Programación Genética y aplicaciones a Robótica: Cortadora de Pasto y Resolución de Laberintos

Navarro, F.[1], Remis, L.[1], Santochi, D.[1], Lenis, J.[1], Will, A.[1,2], y Rodriguez, S.[1,2]
1 - Universidad Nacional de Tucumán – Facultad de Ciencias Exactas y Tecnología
fnxirom@gmail.com, luisremis@gmail.com,awill@herrera.unt.edu.ar
2 – Centro de Investigación en Tecnologías Avanzadas de Tucumán (C.I.T.A.T.), U.T.N., F.R.T.
awill@citat.org.arsrodriguez@citat.org.ar

***Resumen*:**

En el presente trabajo se presenta un sistema para resolver el Problema de la Cortadora de Pasto y la Resolución de Laberintos, basado en Programación Genética. En ambos casos se plantearon escenarios de creciente nivel de complejidad, atacando además el caso de un solo escenario y luego varios escenarios simultáneos, con el objetivo de obtener una estrategia general y no la optimización de un solo laberinto o escenario con obstáculos. Los resultados obtenidos son satisfactorios, resolviendo en el caso de la Cortadora de pasto series de escenarios de hasta 20x20 con 20% de obstáculos. En el caso de laberintos, se resolvieron laberintos multicursales con bucles y complejos hasta de tamaño 30x30, y series de laberintos multicursales con bucles hasta de tamaño 19x17.

I. INTRODUCCIÓN

La programación genética es una metodología basada en algoritmos genéticos que ha sido utilizada con éxito en áeras tan diversas como medicina, biología molecular, economía, control de procesos industriales, diseño de circuitos, etc. Incluso han recibido patentes y han realizado descubrimientos que compiten con invenciones humanas [1][2][3]. Existen dos formas principales de Programación Genética, que se distinguen por la forma en que se codifican los problemas: En uno (tree-based GP), los individuos son árboles, y en otro, los individuos tienen una estructura lineal (Linear GP). Para el presente trabajo y el framework asociado, se eligió la estructura de Tree-Based GP, donde cada individuo esta implementado usando la estructura de dato árbol [1][4][5][6]. Este árbol contiene las instrucciones y flujos de control necesarios para implementar un programa de control de un robot correcto, un sistema de diagnóstico en el caso de Medicina, o un circuito en el caso de electrónica y bioelectrónica.

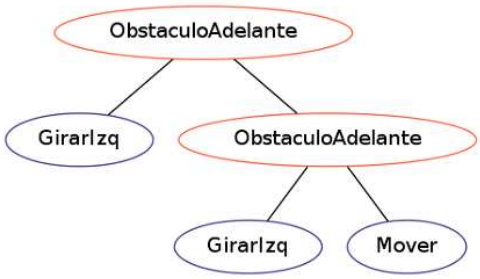

Fig. 1. Árbol de programa de Cortadora de Pasto

A diferencia de los algoritmos genéticos donde sólo se utilizan valores, la programación genética utiliza un conjunto de funciones y un conjunto de terminales en el momento de armar los árboles que representarán los individuos. El conjunto de funciones constituye los nodos internos del árbol, y contiene las operaciones posibles del dominio (funciones a evaluar, controles de flujo del programa, comparaciones lógicas, resultados de sensores, etc). Las terminales contienen los parámetros y acciones a tomar (variables, constantes, acciones a tomar por el robot) y representan los nodos hoja del árbol. En general el algoritmo cuenta con una tabla de las funciones disponibles, y una tabla separada con las terminales disponibles para un problema en particular.

De manera similar a los Algoritmos Genéticos, en Programación Genética se genera una población inicial de forma aleatoria, esa población sufre los procesos de Selección, Cruzamiento y Mutación, lo cual da como resultado una nueva generación. Este proceso se repite un número predeterminado de veces o hasta que se cumpla una condición de terminación. En la versión actual del framework, se implementó un Algoritmo Genético para Programación Genética, de tipo poblacional, cuyas características específicas se detallan en la sección siguiente.

El trabajo está estructurado de la siguiente manera: En la sección 2 se describen las características generales del sistema para Programación Genética desarrollado, en las secciones 3 y 4 se describen las implementaciones y pruebas desarrolladas para el problema de la cortadora de pasto y el problema del laberinto. La sección 5 contiene los resultados obtenidos y la sección 6 contiene las conclusiones y trabajo futuro.

Serie: Permite encadenar dos funciones o terminales, para que se ejecuten serialmente. Ejecuta el hijo izquierdo y luego el hijo derecho.

Las terminales son:

GirarDerecha: Gira 90° hacia la derecha.

GirarIzquierda: Gira 90° hacia la izquierda.

Mover: Avanza una casilla, en el sentido en el que se encuentra orientado el robot, cortando la grilla de pasto a la que se dirige. Si la casilla que tiene adelante es un obstáculo entonces no se mueve.

Una vez que se termina el procesamiento dictado por el recorrido de una rama del árbol, el procesamiento regresa al nodo raíz y se ejecuta nuevamente, hasta completar el número predeterminado de evaluaciones.

El usuario puede configurar el tamaño de la superficie de pasto así como la cantidad de obstáculos, pero la distribución de éstos es siempre aleatoria. También es posible determinar las veces que se ejecuta el árbol que representa una estrategia dada por una solución. Se entiende que este parámetro es importante debido a que si se ejecuta una pequeña cantidad de veces, ahorrará tiempo de proceso, pero podría llegar a cortar menor cantidad de césped. Si en cambio la estrategia se ejecuta una gran cantidad de veces, pasará lo contrario, mayor cantidad de césped podría ser cortado, pero el tiempo de proceso aumentará sin ser ese tiempo necesariamente útil en todos los casos.

B. Función de Evaluación

En versiones anteriores, el Fitness se calculaba contando la cantidad de pasto que lograba cortar la cortadora (1).

$$\frac{N°de\ grillas\ de\ pasto\ cortadas}{N°\ de\ grillas\ total\ -\ N°\ de\ grillas\ con\ obstáculo} \quad (1)$$

Se modificó la función de evaluación, teniendo en cuenta [10][2]. El número de elementos en la grilla se obtiene de multiplicar la cantidad de Filas por la cantidad de Columnas, a los que se debe sustraer el número de grillas con obstáculos para obtener la cantidad total de pasto posible para ser cortado.

Un aspecto importante a tener en cuenta a la hora de evaluar una solución, es que no toda grilla con pasto puede ser cortada, efectuando movimientos permitidos. En efecto, existen zonas no alcanzables, completamente rodeadas por piedras, a las que el robot no puede acceder dado que no puede saltar ni desplazarse en diagonal.

La cantidad de pasto cortado se determina al final de la evaluación de la solución, una vez que el robot cumplió la cantidad prefijada de veces que recorre el árbol solución.

$$\frac{(Cantidad_{Pasto\ cortado}) - c * RM}{filas * columnas - obstaculos} \quad (2)$$

Dond*e c* es una constante y *RM* es Movimientos Redundantes, o sea, cuantas veces pasa por un pasto que ya estaba cortado.

El valor de c es un peso, que mide la importancia de pasar más de una vez por un mismo lugar. En este caso, un peso de 0.2 me dice que es 5 veces más importante cortar un césped nuevo, que pasar de nuevo por el mismo lugar. Esto logra que la estrategia que se obtiene sea más inteligente que la anterior, realizando rutas más directas, ya que para una misma cantidad de pasto cortado, logra recorridos de menor longitud y con menor cantidad de movimientos no útiles. Esto se debe a la penalización efectuada por el valor c, que hace que un recorrido más largo sin necesidad que tenga un valor de fitness más alto, castigando los movimientos redundantes.

. Sin embargo, las pruebas realizadas muestran que, para mapas pequeños, los fitness obtenidos son generalmente más bajos.

Luego de realizar pruebas basadas en estas funciones de evaluación, se decidió hacer una combinación de las mismas (3) de manera de obtener las fidelidades y bondades de ambas soluciones, ya que la función de evaluación cumple un rol fundamental en la obtención de la estrategia.

$$F2 + \frac{1 - GeneracionActual}{CantidadTotalGeneraciones} * (F1 - F2) \quad (3)$$

Donde en el nuevo Fitness, F2 es el Fitness calculado con la segunda forma de evaluación (la más inteligente) y F1 es el Fitness calculado con la primera forma de evaluación. Al aplicar esta evaluación, lo que se consigue es que en las primeras generaciones, el sistema trabaje principalmente en base a la primera forma de evaluación, y a medida que se incrementan las generaciones, se comienza a utilizar la segunda forma de evaluación.

Por último se realizó otra función de evaluación (la cuarta que fue considerada). Se utiliza la primera función de evaluación hasta obtener un Fitness 1. Luego comienza a utilizar la segunda función de evaluación. Esto provoca que se obtenga una estrategia que corta todo el césped, aunque no sea de una forma muy inteligente. Luego comienza a usar la forma de evaluación más inteligente. Este fenómeno es conocido como *Parsimony Pressure* [11]

En cuanto a la Arquitectura del Algoritmo Genético, se encontró que la arquitectura utilizada junto con Subtree Crossover, con 200 generaciones para mapas 10x10 y aumentando con el tamaño del mapa y el porcentaje de obstáculos. Se encontró sin embargo que el método de Mutación podía ser mejorado, por lo que se probaron otros métodos de mutación Hoist Mutation, Subtree Mutation, Shrink Mutation, Permutation Mutation ([1][8][2][3][5]), de lo que Hoist Mutation resultó ser el más adecuado al problema (Tabla 3)

C. Vectores de Mapas

En este punto, y dado que hasta el momento sólo se utiliza un mapa por corrida, el sistema encuentra una solución optimizada para ese sólo mapa. Por lo tanto, y con la idea de que encuentre una solución más general y no tan enfocada a la resolución de un mapa en particular, se decidió implementar un vector de mapas a la hora de la evaluación.

Mediante esta nueva estrategia se obtuvieron menores valores de fitness, lo cual era esperado, debido a que el valor ahora es un valor promedio, sin embargo, la estrategia es más "inteligente", puesto que está probada en diferentes mapas.

IV. RESOLUCIÓN DE LABERINTOS

Una vez resuelto el problema de la cortadora de césped, se encaminaron los esfuerzos a obtener estrategias para un ambiente más hostil y complicado como es el laberinto. Existen muchos trabajos que hablan sobre las distintas formas de resolver laberintos, y si bien algunas funcionan en un amplio espectro de tipos de laberintos, ninguno se adapta a todas las distintas complejidades, reduciendo su solución a nichos estrictos ([12][13]). Un claro ejemplo de esto es la forma de solución colocando la mano derecha en una parad interna del laberinto (más adelante se abarcara los tipos de laberintos y sus soluciones), y avanzar con la mano en la pared hasta encontrar la salida. Otro ejemplo, un poco más interesante, es tomar el laberinto como un grafo e ir tomando todas las posibles alternativas, recorriéndolas en profundidad para encontrar una salida, mediante el descarte de las distintas ramas que no lleven a ningún lado más que a un callejón sin salida (tipo *branch and bound*). Así se va poniendo en una pila los caminos y decisiones tomadas en puntos de bifurcación, se eliminan de la pilalos caminos no llegan a ningún lugar útil, y se continúa el proceso hasta obtener una solución.

Estos métodos aparecen como los más usados en concursos de robótica aplicada a laberintos. Sin embargo no contemplan en general el caso de laberintos que contengan caminos en círculos (o grafos cíclicos).

.

A. Parámetros de Entrada y Codificación

Según los parámetros del problema original, el robot trabaja sobre una grilla bidimensional. En la presente implementación el laberinto se codificó como una matriz con tres valores posibles:0 = Casillero por el que no se pasó, 1= Pared, 2 = Casillero por el que se pasó, 3= Meta, Se resuelve el laberinto.

Los movimientos permitidos del robot son, girar a la derecha, girar a la izquierda y seguir adelante. Se introduce la función Serie equivalente a la función PROG2 de literatura, que indica que debe realizar las acciones de los nodos hijos. Cada árbol correspondiente a una solución candidato, es ejecutado un máximo número de veces a partir del nodo raíz.

El conjunto de funciones está conformado por 2 estructuras de control IF_THEN_ELSE y una función Serie:

CasilleroAdelante: Si hay Casillero no visitado adelante, ejecuta el hijo izquierdo, sino el hijo derecho.

ObstaculoAdelante: Si hay obstáculo adelante, ejecuta el hijo izquierdo, sino el hijo derecho.

Serie: Permite encadenar dos funciones o terminales, para que se ejecuten serialmente. Ejecuta el hijo izquierdo y luego el hijo derecho. Las terminales son:

GirarDerecha: Gira 90° hacia la derecha.

GirarIzquierda: Gira 90° hacia la izquierda.

Mover: Avanza una casilla, en el sentido en el que se encuentra orientado el robot, marcando el Casillero como visitado. Si el Casillero que tiene adelante es un obstáculo entonces no se mueve.

B. Función de Evaluación

La naturaleza de la evaluación usada es lo que marca la principal diferencia entre el problema de resolución de laberintos y el problema de la cortadora de césped, ya que para que se pueda llegar a una óptima solución, se debe tener una evaluación de verdadera calidad que refleje y resalte las diferencias entre una buena solución y una de menor calidad, y la solución buscada es esencialmente diferente en ambos problemas. En el laberinto, no se buscar recorrerlo completo (como en la cortadora de césped, que se busca pasar por toda la superficie posible), sino que se apunta a recorrer el menor camino posible encontrando una estrategia para la resolución de laberintos de similar complejidad.

Por lo anteriormente expuesto, existen varios puntos a tener en cuenta a la hora de buscar una función de evaluación de buena calidad:

- Si no se llega a la meta, se debe tratar de recorrer lo máximo posible.
- Se debe premiar las soluciones que lleguen a la meta, por sobre las que no lo hacen
- Se debe premiar a las soluciones que lleguen a la meta usando el menor camino.
- Se debe buscar un promedio entre fitness de distintos laberintos para comparar soluciones

Bajo estos principios, se decidió armar una función de evaluación que fundamentalmente premie a las soluciones que llegaban a la meta con un factor de premiación, de manera que cualquier solución que consiga resolver el laberinto mejore en gran medida su valor de Fitness.

Entonces, el cálculo del fitness se realiza mediante el uso de una bandera que tendrá cada mapa evaluado indicando si el robot llegó o no a la meta:

SI Laberinto Resuelto

$$Fitness = F_{premiacion} + (1 - MapaCompleto_{\%})(1 - F_{premiacion}) \quad (4)$$

SINO

$$Fitness = (1 - F_{premiacion}) * MapaCompleto_{\%} \quad (5)$$

De esta manera, se tienen funciones que asignan valores de fitness dependiendo si la solución evaluada pudo resolver efectivamente el laberinto, y premiando a la solución que pudo hacerlo.

Experimentalmente se determinó que las mejores soluciones se obtenían cuando se usaba como factor de premiación un valor entre 0,4 y 0,6, no habiendo diferencia significativa cuando se usaba cualquier valor dentro de este rango. Se eligió entonces como factor de premiación 0,5.

También se puede apreciar en las formulas anteriormente expuestas cómo el fitness mejora en el caso que no se llega a

resolver el laberinto, mientras más se explora el mismo. Sin embargo, nunca se obtiene un fitness mayor que el obtenido para cuando se logra llegar a la meta solamente. Cuando se llega a la meta, se marca una diferencia, otorgándole al fitness el valor de premiación, superando así cualquier otra solución que no llegó a la meta. Luego, cuando los individuos empiezan a converger a una solución que llega efectivamente a la meta, obtendrá mayor fitness la solución que haga menos recorrido, tal como se muestra en el segundo término de la formula. Este manejo diferente al anterior del Parsimony Pressure se debe a que la cercanía al objetivo no necesariamente es garantía de que la solución es correcta (ya que puede estarse acercando al objetivo por una vía muerta), a diferencia del caso de la cortadora de pasto, donde claramente cualquier solución que se acerque al 100% estará acercándose correctamente a su objetivo.

C. Vectores de Mapas

Al igual que en el caso de la cortadora de césped, una vez probado el método en laberintos específicos y obteniendo resultados positivos, se buscó complejizar la herramienta para encontrar una estrategia que sirva para un conjunto de laberintos de características similares. De esta manera el método no solo es capaz que encontrar una buena solución para un laberinto en particular, sino que además puede definir una estrategia que sirve para la resolución de laberintos de similar nivel de complejidad.

V. RESULTADOS OBTENIDOS

A. Cortadora de Pasto

Se realizaron distintas pruebas de escenarios simples (un solo escenario), de tamaños entre 10x10 hasta 50x50, y con porcentajes variables desde 5% hasta 30% y 40% en algunos casos. Los resultados muestran que corta toda la parte factible de ser cortada en la mayoría de las corridas (en algunos casos quedan zonas encerradas por obstáculos que resultan imposibles de cortar). También se realizaron pruebas con series de escenarios, llevando las pruebas hasta 10 escenarios de 20x20 con 20% de obstáculos en los que se obtuvo un fitness de 0.97, con 200 individuos y 500 generaciones. Los resultados se muestran en la Fig 3 y la Tabla 1 para mapas únicos y Tabla 2 para vectores de mapas.

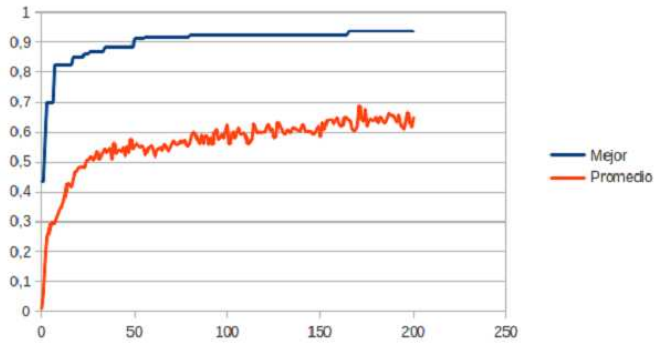

Fig 3. Generaciones vs Fitness para el problema de la Cortadora de Pasto

Mapa	Obstáculos	Individuos	Generaciones	Pasos	Fitness
10x10	5 %	200	200	400	1
10x10	10 %	200	200	400	1
10x10	15 %	200	200	400	1
10x10	20 %	200	200	200	1
10x10	25 %	200	200	200	1
10x10	30 %	200	200	200	0,9857
20x20	5 %	300	300	800	1
20x20	10 %	300	300	800	1
20x20	15 %	300	300	800	1
50x50	5 %	200	200	10000	0,9577

Tabla 1. Resultados obtenidos con las diferentes pruebas en el caso de la cortadora de pasto.

Mapa	**Obstáculos**	**Individuos**	**Generaciones**	**Pasos**	**Fitness**
10x10	10 %	200	200	400	1
10x10	10 %	400	200	400	1
20x20	20 %	500	200	400	0,9752

Tabla 2. Resultados obtenidos aplicados a vectores de mapas.

También se realizó una tanda de pruebas tendientes a determinar cuál de los diferentes tipos de mutación resultaba más adecuado al problema. Los resultados se muestran en la Tabla 3

Mutacion	**% Obstaculos**	**Generaciones**	**Fitness Promedio**
Sub-tree	10 %	200	0,4946
Hoist	10 %	200	0,6538
Shrink	10 %	200	0,4302
Permutation	10 %	200	0,5253

Tabla 3. Pruebas realizadas con diferentes tipos de Mutación para el problema de la cortadora de pasto

B. Resolución de Laberintos

Se demostró tener una rápida convergencia para laberintos simples de baja complejidad (Fig 4), así como para conjuntos de laberintos de alta complejidad (Fig. 5), confirmando la robustez y eficiencia del método utilizado. Se puede apreciar una clara diferencia en la cantidad de generaciones (eje horizontal) necesarias para la convergenciade la solución, ya que en Fig 4 solo se tienen laberintos únicos y de baja complejidad, y en Fig 5 se tienen vectores de laberintos de alta complejidad. De esta manera, se demuestra como mediante este método se obtiene una estrategia al problema de la resolución de laberintos complejos.

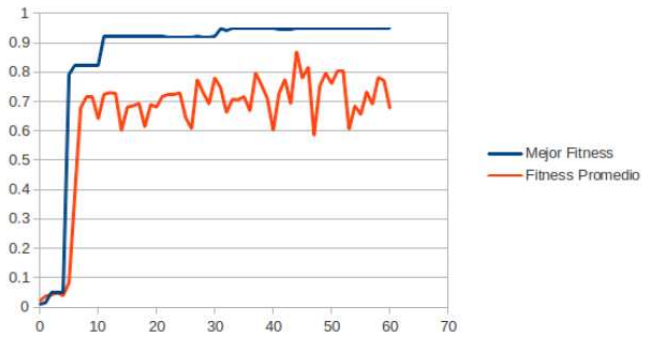

Fig 4. Generaciones vs Fitness para Laberintos simples de baja complejidad

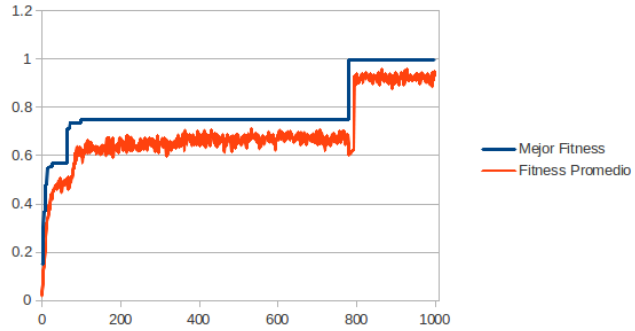

Fig 5. Generaciones vs Fitness para Series de Laberintos de Alta Complejidad

Tipo	Bucles	Tamaño	Vectores	Resuelto
Unicursal	No	19X17	No	1/1
Unicursal	Si	10X10	No	1/1
Multicursal	No	10X10	No	1/1
Multicursal	Si	10X10	No	1/1
Multicursal	Si	19X17	No	1/1
Multicursal	Si	24X24	No	1/1
Multicursal	Si	29X29	No	1/1
Multicursal	Si	34X34	No	1/1
Multicursal	Si	19x17	Si	5/5
Multicursal	Si	19x17	Si	7/7
Multicursal	Si	Varios	Si	6/7
Multicursal	Si	19x15	Si	7/8

Tabla 4. Resultados obtenidos para Laberintos

VI. CONCLUSIONES

Las aplicaciones desarrolladas alcanzaron las expectativas planteadas, tanto en tiempo como en performance. En las pruebas efectuadas el sistema probó ser estable a repeticiones, encontrando la solución correcta al problema con un gran porcentaje de éxito y en igual cantidad de individuos y generaciones. Esta es una de las características más difíciles de lograr debido a la aleatoriedad intrínseca de ambos procesos: La robustez y estabilidad de un programa basado en Algoritmos Genéticos, fuertemente basado en aleatoriedad, se puede apreciar cuando se obtienen similares soluciones en diferentes ejecuciones con los mismos parámetros.

Los resultados mostrados exponen claramente que la aplicación funciona incluso frente a condiciones complejas de resolución, ya que se utilizaron laberintos complejos (multicursal de conexión múltiple). Los resultados obtenidos coinciden en general con las soluciones intuitivas, sin repeticiones de movimientos, vueltas cerradas, ni longitudes excesivas.

En cuanto al problema de la cortadora de césped, se lograron muy buenos porcentajes de cortado de pasto en referencia a lo encontrado en la literatura [14], si se tiene en cuenta que la presente versión no implementa búsqueda de subprogramas.

El objetivo futuro es por un lado completar la implementación del robot físico, y por otro agregar complejidad a ambos problemas (inclinación del terreno, gasto de combustible, escenarios de mayor tamaño, etc.), y mejorar las técnicas implementadas en el sistema (búsqueda y reuso de subprorgamas, mejoras en la búsqueda de constantes, mejoras en el sistema de AG para abarcar otras posibilidades).

De esta manera se pretende mejorar la performance y utilidades del sistema, así como acercar las aplicaciones dadas a casos reales como cosechadoras mecánicas o cortadoras de pasto comerciales en escenarios con obstáculos.

REFERENCIAS

[1] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Lulu.com, 2008.

[2] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[3] J. R. Koza, *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, 1999.

[4] R. Poli and N. F. McPhee, “General schema theory for genetic programming with subtree-swapping crossover: Part II,” *Evol. Comput.*, vol. 11, no. 2, pp. 169–206, 2003.

[5] U.-M. O’Reilly, “Genetic programming: a tutorial introduction,” in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, Dublin, Ireland, 2011, pp. 861–874.

[6] L. Spector, K. Harrington, and T. Helmuth, “Tag-based modularity in tree-based genetic programming,” in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, Philadelphia, Pennsylvania, USA, 2012, pp. 815–822.

[7] P. Kouchakpour, A. Zaknich, and T. Br\&\#x00e4;unl, “A survey and taxonomy of performance improvement of canonical genetic programming,” *Knowl. Inf. Syst.*, vol. 21, no. 1, pp. 1–39, 2009.

[8] K. E. Kinnear, *Advances in Genetic Programming*. MIT Press, 1994.

[9] J. R. Woodward, “Modularity in genetic programming,” in *Proceedings of the 6th European conference on Genetic programming*, Essex, UK, 2003, pp. 254–263.

[10] D. Robilliard, S. Mahler, D. Verhaghe, and C. Fonlupt, “Santa fe trail hazards,” in *Proceedings of the 7th international conference on Artificial Evolution*, Lille, France, 2006, pp. 1–12.

[11] S. Luke and L. Panait, “Fighting Bloat with Nonparametric Parsimony Pressure,” in *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, 2002, pp. 411–421.

[12] M. M. Filesi, *El laberinto: Historia y mito*. Alba Editorial, 2009.

[13] A. Fisher, *The Amazing Book of Mazes*. Harry N. Abrams, 2006.

[14] J. A. Walker and J. F. Miller, “Embedded cartesian genetic programming and the lawnmower and hierarchical-if-and-only-if problems,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, Seattle, Washington, USA, 2006, pp. 911–918.