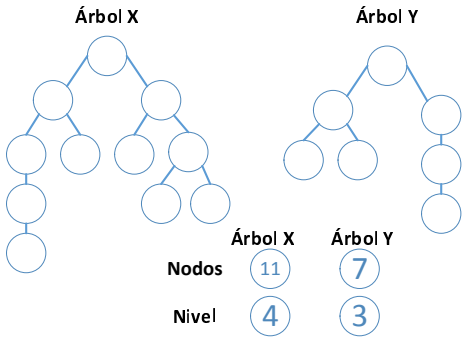


Figura 6.1: Aquí se muestran dos árboles aleatorios de una población con sus medidas individuales sin ser comparadas aun.

Siguiendo las Figuras [6.1,6.2,6.3] ejecutemos la Ecuación 6.3

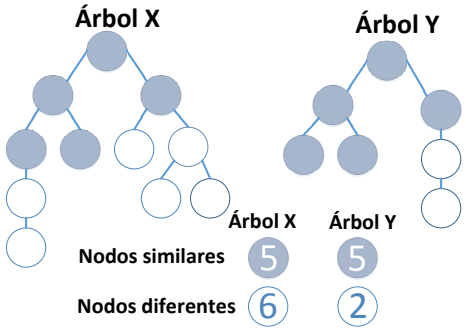


Figura 6.2: Los nodos similares son aquellos que se interpolan independientemente de la aridad. Como la imagen superior lo muestra, son los nodos oscuros. Y los nodos diferentes son todos aquellos fuera de la estructura interpolada.

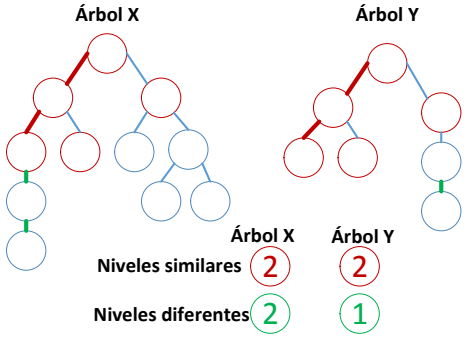


Figura 6.3: Los niveles se calculan al identificar la sección similar como lo muestra la estructura en rojo, con esto podemos calcular el nivel máximo donde son semejantes, el cual es restado al nivel total del árbol para calcular el nivel en diferencia entre árboles.

$$\delta = (((6+2)/(11+7)+(2+1)/(4+3)))/2 = (,444+,428)/2 = ,872/2 = ,436 \quad (6.3)$$

Si el valor obtenido es mayor que la distancia límite predeterminada por δ_0, el árbol no pertenece a la especie, de lo contrario el individuo es agrupados como una especie en común con el árbol en comparación. Los valores que puede tomar la distancia límite predeterminada por δ_0 se encuentran dentro del rango de $0 > \delta_0 < 1$ o [0,1], donde un valor de δ_0=.15 indica que los arboles deben tener un 15 % de diferencia con el árbol en comparación para ser considerados otra especie.

6.1.3. Penalización

La penalización en NEAT es por aptitud compartida, donde las NNets son evaluadas para calcular su aptitud, ya con la aptitud calculada es penaliza en base a la cantidad de NNets de la misma especie en la población. La forma de penalizar de NEAT es dividir la aptitud de la NNet por la cantidad de individuos en la especie, tomando en cuenta que se desea maximizar (el objetivo es obtener un valor especifico de forma incremental). Además de utilizar otros criterios para evitar el estancamiento de especies obsoletas (age significance) que elimina las especies en base a un criterio preestablecido o bien solo seleccionar las dos mejores especies y eliminar el resto después de una cantidad específica sin mejora alguna. Por ultimo NEAT cuenta con un parámetro que indica cuantas NNets de las especies no son penalizadas por ser las mejores de la especie, esto con el fin de mantener el elitismo y no perder ejemplares que pudieran generar la solución.

Para neat-GP

El objetivo es minimizar por lo tanto no se penaliza dividiendo sino multiplicando la aptitud del árbol, se multiplica la aptitud del árbol por la cantidad de árboles de la misma especie en la población. Y solo queda exento de penalización el mejor árbol de cada especie.

6.1.4. Selección

En NEAT la selección de reproducción es elitista solo los mejores puede reproducirse, con un porcentaje predefinido en los parámetros de configuración del algoritmo. Algo notable del método es que aquellos seleccionados para reproducción son los que sobreviven la siguiente generación. Y lo menos aptos son remplazados por la mejor descendencia generada. Por ejemplo en una población de individuos el porcentaje de selección para reproducción del 80 % de la población tiene la oportunidad de reproducirse y el 20 % restante será remplazado por los mejores hijos generados en la reproducción.

Para neat-GP

En la versión de GP se reproduce el método sin modificaciones

6.1.5. Reproducción

La siguiente sección se reproduce el operador de cruce NEAT en cuestión de cambios de operadores dentro de la estructura similar y agregando ramas completas en los bordes de la estructura similar, pero siguiendo el enfoque de algoritmos evolutivos.

Para neat-GP

La reproducción tiene un comportamiento elitista aleatorio con un porcentaje de 50 %, lo que permite que se reproduzcan los individuos con mejor aptitud, así como los arboles emergentes con aptitudes aun no optimizadas que requieren de cruce o mutación para poder mejorar.

Aquellos árboles modificados por los operadores de cruce y mutación son llamados descendencia, y cada árbol seleccionado para reproducción tiene un número finito de descendencia que puede producir, que se calcula por la Ecuación (6.4).

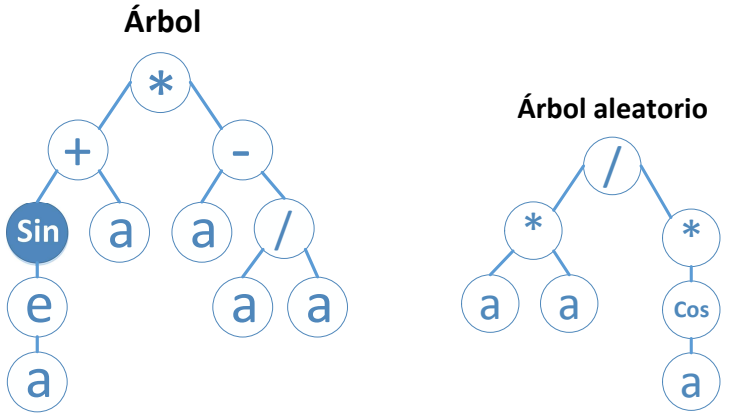
$$arbol(x)_{NoDescendencia} \approx \frac{\overline{Aptitud_{poblacion}}}{Aptitud_{arbol(x)SinPenalizar}} \tag{6.4}$$

La cantidad de descendencia de cada individuo es reducida cada vez que genera un hijo, en caso de mutación se reduce 1 y para cruce se reduce .5 para cada padre ya que tiene un 50 % de posibilidad de heredar sus características.

La descendencia se genera aplicando los operadores de mutación y cruce, desarrollado específicamente para NNets. Los detalles ya fueron vistos en la Sección NEAT. Pero es necesario desarrollar un método para árboles que realice funciones semejantes.

Mutación:

Realiza cambios aleatoriamente en la estructura del árbol seleccionado con un porcentaje pre definido previamente. Se genera un árbol aleatoriamente, el cual sera insertado en un nodo aleatorio en el árbol seleccionado para mutación, ver Figuras [6.4,6.5].



(a) Se selecciona un nodo aleatoriamente (nodo de color azul oscuro), y se genera un árbol aleatoriamente,

(b) El cual remplazara la sección del nodo seleccionado previamente para ser mutado.

Figura 6.4: Mutación en neat-GP

Cruce-NEAT:

El cruce neat-GP se realiza entre árboles de la misma especie y solo en casos donde no se encuentre un árbol de la misma especie se realizara con una especie distinta. Y solo genera un nuevo árbol como descendencia a diferencia del método de cruce de GP que genera dos árboles de descendencia.

Se seleccionan dos árboles, ver Figura 6.6.

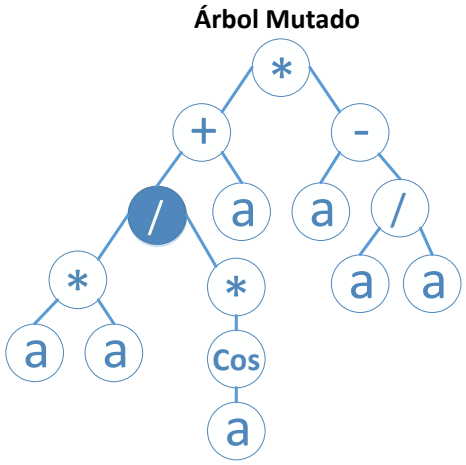


Figura 6.5: Generando un nuevo árbol con nuevas características, expandiendo el espacio de busqueda (nodo de color azul oscuro es el punto de mutación).

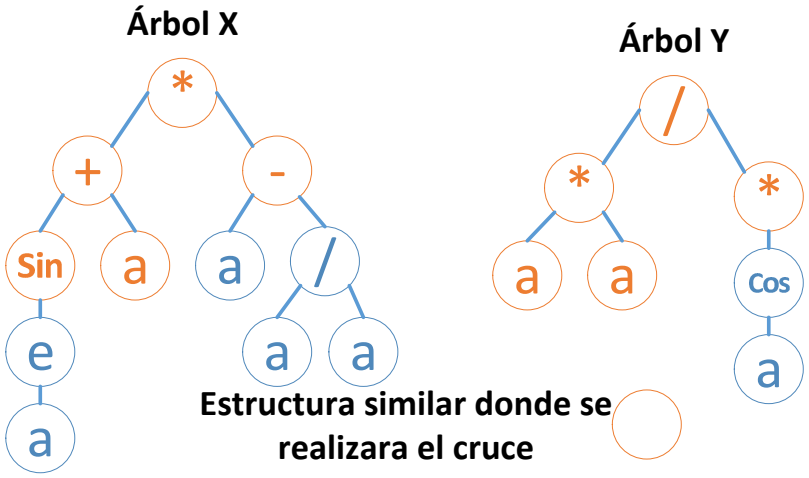


Figura 6.6: Estructura similar para realizar el cruce.

Se identifican los nodos donde la estructura es similar para poder realizar los cruces pertinentes, en los nodos con estructura similar se intercambiaran los operadores que compartan la misma aridad con una probabilidad de 50 % para intercambiar el operador o no, y se intercambiaran secciones con la misma probabilidad en los nodos con distinta aridad, ver Figuras [6.8,6.8,6.9,6.10].

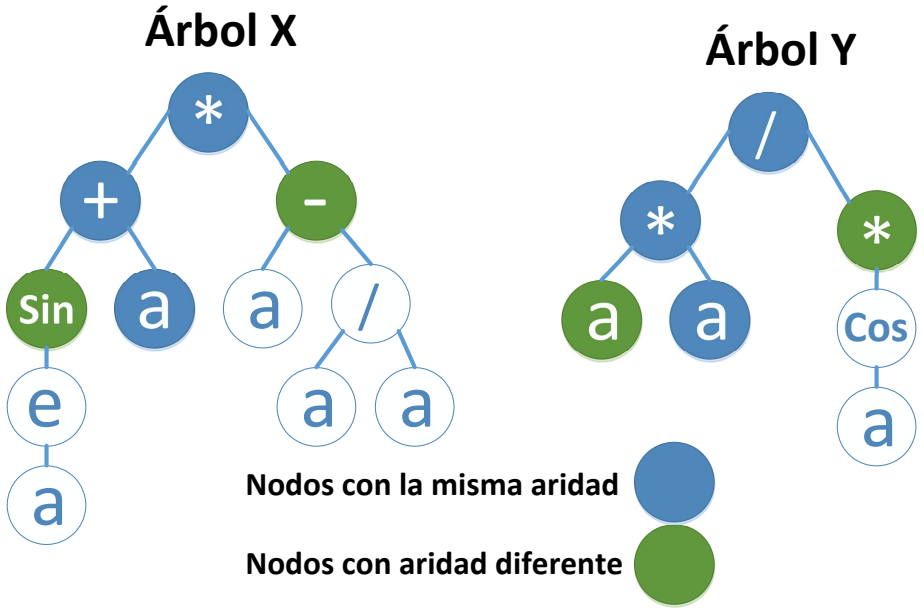


Figura 6.7: Cambio de operador en la estructura similar donde la aridad es idéntica en ambos árboles y aridad diferente en los bordes de la estructura (con probabilidad de 50 %)

Los nuevos individuos generados, son comparados con la población de padres para asignar la especie a la que pertenecen, y también son penalizados en base a la cantidad de individuos dentro de la especie, recordando que solo el mejor de toda la especie incluyendo hijos y padres será el único sin penalización.

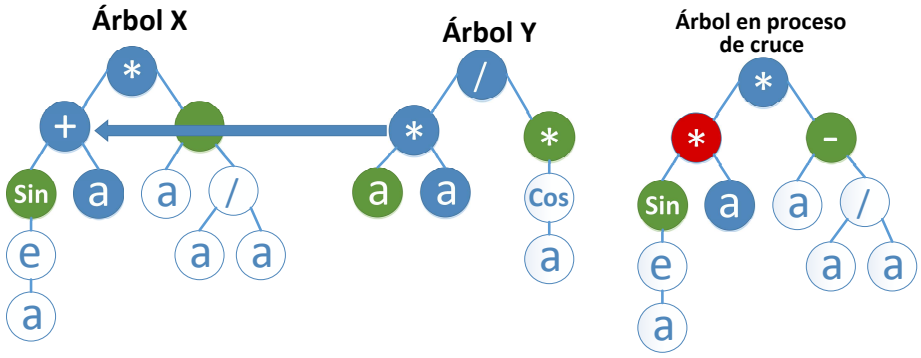


Figura 6.8: Cambios en operador en nodos con aridad idénticas dentro de la estructura similar con probabilidad de 50 % en cada nodo

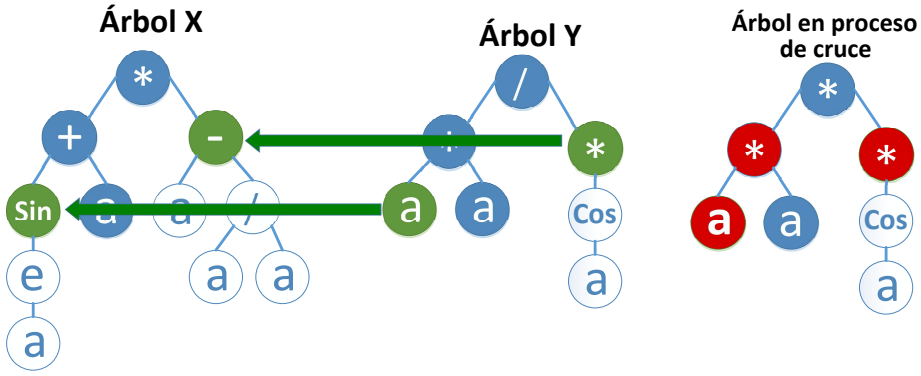


Figura 6.9: Cambio de ramificación en nodos con aridad diferente dentro de la estructura similar con probabilidad de 50 % en cada nodo

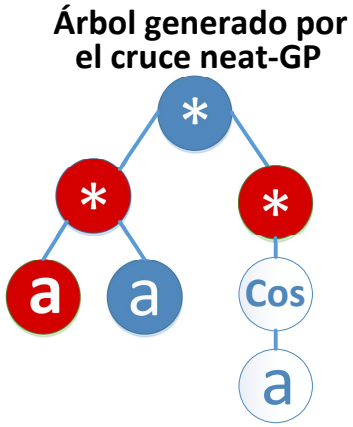


Figura 6.10: Nuevo árbol generado por el cruce neat-GP

6.2. Experimentos

Diferentes versiones del algoritmo neat-GP son puestos a prueban, para ilustrar el efecto que cada componente tiene en el rendimiento de la aptitud y el bloat. Estas versiones son: *neat*-GP como se describe en la sección anterior; *neat*-GP-SC que utiliza cruce de subárbol estándar en lugar del cruce-NEAT; *neat*-GP-Spe que omite la restricción de especiación; *neat*-GP-Sel que utiliza la selección de torneo en lugar del algoritmo de selección propuesto en neat-GP; y finalmente *neat*-GP-FS que no emplea la penalización por aptitud compartida. Además se utiliza Programación Genética como un método de referencia, de ahora en adelante referido como GP.

Dos tipos de problemas se utilizan para probar los algoritmos. En primer lugar, nueve problemas de regresión simbólicos, que son elegidos sobre la base de las propuestas formuladas en [12, 13, 15, 32, 34], los problemas seleccionados se resumen en la Tabla 6.1. Para GP, los parámetros se dan en la Tabla 6.2, para *neat*-GP y sus variantes se utilizan los especificados en la Tabla 6.3. En todos los problemas de regresión simbólica, la aptitud se calcula como la raíz del error cuadrático medio entre los productos previstos y esperados, para los problemas de clasificación es el porcentaje de exactitud al clasificar los datos. Treinta corridas independientes se realizan para cada problema, con diferentes conjuntos de entrenamiento y prueba, los resultados se presentan como estadísticas sobre todas las corridas.

En segundo lugar, se utilizan cinco problemas de clasificación del mundo real, tomados del repositorio en linea de la University of California Irvine (UCI) conocida como repositorio de aprendizaje maquina [2], los problemas seleccionados se resumen en la Tabla 6.4.

6.3. Resultados

Las Figuras 6.11 a 6.19 muestran los resultados de los nueve problemas de regresión simbólica, que muestra diagramas de caja para las treinta corridas midiendo el desempeño en base a: 1) la aptitud de prueba de la mejor solución; 2) el tamaño promedio de la población basada en el número de nodos; y 3) la profundidad promedio. Por otra parte, la Tabla 6.5 resume la comparación estadística hecha entre GP y todas las variantes de *neat*-GP mediante la prueba de rangos de Wilcoxon con

Tabla 6.1: Problemas de referencia de regresión simbólica.

No	Problem	Function	Fitness/Test cases	Function Set
1	$Koza-1$	$x^4+x^3+x^2+x$	20 random $\subseteq[-1,1]$	$Table-6{,}2$
2	$Nguyen-3$	$x^5+x^4+x^3+x^2+x$	20 random $\subseteq[-1,1]$	$Table-6{,}2$
3	$Nguyen-5$	$sin(x^2)*cos(x)-1$	20 random $\subseteq[-1,1]$	$Table-6{,}2$
4	$Nguyen-7$	$\ln(x+1)+\ln(x^2+1)$	20 random $\subseteq[0,2]$	$Table-6{,}2$
5	$Nguyen-10$	$2sin(x)*cos(y)$	100 random $\subseteq[-1,1]x[-1,1]$	$Table-6{,}2$
6	$Keijzer-6$	$\sum_i^x \frac{1}{i}$	$E[1,50,1], E[1,120,1]$	$Keijzer$
7	$Korns-12$	$2-2{,}1cos(9{,}8x)sin(1{,}3w)$	$U[-50,50,10000]$	$Korns$
8	$Vladislavleva-1$	$\frac{e^{-(x-1)^2}}{1{,}2+(y-2{,}5)^2}$	$U[0{,}3,4,100]$	$Vladislavleva-B$
9	$Pagie-1$	$\frac{1}{1+x^{-4}}+\frac{1}{1+y^{-4}}$	$E[-5,5,0{,}4]$	$Koza$

Tabla 6.2: Parámetros utilizados en problemas de referencia con GP.

Parameter	Description
Population size	500 for regression; 200 for classification
Generations	100 generations for regression; 200 for classification
Initialization	*Ramped Half-and-Half*, with 6 levels of maximum depth
Operator probabilities	Crossover $p_c = 0{,}9$, Mutation $p_\mu = 0{,}05$
Function set (regression)	$\{+,-,\times,\div,sin,cos,exp,log\}$, $Keijzer$, $Korns$, $Vladislavleva-B$ and $Koza$.
Function set (classification)	$\{+,-,\times,\div,sin,cos,\exp,\sqrt{},x^y,\lvert x\rvert,if\}$
Terminal set (regression)	$x, 1$ for single variable problems and x, y for bivariable problem
Terminal set (classification)	problem features
Initial dynamic depth	6 levels
Hard maximum depth	20 levels
Selection	Tournament selection of size 3
Elitism	Best individual always survives

Tabla 6.3: Parámetros utilizados en problemas de referencia con *neat*-GP.

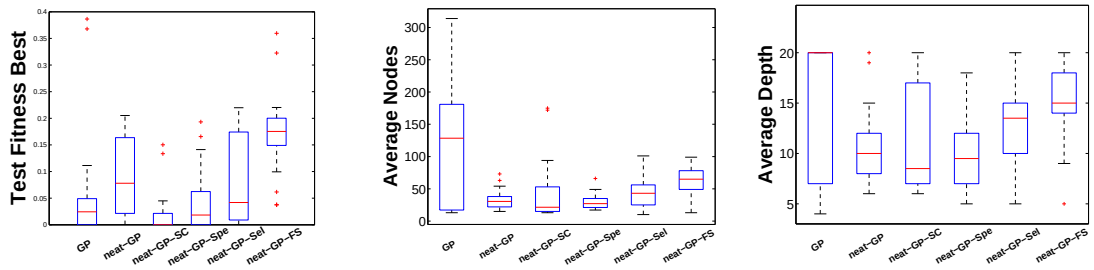
Parameter	Description
Population size	500 for regression; 200 for classification
Generations	100 generations for regression; 200 for classification
Initialization	*Full initialization*, with 3 levels of maximum depth
Operator probabilities	NEAT-Crossover $p_c = 0{,}7$, Mutation $p_\mu = 0{,}3$
Function set (regression)	$\{+,-,\times,\div,sin,cos,exp,log\}$, $Keijzer$, $Korns$, $Vladislavleva-B$ and $Koza$.
Function set (classification)	$\{+,-,\times,\div,sin,cos,\exp,\sqrt{},x^y,\lvert x\rvert,if\}$
Terminal set (regression)	$x, 1$ for single variable problems and x, y for bivariable problem
Terminal set (classification)	problem features
Initial dynamic depth	3 levels
Hard maximum depth	20 levels
Selection	Eliminate the worst individuals of each species by the factor of $p_{worst}\,\%$
Elitism	Don't penalize the best individual of each species
Survival threshold	0.5
Specie threshold value	h=0.15 with α=0.5

$\alpha = 0{,}01$ para validar que los resultados obtenidos son diferentes a los resultados de GP.

Tabla 6.4: Problemas de clasificación del mundo real para GP y *neat*-GP

Problem	**Classes**	**Features**	**Samples**
UCI2 (Breast cancer Wisconsin)	2	8	441
UCI16 (Ionosphere)	2	32	350
UCI20 (Parkinsons)	2	22	195
UCI21 (Pima Indians Diabetes)	2	8	768
UCI22 (Sonarall)	2	60	208

Los resultado muestran tendencias, que son evidente en la mayoría de las gráficas de caja. En cuanto a la aptitud, es razonable afirmar que el objetivo de un método de control de bloat es reducir el tamaño promedio de los árboles en la población sin incurrir en una disminución del rendimiento. En otras palabras, es un resultado deseable pero no necesario, sería lograr una mejora del rendimiento con respecto a GP y al mismo tiempo eliminar el bloat. Todas las variantes de *neat*-GP tienden a producir árboles más pequeños que GP, como se muestra en las Figuras 6.20 a 6.28, con la distribución creciente de GP y la estabilidad de tamaño de *neat*-GP. Sin embargo, sólo un método, *neat*-GP-SC, logra un rendimiento igual o mejor que GP basado en la aptitud, con la prueba estadística se tiene un mejor desempeño en tres de los problemas, ver Tabla 6.5.



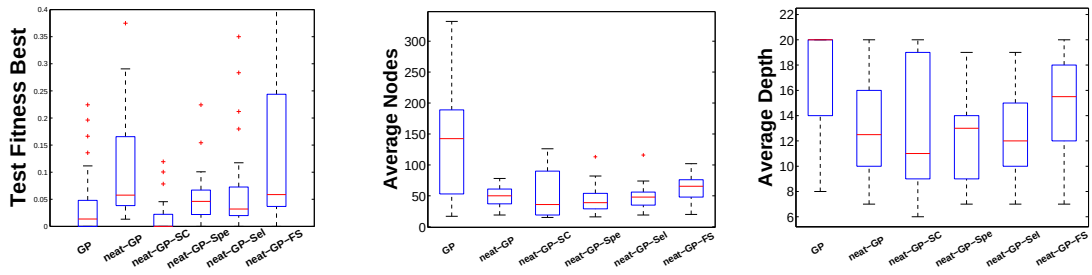
(a) (b) (c)

Figura 6.11: Diagramas de caja para los resultados del problema de regresión Koza-1: a) prueba de aptitud; b) nodos; c) profundidad.

Para ilustrar el efecto del algoritmo *neat*-GP, las Figuras 6.20 a 6.28 muestran la distribución de tamaño de programa a través de todas las generaciones para los

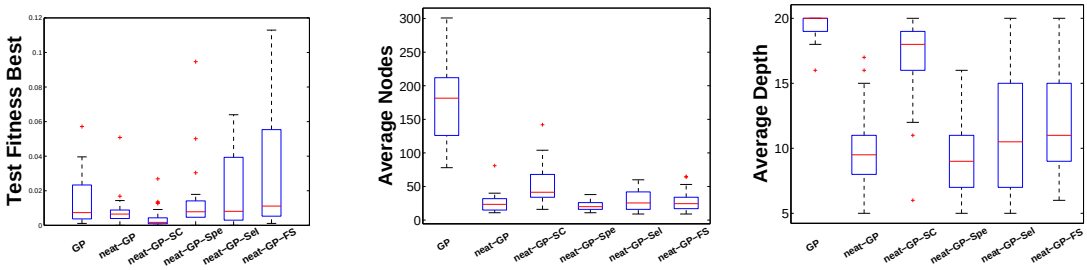
Tabla 6.5: La comparación estadística de las variantes *neat*-GP con GP en regresión simbólica, mediante la prueba de Wilconxon con $\alpha = 0{,}01$. Los resultados se presentan como un conjunto triple (x, y, z), que corresponden respectivamente a los resultados de la prueba basada en la aptitud de prueba, tamaño y profundidad. Tres resultados son posibles, que puede ser: W (peor) si la aptitud es menor y el tamaño de nodos y profundidad es mayor, E (igual) si las distribuciones son iguales o B (mejor) si la aptitud es mayor y los nodos y profundidad son menores.

Problem/Method	***neat*-GP**	***neat*-GP-SC**	***neat*-GP-Spe**	***neat*-GP-Sel**	***neat*-GP-FS**
Koza-1	(W,B,B)	**(E,B,B)**	(W,B,B)	(W,B,B)	(W,B,B)
Nguyen-3	(W,B,B)	**(E,B,B)**	(W,B,B)	(W,B,B)	(W,B,B)
Nguyen-5	(E,B,B)	**(B,B,B)**	(E,B,B)	(E,B,B)	(W,B,B)
Nguyen-7	(W,B,B)	**(E,B,B)**	(W,B,B)	(W,B,B)	(W,B,B)
Nguyen-10	(E,B,B)	**(B,B,B)**	(E,B,B)	(E,B,B)	(W,B,B)
Keijzer-6	(W,B,B)	**(E,B,B)**	(W,B,B)	(W,B,B)	(W,B,B)
Korns-12	(B,B,B)	**(B,B,B)**	(B,B,B)	(B,B,B)	(B,B,B)
Vladislavleva-1	(E,B,B)	**(E,B,B)**	(E,B,B)	(E,B,B)	(E,B,B)
Pagie-1	(W,B,B)	**(E,B,B)**	(W,B,B)	(W,B,B)	(W,B,B)



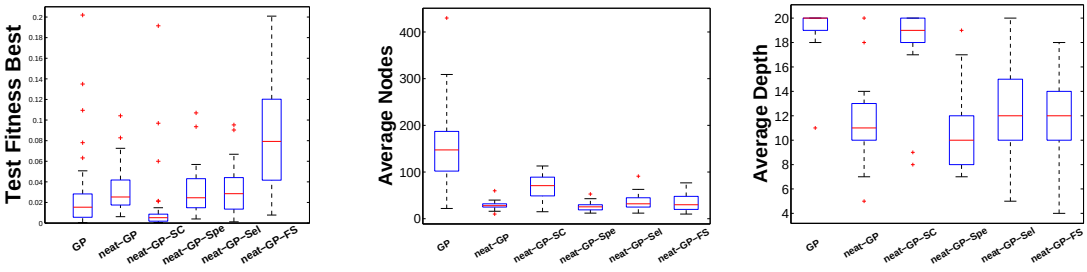
(a) (b) (c)

Figura 6.12: Diagramas de caja para los resultados del problema de regresión Nguyen-3: a) prueba de aptitud; b) nodos; c) profundidad.



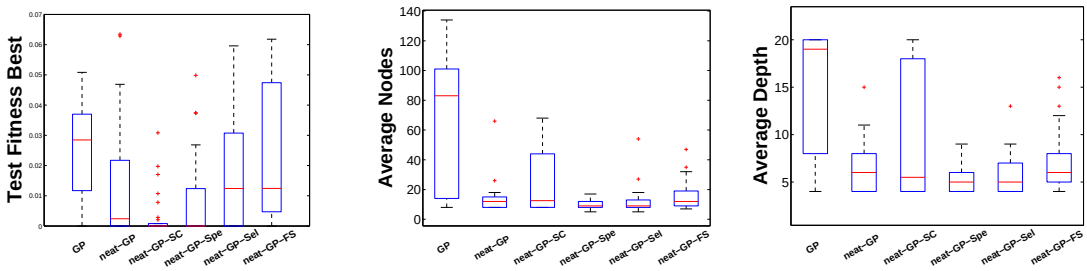
(a) (b) (c)

Figura 6.13: Diagramas de caja para los resultados del problema de regresión Nguyen-5: a) prueba de aptitud; b) nodos; c) profundidad.



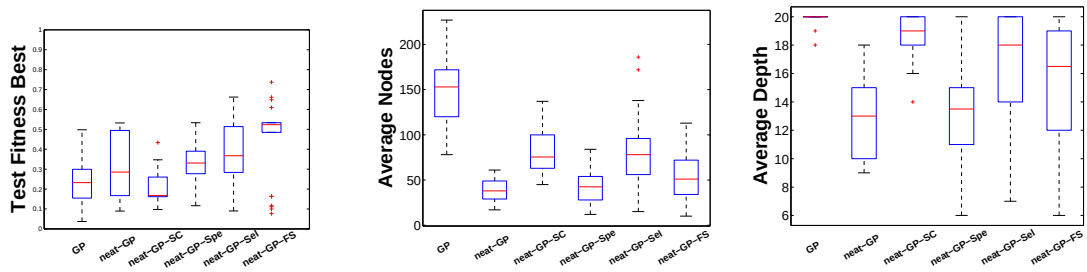
(a) (b) (c)

Figura 6.14: Diagramas de caja para los resultados del problema de regresión Nguyen-7: a) prueba de aptitud; b) nodos; c) profundidad..



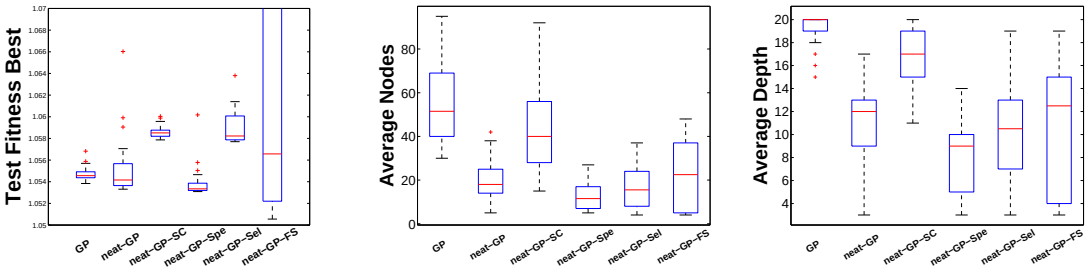
(a) (b) (c)

Figura 6.15: Diagramas de caja para los resultados del problema de regresión Nguyen-10: a) prueba de aptitud; b) nodos; c) profundidad.



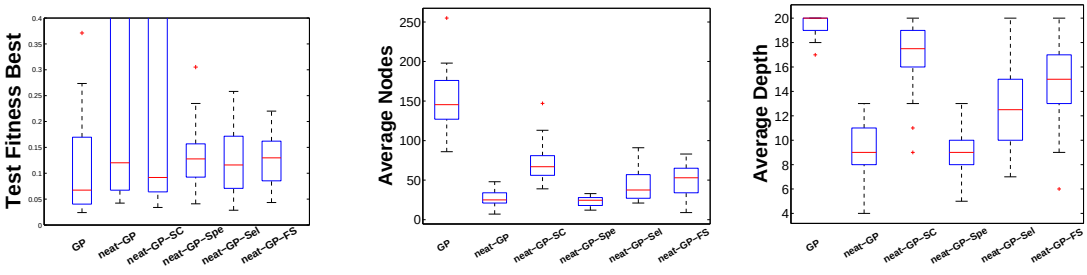
(a) (b) (c)

Figura 6.16: Diagramas de caja para los resultados del problema de regresión Nguyen-6: a) prueba de aptitud; b) nodos; c) profundidad.



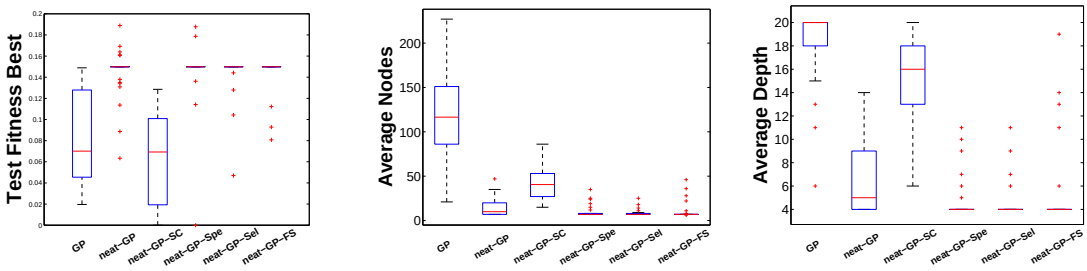
(a) (b) (c)

Figura 6.17: Diagramas de caja para los resultados del problema de regresión Korns-12: a) prueba de aptitud; b) nodos; c) profundidad.



(a) (b) (c)

Figura 6.18: Diagramas de caja para los resultados del problema de regresión Vladislavleva-1: a) prueba de aptitud; b) nodos; c) profundidad.



(a) (b) (c)

Figura 6.19: Diagramas de caja para los resultados del problema de regresión Pagie-1: a) prueba de aptitud; b) nodos; c) profundidad.

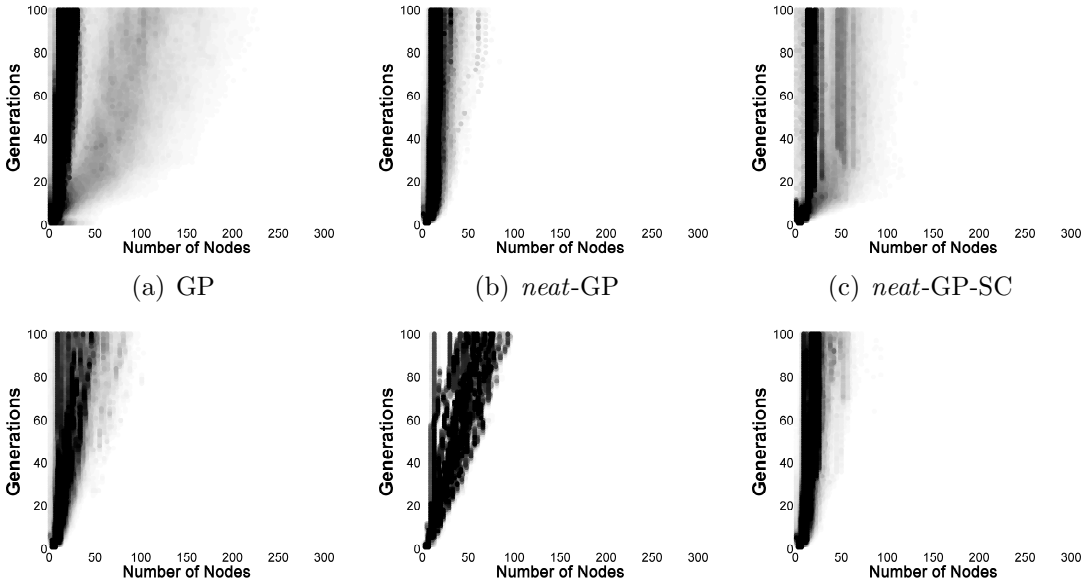
Tabla 6.6: La mediana de tamaño de la mejor solución para los problemas de regresión simbólica.

Problem/Method	**GP**	***neat*-GP**	***neat*-GP-SC**	***neat*-GP-Spe**	***neat*-GP-Sel**	***neat*-GP-FS**
Koza-1	128.5	30.5	21.5	27.0	43.0	65.0
Nguyen-3	142.5	50.0	36.0	39.0	48.0	65.5
Nguyen-5	181.5	23.5	41.5	20.0	25.5	24.5
Nguyen-7	147.5	28.0	71.0	25.5	32.0	30.0
Nguyen-10	83.0	12.0	12.5	9.0	9.0	12.0
Keijzer-6	153.0	38.0	75.5	42.5	78.0	51.0
Korns-12	51.50	18.0	40.0	11.5	15.5	22.5
Vladislavleva-1	145.5	25.0	67.0	24.5	37.5	53.0
Pagie-1	116.5	10.0	40.5	7.0	7.0	7.0

problemas de regresión simbólica. Las gráficas son promedios de 30 ejecuciones realizadas por cada método, donde las regiones más oscuras indica que hay muchos árboles de un tamaño determinado en un caso dado durante la búsqueda. En primer lugar, hay que considerar las Figuras 6.20a-6.28a, que corresponden a GP, en todos los problemas el comportamiento es similar, el tamaños de los programas aumenta rápidamente al comienzo de la búsqueda, y es empujado progresivamente hacia los árboles más grandes cuando la búsqueda progresa. Además, en la mayoría de los casos, la búsqueda tiende a generar una distribución multimodal de tamaños de programa al final de la corrida.

En algunas áreas durante la ejecución(generaciones avanzadas) las poblaciones no contienen programas de algunos tamaños Figuras 6.20a-6.28a, particularmente tamaños intermedios entre los programas más grandes y más pequeños de la población, el obtener tal distribución durante la búsqueda sesga la selección a árboles con un gran numero de nodos, como es algo esperado en base a la teoría de CBT. Por otro lado, la mayoría de las variantes de *neat*-GP muestran una tendencia diferente, que parecen producir una distribución casi uniforme en un rango determinado de tamaños de programas, que varía con cada problema, con un tendencia casi constante durante varias o el total de las generaciones. En algunos casos, los pequeños tamaños de los programas se mantienen durante toda la búsqueda, pero en otros casos es evidente que la búsqueda elimina árboles extremadamente pequeños, sin sesgar la búsqueda hacia los grandes tamaños de programas, como sucede para GP, ver Figuras 6.20b:f-6.28b:f. Si bien estas tendencias son verdaderas para todas las variantes de *neat*-GP, los métodos más consistentes parecen ser *neat*-GP y *neat*-GP-SC, ya que las otras variantes, a veces producen distribuciones uniformes o menos compacta, los beneficios en la reducción de bloat son mas evidentes en la Tabla 6.6 donde podemos apreciar la mediana de tamaño de la mejor solución, donde *neat*-GP y sus variantes presentan un tamaño menor que GP.

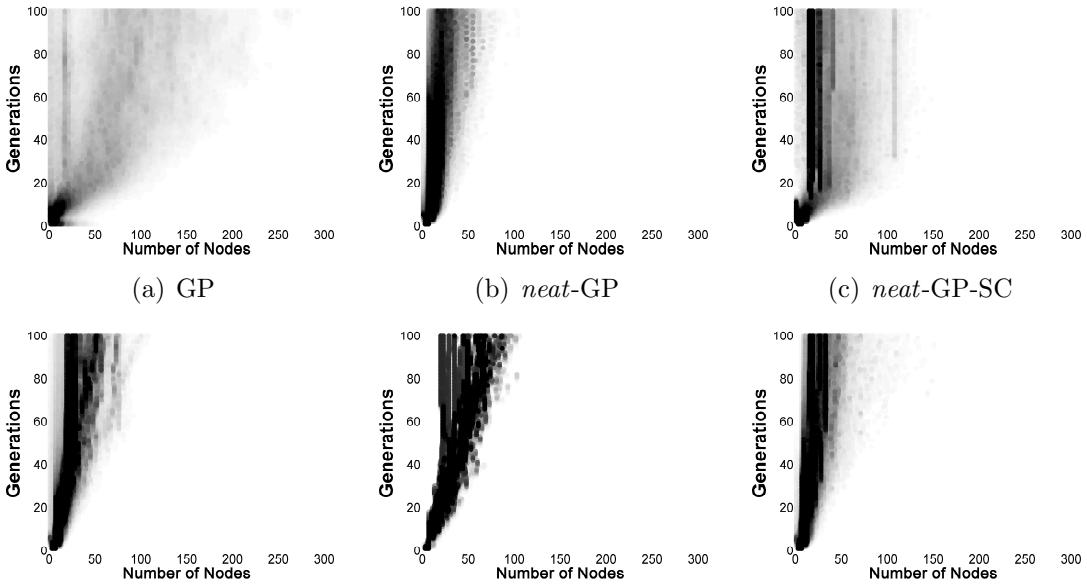
Dado al mejor rendimiento general basado en la aptitud de prueba de *neat*-GP y *neat*-GP-SC, sólo estos métodos se comparan con GP sobre los problemas de clasificación del mundo real. Las Figuras 6.29 a 6.33 muestran los resultados para los problemas de clasificación, que consiste en diagramas de caja para cada método utilizando, pero en este caso la aptitud y rendimiento de la prueba está dado



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

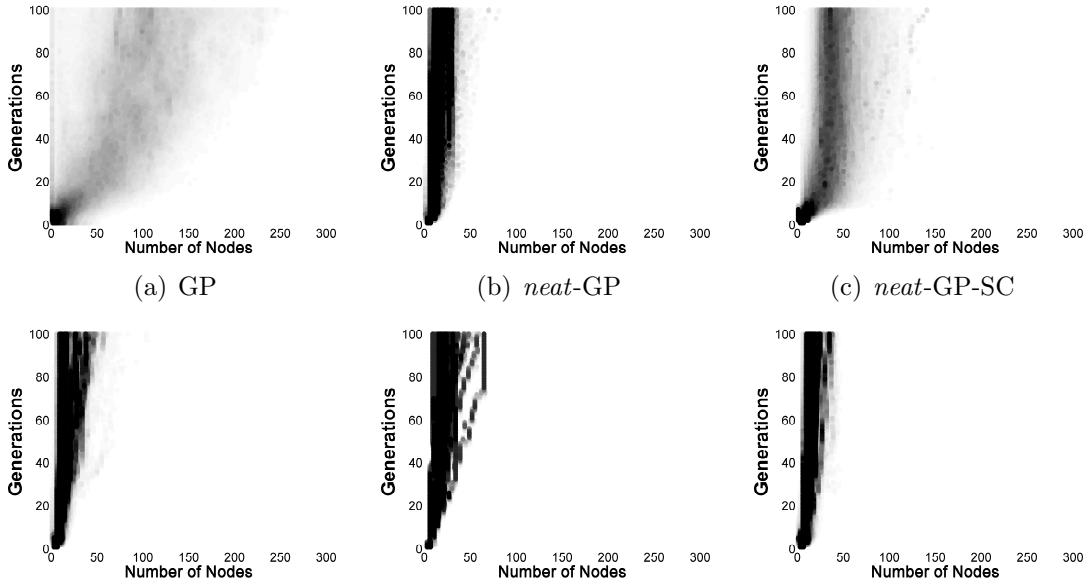
Figura 6.20: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Koza-1.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

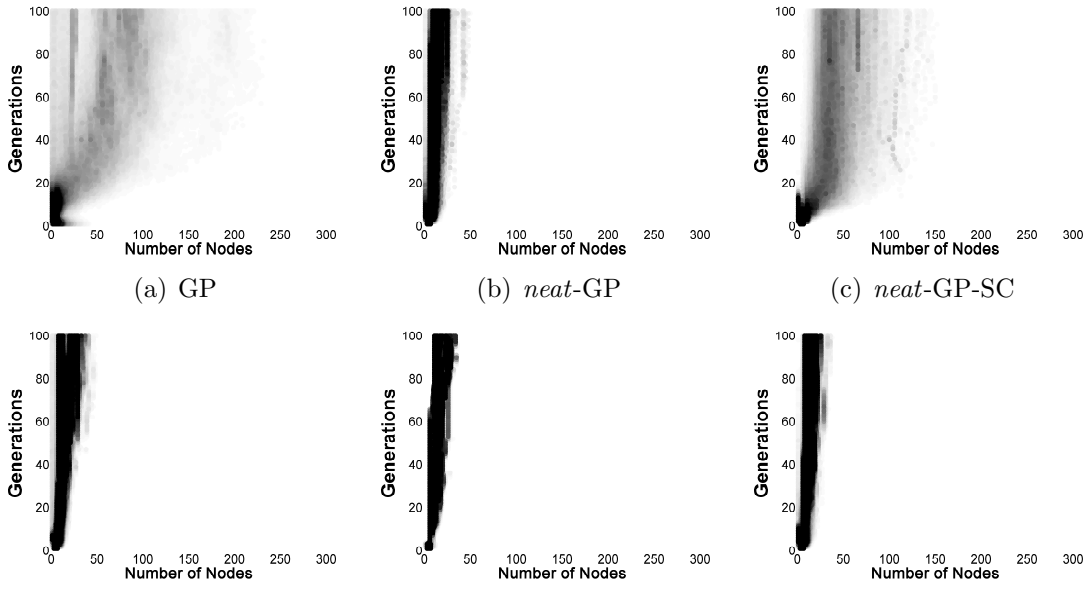
Figura 6.21: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Nguyen-3.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

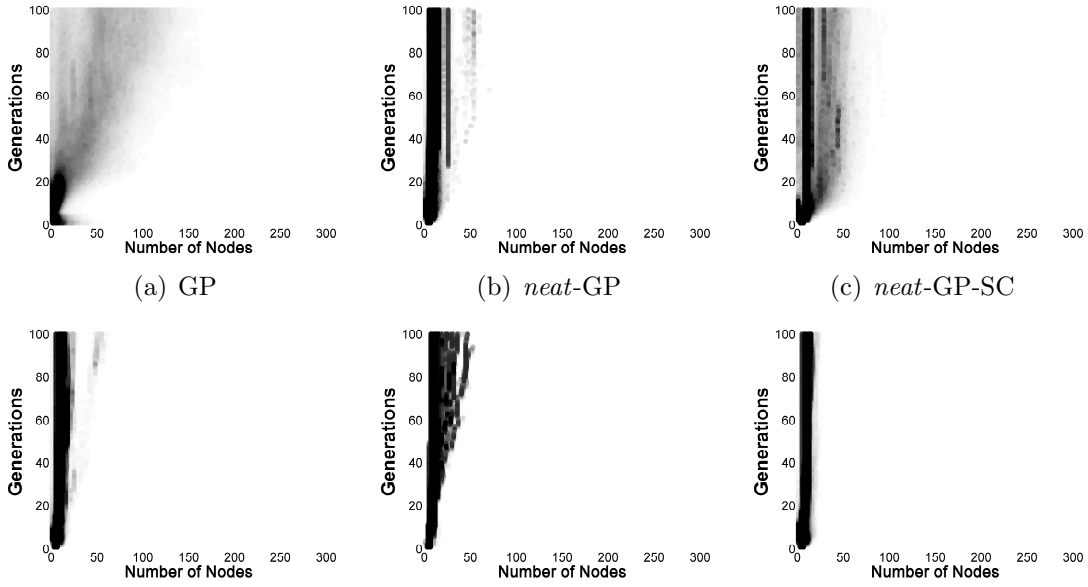
Figura 6.22: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Nguyen-5.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

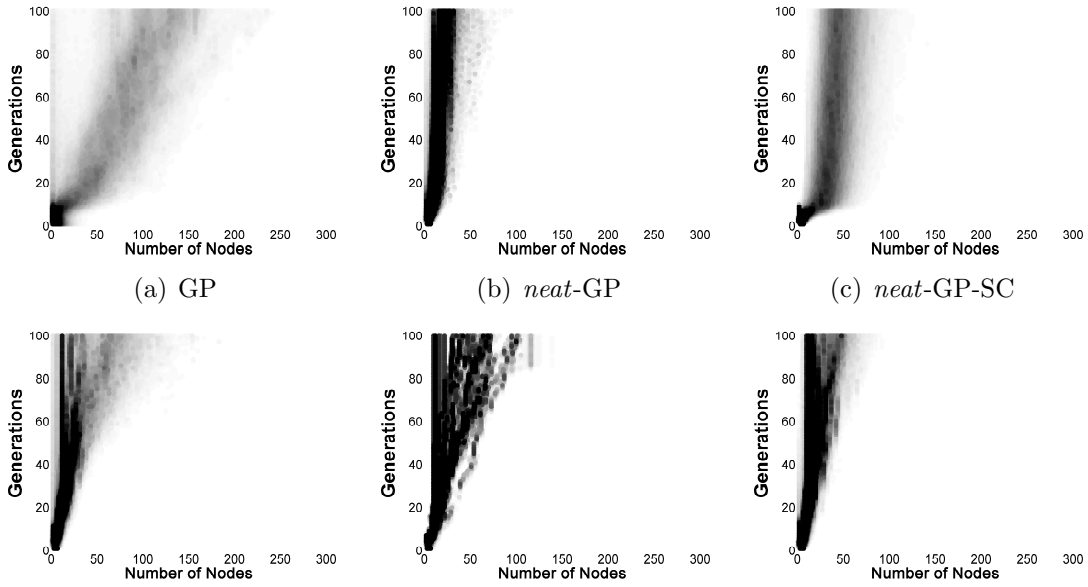
Figura 6.23: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Nguyen-7.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

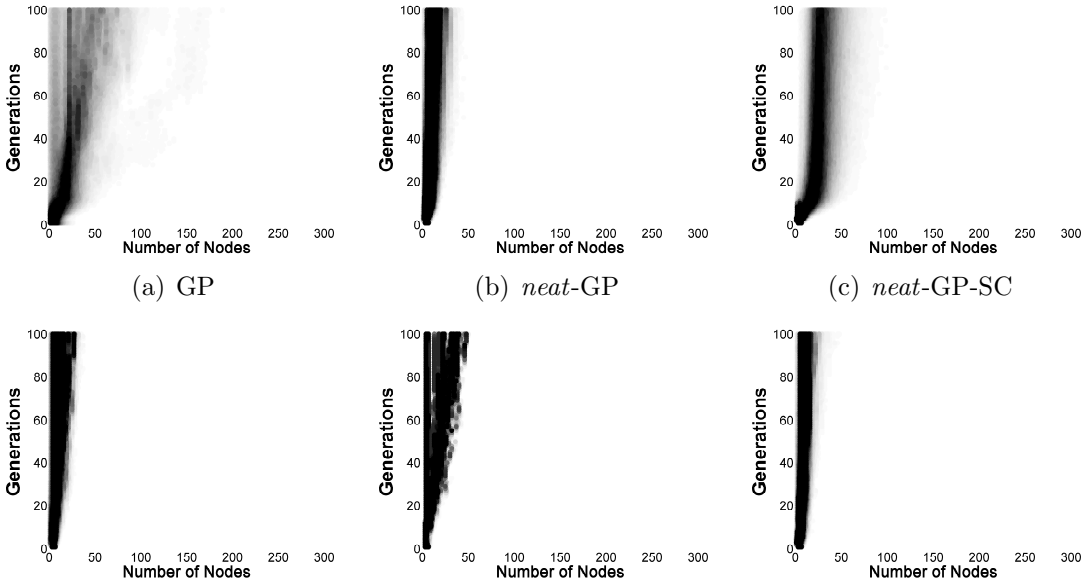
Figura 6.24: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Nguyen-10.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

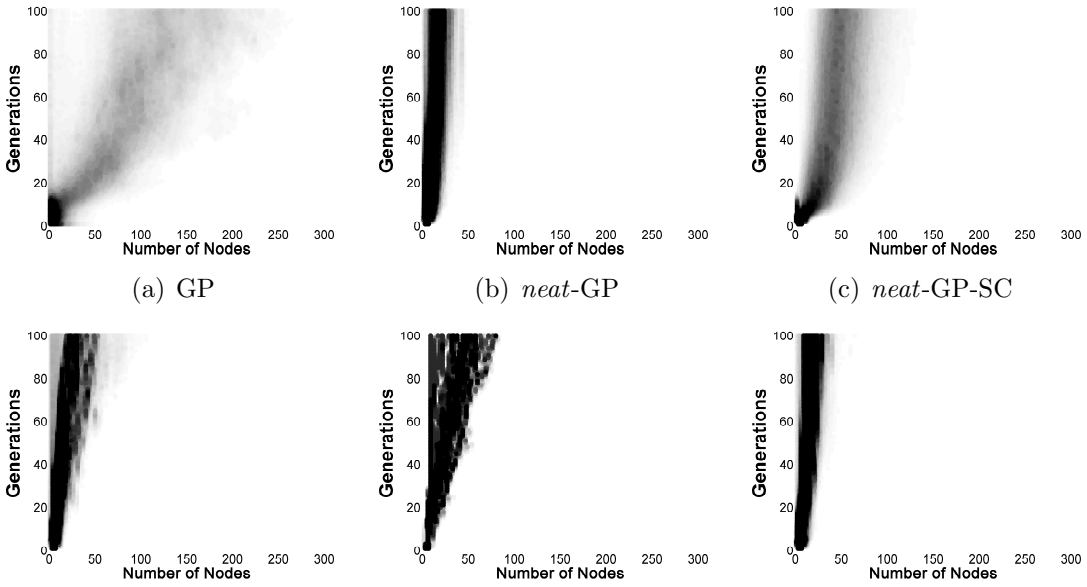
Figura 6.25: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Keijzer-6.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

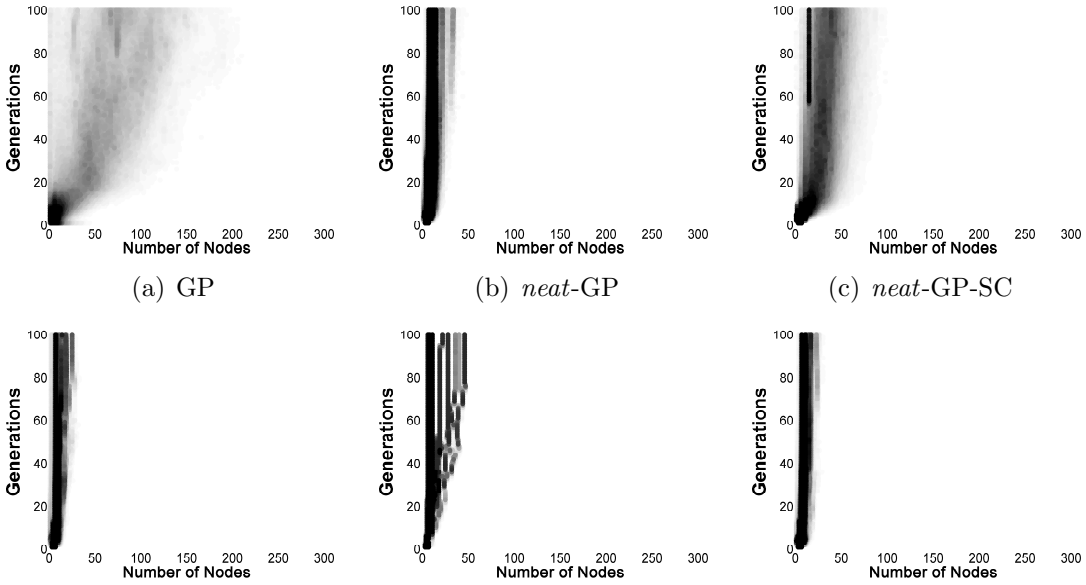
Figura 6.26: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Korns-12.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

Figura 6.27: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Vladislavleva-1.



(a) GP (b) *neat*-GP (c) *neat*-GP-SC

(d) *neat*-GP-Sel (e) *neat*-GP-FS (f) *neat*-GP-Spe

Figura 6.28: Distribución promedio de tamaños de programas sobre las 30 corridas para el problema de referencia Pagie-1.

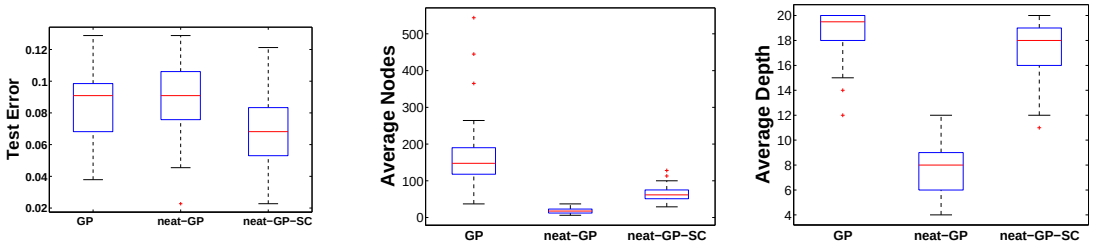
por el porcentaje de muestras mal clasificadas en otras palabras la exactitud de etiquetado. Las comparaciones estadísticas se presentan en la Tabla 6.7. Para estos problemas *neat*-GP-SC logra básicamente el mismo rendimiento que GP, basada tanto en la aptitud de prueba y sorprendentemente también en relación con el tamaño del programa, es decir, en estas pruebas *neat*-GP-SC no controla el bloat. De hecho, *neat*-GP-SC produce tamaños de programas más grandes en varios de los problemas de clasificación, es inesperado dado el buen comportamiento que exhibió en todos los problemas de referencia de regresión simbólica.

Por otro lado, el método *neat*-GP produce un buen rendimiento, logrando un rendimiento equivalente a *neat*-GP-SC, pero superando significativamente a ambos métodos en términos de tamaño del programa y la profundidad del árbol, controlando el bloat en todos los problemas.

De hecho, el control de bloat de *neat*-GP se puede apreciar si sólo tomamos en cuenta el tamaño de la mediana del mejor individuo que se encuentra en cada ejecución, ver Tabla 6.8. La disminución en el tamaño del programa es bastante grande en algunos casos, con la disminución del tamaño medio entre 46 % a 88 %, cuando

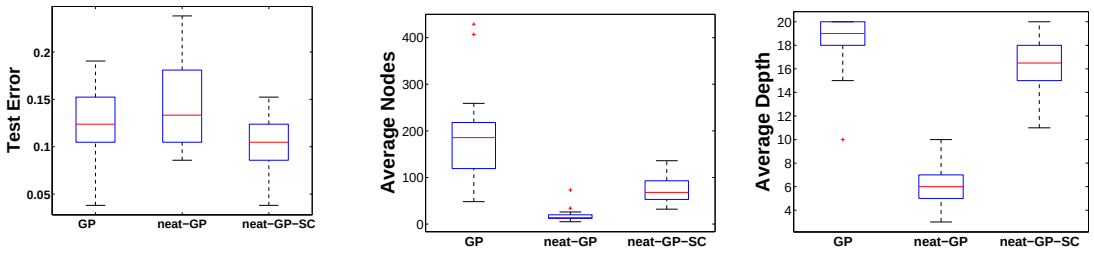
Tabla 6.7: La comparación estadística de todas las variantes de *neat*-GP contra GP sobre los problemas de clasificación, mediante la prueba de rangos de Wilconxon con $\alpha = 0{,}01$. Para mayor claridad, los resultados se presentan como un conjunto triple (x, y, z), que corresponden respectivamente a los resultados de la prueba basada en la aptitud, tamaño y profundidad. Tres resultados son posibles, que puede ser: W (peor) cuando la aptitud es menor y la cantidad de nodos y profundidad es mayor, E (igual) se tiene la misma cantidad o B (mejor) cuando la aptitud es mayor y los nodos como la profundidad es menor.

Problem/Method	***neat*-GP**	***neat*-GP-SC**
UCI-2	**(W,B,B)**	(W,B,B)
UCI-16	**(B,B,B)**	(B,E,W)
UCI-20	**(E,B,B)**	(E,W,W)
UCI-21	**(E,B,B)**	(E,W,W)
UCI-22	**(E,B,B)**	(E,E,W)



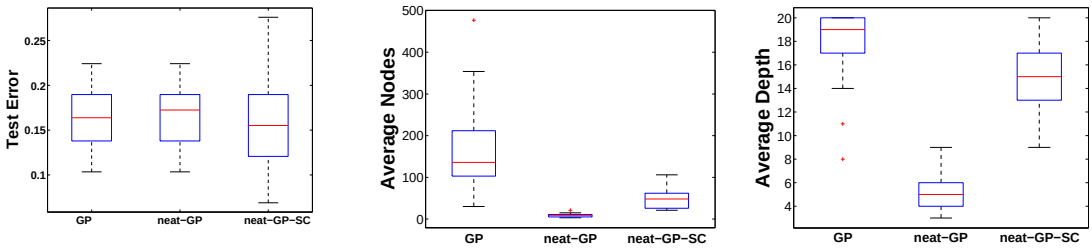
(a) (b) (c)

Figura 6.29: Diagramas de caja para los resultados del problema de clasificación Breast cancer: a) prueba de aptitud; b) nodos; c) profundidad.



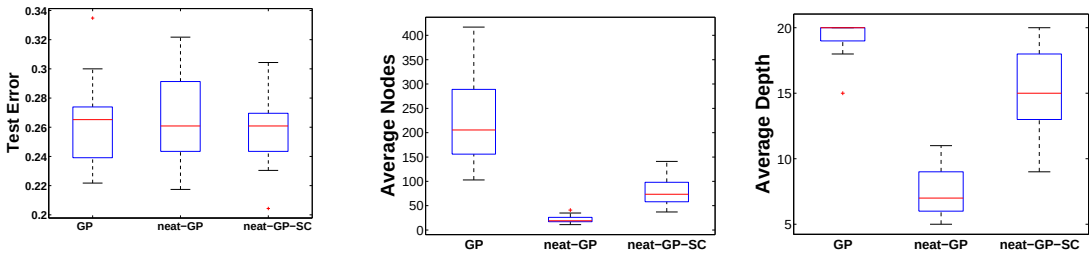
(a) (b) (c)

Figura 6.30: Diagramas de caja para los resultados del problema de clasificación Ionosphere: a) prueba de aptitud; b) nodos; c) profundidad.



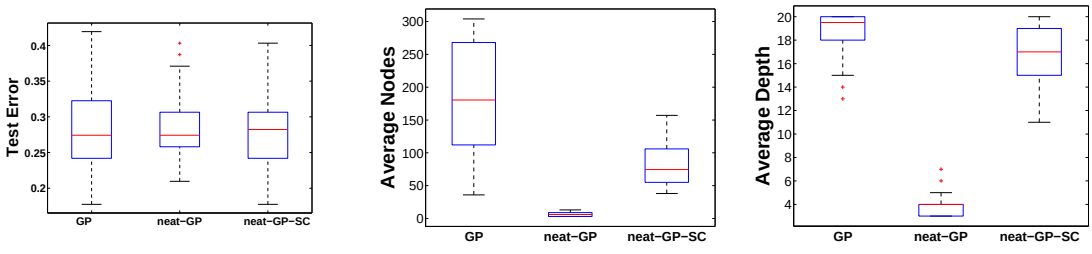
(a) (b) (c)

Figura 6.31: Diagramas de caja para los resultados del problema de clasificación Parkinson: a) prueba de aptitud; b) nodos; c) profundidad.



(a) (b) (c)

Figura 6.32: Diagramas de caja para los resultados del problema de clasificación pima indians diabetes: a) prueba de aptitud; b) nodos; c) profundidad.



(a) (b) (c)

Figura 6.33: Diagramas de caja para los resultados del problema de clasificación Sonarall: a) prueba de aptitud; b) nodos; c) profundidad.

se compara contra GP. Por último, con el fin de evaluar los costos computacionales de cada método, la relación de aceleración T_{neat}/T_{GP} para cada método, donde T_{neat} representa la mediana de tiempo de ejecución de cada experimento *neat*-GP, y T_{GP} para GP. En esta prueba, *neat*-GP da un aumento en velocidad de ejecucin por 2x y *neat*-GP-SC da una velocidad de hasta 1.5x, ambos son mejoras sustanciales con respecto a GP.

Tabla 6.8: La mediana de tamaño de la mejor solución para los problemas de clasificación del mundo real.

Problem/Method	**GP**	***neat*-GP**	***neat*-GP-SC**
UCI-2	41.5	17.5	61.5
UCI-16	59.0	14	68
UCI-20	42.5	9.0	48.0
UCI-21	42.5	19.0	73.5
UCI-22	50	6	74.5

6.4. Discusión

Los resultados obtenidos en la sección anterior muestran que *neat*-GP es capaz de reducir el crecimiento del código sin disminuir el rendimiento de la solución, y en algunos casos mejorar la calidad de las soluciones evolucionadas en comparación a GP. Por otra parte, *neat*-GP tienden a generar una distribución de tamaño de programas que es consistente con Flat-OE, que indica si una distribución uniforme de tamaño de programas se mantiene entonces el bloat puede ser reducido o eliminado. De hecho, se observa que algunos de las variantes de *neat*-GP, particularmente *neat*-GP y *neat*-GP-SC pueden producir distribuciones con tendencias planas que son casi constantes durante todo el proceso de búsqueda. Además, el rango de tamaño de los programas varía con cada problema, lo que sugiere que *neat*-GP es capaz de enfocar automáticamente la búsqueda en un rango específico de tamaño en función de la aptitud.

En los casos de prueba de *neat*-GP-Sel, *neat*-GP-FS y *neat*-GP-Spe respectivamente, cuando se retira las características generales del método de NEAT por un

	GP	neat-GP	neat-GP-SC
UCI2 (Breast cancer Wisconsin)	**0.0577**	0.0909	0.0682
	41.5000	**17.5000**	61.5000
	13.5000	**8.0000**	18.0000
UCI16 (Ionosphere)	0.1143	0.1333	**0.1048**
	59.0000	**14.0000**	68.0000
	13.0000	**6.0000**	16.5000
UCI20 (Parkinsons)	**0.1538**	0.1724	0.1552
	22.0000	**9.0000**	48.0000
	7.5000	**5.0000**	15.0000
UCI21 (pima indians diabetes)	0.2630	**0.2609**	**0.2609**
	42.5000	**19.0000**	73.5000
	11.5000	**7.0000**	15.0000
UCI22 (Sonarall)	0.2976	**0.2742**	0.2823
	50.0000	**6.0000**	74.5000
	12.5000	**4.0000**	17.0000

Tabla 6.9: La mediana de la mejor solución de problemas de clasificación, el primer valor indica la aptitud (fitness), el segundo indica el numero de nodos y el tercer valor la profundidad.

componente de enfoque de GP, el rendimiento parece estar comprometido. Mientras que el bloat puede ser controlado por cada variante, el rendimiento en los casos de prueba se reduce significativamente en varios casos y en algunos sustancialmente, ver Figuras 6.11 a 6.19, en particular para *neat*-GP-FS. Estos resultados sugieren que los tres mecanismos contribuyen a la producción de soluciones de alta calidad.

Sin embargo, el cruce-NEAT, como el operador de búsqueda principal, parece ser dependiente de dominio ya que en problemas de clasificación los resultados obtenidos fueron iguales o mejores que el cruce estándar de GP, pero en problemas de regresión simbólica solo alcanzo igualar los resultados del cruce estándar de GP. Por otro lado, el cruce estándar, que se prueba con *neat*-GP-SC, parece mejorar el rendimiento en la regresión simbólica, logrando mejores resultados con respecto a la prueba de aptitud.

No obstante, es importante señalar que *neat*-GP-SC tiende a producir árboles de mayor tamaño que otras variantes de *neat*-GP, pero las poblaciones sigue siendo significativamente pequeñas a comparación de GP.

Otra característica interesante es considerar la profundidad de los programas

Tabla 6.9, donde *neat*-GP-SC produce árboles con una profundidad mayor en comparación con *neat*-GP.

Por otro parte, para los problemas de clasificación el mtodo *neat*-GP logra claramente los mejores resultados, superando a GP basado en error de clasificación, y controla el bloat sustancialmente mejor que *neat*-GP-SC. De hecho, si sólo el mejor individuo se considera ver Tabla 6.9, la reducción de código puede ser tan alta como 88 %. Por lo tanto, parece que cada operador de cruce tiene un claro dominio de la competencia, dentro del método *neat*-GP.

6.5. Conclusión

neat-GP es capaz de reducir el crecimiento del código sin disminuir el rendimiento de GP, y en muchos casos mejorar la calidad de las soluciones evolucionadas.

neat-GP tiende a generar una distribución de tamaño de programas, que es consistente con lo que sugiere Flat-OE, para contrarrestar el efecto de bloat.

El rango de tamaño de los programas varía con cada problema, lo que indica que neat-GP es capaz de enfocar automáticamente la búsqueda en un rango específico de tamaño, en función de la aptitud.

Los experimentos de la sección anterior donde se utilizan distintas variantes de *neat*-GP, examinan el efecto o contribución de cada uno de los principales componentes dentro de neat-GP. En general, los componentes más importantes son las restricciones en el mecanismo de selección, penalización de aptitud y la restricción de cruce entre especies del mismo tipo.

En las tres variantes *neat*-GP-Sel, *neat*-GP-FS y *neat*-GP-Spe respectivamente, cuando se elimina cualquiera de los componentes y un enfoque estándar de GP se utilizado en su lugar, el rendimiento parece estar comprometido.

neat-GP es capaz de controlar el bloat de una forma natural, ya que no consume recursos de cómputo en procesos e individuos que serán descartados sin contribuir el la búsqueda de una mejor solución, al adoptar técnicas generales del método de NEAT en distintas fases del proceso evolutivo:

- Evaluación con aptitud compartida para cada especie.

- Método de seleccin basado en NEAT, donde se selecciona un número predefinido de los padres para la sobrevivencia, que también tiene brinda la posibilidad de reproducción, que es proporcional al valor de aptitud de la descendencia.
- Reproducción sólo entre la misma especie e intraespecie cuando no sea posible realizar el cruce con individuos de la misma especie.
- La supervivencia es altamente elitista, donde sólo el mejor de los padres y descendencia sobreviven.
- Nuevo operador de cruce-NEAT, para optimización de estructuras similares.

Creando un nuevo método de GP (*neat*-GP vs GP estándar) que no desperdiciar recursos, se ejecuta en un menor tiempo, con un crecimiento estructural estable, con menor numero de nodos y profundidad promedio en la población, con un rendimiento competitivo o mejor en la solución del problemas.

Capítulo 7

Conclusión General

El fenómeno de bloat ha estado presente desde el princio de GP, con el crecimiento de tamaño(nodos que contiene un árbol) promedio en la población sin presentar una mejora en la aptitud de la mejor solución, provocando varios efectos secundarios indeseables, como la evaluación de programas que requieren más recursos y tiempo de procesamiento, y soluciones de gran tamaño son difíciles de interpretar para los usuarios de GP.

Se han desarrollado una gran variedad de métodos de control de bloat [26], pero solo el método de OE [4] presenta buenos resultados, pero sacrificando una gran cantidad de recursos de tal forma que el beneficio en el gasto de recursos es poco, lo que motivo a Silva a desarrollar un método con los mismo principios pero gastando menos recursos [24], el cual fracaso, pero demostró los puntos clave de OE que controlan el bloat al desarrollar una variante de OE llamada Flat-OE.

La presente tesis toma los puntos clave de Flat-OE que son:

- Mantener una distribución de todo los de tamaños en la población(inducir una distribución de árboles con tendencias de tamaño uniforme en la población).
- Crecer lo suficiente para encontrar la solución(árboles con suficiente complejidad (tamaño) para formar la solución).

Tomando los puntos clave, se formulo un enfoque distinto a los métodos actuales

para controlar el bloat en base a los descubrimientos de Silva [24], se realizo una investigación sobre el algoritmo de Neuroevolution of Augmenting Topologies (NEAT [27], ya que los usuarios del algoritmo suponen que no presenta bloat en su población.

La investigación sobre NEAT confirmo que el algoritmo no presenta bloat en su población, lo cual es verdad solo si se cambia la configuración a favorecer las redes neuronales de menor tamaño como lo recomienda Silva para métodos de GP en [24]. Los resultados obtenidos en esta investigación fueron publicados en el congreso EVOstar 2014 con el nombre NEAT, Theres No Bloat [29]

Pero bien validando que un algoritmo para redes neuronales no presenta bloat en base a los principios de anti-bloat de GP, es una buena iniciativa para generar un nuevo método contra el bloat en GP, por lo cual, la investigación se centro en analizar el algoritmo de NEAT en busca de sus cualidades anti-bloat.

Las cualidades encontradas son las siguientes:

- Iniciar con una población estructural mínima, que incrementara en base a los mejores individuos.
- Aptitud compartida por especie, para que una especie no sobre poblé toda la población.
- Una selección de sobrevivivencia elitista y fija, que permite el remplazo de la población, con un impacto fijo que genera cambios suaves en la topología de la población.
- Reproducción sólo entre la misma especie e intraespecie cuando no sea posible realizar el cruce con individuos de la misma especie.
- Protección elitista, al no penalizar a los mejores individuos de cada especie.
- Operador de cruce especializado para conservar las características topológicas de la red neuronal.

Al detectar las cualidades clave del algoritmo, se implementaron dentro de GP, esta implementación fue una generalización simplificada ya que GP no utiliza redes neuronales sino árboles para representar a los individuos de la población.

La implementación llamada neat-GP es capaz de controlar el bloat de una forma natural, ya que no consume recursos de cómputo en procesos e individuos que serán descartados sin contribuir en la búsqueda de una mejor solución. Los resultados de la investigación fueron sometidos a la revista de Evolutionary Computation de Massachusetts Institute of Technology (MIT) bajo el titulo de neat Genetic Programming: Controlling Bloat Naturally.

7.1. Resultados relacionados

Al comienzo de la investigación se buscaban formas de enfrentar el bloat(como es la carga de procesamiento que requiere evaluar individuos con bloat) de una forma mas efectiva. Una de las primeras ideas fue dividir y vencer por medio de computo distribuido con la interface de Berkeley Open Infrastructure for Network Computing (BOINC) para dividir la carga de trabajo en equipos voluntarios y con la librería de Java-based Evolutionary Computation (ECJ) para utilizar GP, con lo que se logro reducir el tiempo de ejecución significativamente, los resultados concretos pueden ser vistos en el articulo A Hybrid ECJ+BOINC Tool for Distributed Evolutionary Algorithms [5] que fue presentado en la conferencia internacional Regional Academic Encounter (ERA).

Otro trabajo desarrollado fue mejorar el método de Multi-dimensional Multi-class Genetic Programming(M2GP) [7] que trabajaba solo con poblaciones de dimension fija (el numero de nodos que contiene la raíz del árbol), la nueva implementación sigue los mismos principios desarrollados en la tesis, como comenzar con una dimension mínima y aumentar el numero de dimensiones de forma dinámica por medio del operador de mutación, siguiendo elitistamente a los mejores individuos de la población, las modificaciones realizadas permitieron al algoritmo utilizar una población multidimensional y omitir la selección de dimension inicial que limitaba a la solución a dimensiones de óptimos locales. La investigacin fue sometida al EuroGP con el nombre M3GP Multiclass Classification with GP.

7.2. Trabajo futuro

El nuevo método *neat*-GP presenta resultados prometedores en problemas de regresión simbólica y de clasificación. Siendo prometedores por generar soluciones pequeñas a comparaciones de GP, pero también manteniendo la calidad de la solución y en casos mejorando los resultados obtenidos. Mitigando el fenómeno de bloat en la población de GP, los efectos secundarios como la evaluación de los programas grandes en la población que requieren más tiempo de procesamiento y soluciones de gran tamaño que son difíciles de interpretar, son eliminados. Lo que permite aplicar otras técnicas conocidas que no suelen ser efectivas en individuos de gran tamaño como el método de búsqueda local(local search) [3] que optimiza los árboles en regiones especificas en busca de un mejor resultado, por lo tanto un árbol mas pequeño favorecería al método, otro método donde puede ser aplicado para optimizar la búsqueda es en Multi-dimensional Multi-class Genetic Programming (M2GP)[7] donde se generan grandes soluciones por la gran cantidad de nodos que se manejan en los árboles multi-dimensionales para la transformación de las muestras a clasificar.

El rango de aplicación de *neat*-GP puede ser todo donde halla sido aplicado GP, esto brinda una gran área de oportunidad para comparar resultados y el análisis de estructuras emergentes, que permitan detectar las debilidades y fortalezas de este nuevo método de GP [19].

Glosario

Parity: Se usan en telecomunicaciones para detectar, y en algunos casos corregir, errores en la transmisión. Para ellos se añade en el origen un bit extra llamado bit de paridad a los n bits que forman el carácter original.

hiper-heurística: Una heurística es un método basado en la experiencia que puede utilizarse como ayuda para resolver problemas de diseño. Pero una hiper-heurística es utilizado para generar heurística.

off the shelf: Traducido al español es tomado del estante, que puede ser utilizado de forma genérica, por lo tanto no requiere modificación o ajustes para ser utilizado.

Regresión lineal: La regresión lineal o ajuste lineal es un método matemático que modela la relación entre una variable dependiente Y, las variables independientes X_i y un término aleatorio ε.

bloat: Una traducción bloat al espaól es hinchazón. Para la GP el el fenómeno que se produce cuando los árboles de programas tienden a crecer innecesariamente grandes, sin un aumento correspondiente en la aptitud.

subyacen: Estar algo oculto o por debajo de otra cosa.

feedforward: Para el ámbito de redes neuronales es una red sin retroalimentación, esto implica que las salidas de una capa de la red van a la siguiente capa, pero no a capas anteriores.

benchmark: Son problemas utilizados para medir el rendimiento de un sistema o componente del mismo, frecuentemente en comparación con otros problemas conocido que sirven como referencia para medir los resultados obtenidos.

sintáctica: Corresponde al análisis de la relación existente entre los distintos

símbolos o signos del lenguaje.

semántica: Para la programación genética la semántica es el estudio y relación de los vectores de salida que produce el árbol.

fitness: En español es aptitud, es el criterio de evaluación numérica que presenta los individuos de una población para sobrevivir.

lenguajes imperativos: Los programas imperativos son un conjunto de instrucciones que le indican al computadora cómo realizar una tarea.

correlación: Indica la fuerza y la dirección de una relación lineal y proporcionalidad entre dos variables estadísticas. Se considera que dos variables cuantitativas están correlacionadas cuando los valores de una de ellas varían sistemáticamente con respecto a los valores homónimos de la otra.

Brood recombination: Método desarrollado para producir una gran cantidad de descendencia de un o varios individuo, donde solo se seleccionan los mejor **??**.

XOR: Representa la función de la desigualdad, es decir, la salida es verdadera si las entradas no son iguales, de otro modo el resultado es falso.

sesgada: Que no es objetiva o imparcial, sino que está condicionada para favorecer determinados intereses.

Koza: John R. Koza. es el creador de la programación genética de tipo árbol.

intrínsecas: Para designar lo que corresponde a un objeto por razón de su naturaleza y no por su relación con otro.

uniforme: Que no presenta variaciones en su conjunto, en su totalidad o en su duración.

sustancial: Es un concepto primitivo que representa una cosa cualitativamente indeterminada, esencial, fundamental o de gran relevancia.

plug and play: En espaol enchufar y usar, un concepto mas cercano a la programación es una librería lista para ser utilizada, si la necesidad de realizar ajustes.

multimodal: La utilización de varios métodos para simplificar la operación o transmisión objetivo.

genoma: Es el conjunto de genes contenidos en los cromosomas, lo que puede interpretarse como la totalidad de la información genética que posee un organismo o una especie en particular.

Acrónimos

SVM: Support Vector Machine / Maquina de Soporte Vectorial
Flat-OE: Flat Operator Equalization
NEAT: NeuroEvolution of Augmenting Topologies
EC: Evolutionary Computation / Computo Evolutivo
NNets: Neuronal Networks / Redes Neuronales
GP: Genetic Programming / Programación Genética
UCI: University of California Irvine
GA: Genetic Algorithms / Algoritmos Genéticos
FCBT: Fitness Causes Bloat Theory
OE: Operator Equalisation
BOINC: Berkeley Open Infrastructure for Network Computing
ECJ: Java-based Evolutionary Computation
MIT: Massachusetts Institute of Technology
M2GP: Multi-dimensional Multi-class Genetic Programming
M3GP: M2GP with multidimensional populations

Bibliografía

[1] Lee Altenberg. The evolution of evolvability in genetic programming. En Kenneth E. Kinnear, Jr., ed., *Advances in Genetic Programming*, págs. 47–74. MIT Press, Cambridge, MA, USA, 1994.

[2] K. Bache y M. Lichman. UCI machine learning repository. 2013. URL http://archive.ics.uci.edu/ml.

[3] Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim, y Kay Chen Tan. A multi-facet survey on memetic computation. *IEEE Trans. Evolutionary Computation*, 15(5):591–607, 2011. URL http://dblp.uni-trier.de/db/journals/tec/tec15.html#ChenOLT11.

[4] Stephen Dignum y Riccardo Poli. Operator equalisation and bloat free gp. En *Proceedings of the 11th European conference on Genetic programming*, EuroGP'08, págs. 110–121. Springer-Verlag, Berlin, Heidelberg, 2008.

[5] Francisco Chvez Enrique Mediero Francisco Fernndez de Vega, Leonardo Trujillo y Luis Muoz. A hybrid ecj+boinc tool for distributed evolutionary algorithms. *Research in Computing Science,Control and Communications*, 69(1):120–130, 2014.

[6] David E. Goldberg y Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. En *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, págs. 41–49. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1987.

[7] Vijay Ingalalli, Sara Silva, Mauro Castelli, y Leonardo Vanneschi. A multi-

dimensional genetic programming approach for multi-class classification problems. En *EuroGP-2014*. Springer, 2014.

[8] John Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3):251–284, 2010.

[9] William B. Langdon y Riccardo Poli. Fitness causes bloat. En *Proceedings of the Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, págs. 13–22. Springer-Verlag, 1997.

[10] William B. Langdon y Riccardo Poli. Fitness causes bloat: Mutation. En *Proceedings of the First European Workshop on Genetic Programming*, EuroGP '98, págs. 37–48. Springer-Verlag, London, UK, UK, 1998.

[11] Joel Lehman y Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.*, 19(2):189–223, 2011.

[12] Yuliana Martínez, Enrique Naredo, Leonardo Trujillo, y Edgar Galván López. Searching for novel regression functions. En *Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC)*, págs. 16–23. IEEE Press, 2013.

[13] Yuliana Martínez, Leonardo Trujillo, Enrique Naredo, y Pierrick Legrand. A comparison of fitness-case sampling methods for symbolic regression with genetic programming. En *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, págs. 201–212. 2014.

[14] Trent McConaghy. FFX: Fast, scalable, deterministic symbolic regression technology. En Rick Riolo, Ekaterina Vladislavleva, y Jason H. Moore, eds., *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, cap. 13, págs. 235–260. Springer, Ann Arbor, USA, 2011.

[15] James McDermott, David R. White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, y Una-May O'Reilly. Genetic programming needs better benchmarks. En *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, págs. 791–798. ACM, New York, NY, USA, 2012.

[16] E. Naredo, L. Trujillo, y Y. Martínez. Searching for novel classifiers. En *Proceedings from the 16th European Conference on Genetic Programming, EuroGP 2013*, tomo 7831 de *LNCS*, págs. 145–156. Springer-Verlag, 2013.

[17] Enrique Naredo y Leonardo Trujillo. Searching for novel clustering programs. En *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, GECCO '13, págs. 1093–1100. ACM, New York, NY, USA, 2013.

[18] Michael OÑeill, Leonardo Vanneschi, Steven Gustafson, y Wolfgang Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.

[19] Riccardo Poli, Mario Graff, y Nicholas Freitag McPhee. Free lunches for function and program induction. En Ivan I. Garibay et al., eds., *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms (FOGA)*, págs. 183–194. ACM, 2009.

[20] Riccardo Poli, William B. Langdon, y Stephen Dignum. On the limiting distribution of program sizes in tree-based genetic programming. En *Proceedings of the 10th European conference on Genetic programming*, EuroGP'07, págs. 193–204. Springer-Verlag, Berlin, Heidelberg, 2007.

[21] Riccardo Poli, William B. Langdon, y Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[22] Riccardo Poli, Leonardo Vanneschi, William B. Langdon, y Nicholas Freitag Mcphee. Theoretical results in genetic programming: the next ten years? *Genetic Programming and Evolvable Machines*, 11(3-4):285–320, 2010.

[23] S. Silva y J. Almeida. Gplab–a genetic programming toolbox for matlab. En L. Gregersen, ed., *Proceedings of the Nordic MATLAB conference*, págs. 273–278. 2003.

[24] Sara Silva. Reassembling operator equalisation: a secret revealed. En *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, págs. 1395–1402. ACM, New York, NY, USA, 2011.

[25] Sara Silva y Ernesto Costa. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179, 2009.

[26] Sara Silva, Stephen Dignum, y Leonardo Vanneschi. Operator equalisation for bloat free genetic programming and a survey of bloat control methods. *Genetic Programming and Evolvable Machines*, 13(2):197–238, 2012.

[27] Kenneth O. Stanley y Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, 2002.

[28] Leonardo Trujillo, Luis Munoz, Enrique Naredo, y Yuliana Martinez. NEAT, there's no bloat. En M. Nicolau et al., eds., *17th European Conference on Genetic Programming*, tomo 8599 de *LNCS*, págs. 174–185. Springer, 2014.

[29] Leonardo Trujillo, Luis Muoz, Enrique Naredo, y Yuliana Martnez. Neat, there?s no bloat. En Miguel Nicolau, Krzysztof Krawiec, MalcolmI. Heywood, Mauro Castelli, Pablo Garca-Snchez, JuanJ. Merelo, VictorM. Rivas Santos, y Kevin Sim, eds., *Genetic Programming*, tomo 8599 de *Lecture Notes in Computer Science*, págs. 174–185. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44302-6. doi:10.1007/978-3-662-44303-3_15. URL http://dx.doi.org/10.1007/978-3-662-44303-3_15.

[30] Leonardo Trujillo, Enrique Naredo, y Yuliana Martnez. Preliminary study of bloat in genetic programming with behavior-based search. En Michael Emmerich et al., eds., *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV*, tomo 227 de *Advances in Intelligent Systems and Computing*, págs. 293–305. Springer International Publishing, 2013.

[31] Leonardo Trujillo, Gustavo Olague, Evelyne Lutton, Francisco Fernndez de Vega, Len Dozal, y Eddie Clemente. Speciation in behavioral space for evolutionary robotics. *Journal of Intelligent & Robotic Systems*, 64(3-4):323–351, 2011.

[32] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael OÑeill, R. I. Mckay, y Edgar Galván-López. Semantically-based crossover in genetic programming: applica-

tion to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

[33] Leonardo Vanneschi, Mauro Castelli, y Sara Silva. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214, 2014.

[34] David R. White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W. Goldman, Gabriel Kronberger, Wojciech Jaskowski, Una-May O'Reilly, y Sean Luke. Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, 2013.