Memoria Práctica 3

Programación genética/Gramáticas Evolutivas

11/06/2025

—

Programación Evolutiva

—

Bryan Xavier Quilumba Farinango

1. Introducción

El objetivo de esta práctica es aplicar técnicas de Programación Genética (PG) para resolver el problema clásico de la hormiga artificial que sigue un rastro de comida. Específicamente, se aborda el "Rastro de Santa Fe", un camino irregular compuesto por 89 unidades de comida sobre un tablero toroidal de 32x32 celdas.

La hormiga comienza en la casilla (0,0) mirando hacia el Este. Debe desarrollar un programa (estrategia) que le permita consumir la mayor cantidad posible de comida, idealmente las 89 unidades, dentro de un límite máximo de 400 pasos (acciones de movimiento o giro). El desafío radica en la irregularidad del camino, que incluye giros, saltos y esquinas con huecos, requiriendo que la hormiga evolucione una estrategia robusta utilizando un conjunto limitado de acciones.

ESTA ES LA VERSION 2. Con cambios en la arquitectura, lógica del fitness y limpieza.

2. Detalles de Implementación

Se desarrolló una aplicación en Java con interfaz gráfica Swing para implementar y experimentar con el algoritmo de Programación Genética aplicado a este problema.

2.1. Representación del Individuo (Programa)

Cada individuo en la población representa un programa que dicta el comportamiento de la hormiga. La estructura del programa es un árbol, construido utilizando los siguientes conjuntos de terminales y funciones:

- **Terminales (Acciones):**
  - AVANZA: Mueve la hormiga una casilla hacia adelante en la dirección actual. Consume comida si la encuentra. Consume 1 paso.
  - DERECHA: Gira la hormiga 90 grados a la derecha sobre sí misma. Consume 1 paso.
  - IZQUIERDA: Gira la hormiga 90 grados a la izquierda sobre sí misma. Consume 1 paso.
- **Funciones (Nodos Internos):**
  - SICOMIDA(rama_si, rama_no): Ejecuta rama_si si hay comida en la casilla directamente enfrente de la hormiga; en caso contrario, ejecuta rama_no. La detección no consume paso.
  - PROG2(accion1, accion2): Ejecuta accion1 y luego accion2 secuencialmente.
  - PROG3(accion1, accion2, accion3): Ejecuta accion1, accion2 y accion3 secuencialmente.

2.2. Función de Aptitud (Fitness)

El objetivo es maximizar la cantidad de comida recogida. La función de aptitud (o fitness) de un individuo se define como:

- **Fitness = Número total de unidades de comida consumidas** por la hormiga al ejecutar su programa.

La evaluación de un individuo se detiene cuando se alcanza una de las siguientes condiciones:

- Se han consumido las 89 unidades de comida (solución óptima).
- Se ha alcanzado el límite de 400 pasos ejecutados.

2.3. Parámetros del Algoritmo Genético

La interfaz gráfica permite configurar los siguientes parámetros clave del algoritmo:

- **Tamaño de Población:** Número de individuos en cada generación.
- **Número de Generaciones:** Límite de iteraciones del ciclo evolutivo.
- **Probabilidad de Cruce:** Probabilidad de que dos individuos seleccionados se crucen.
- **Probabilidad de Mutación:** Probabilidad de que un individuo seleccionado mute (aplicando el método de mutación elegido).
- **Profundidad Inicial:** Profundidad máxima de los árboles generados en la población inicial.
- **Método Inicialización:** Técnica para crear los árboles iniciales ("Ramped", "Creciente", "Completa").
- **Profundidad Máxima (Bloating):** Límite de profundidad aplicado después de operaciones genéticas para controlar el crecimiento excesivo de los árboles.
- **Método de Selección:** Algoritmo para elegir los padres de la siguiente generación (e.g., "Torneo Determinístico", "Ruleta").
- **Método de Mutación:** Tipo específico de mutación a aplicar (e.g., "Subárbol", "Terminal").
- **Porcentaje de Elitismo:** Porcentaje de los mejores individuos de una generación que pasan directamente a la siguiente.
- (Opcional: Añadir otros como Tamaño de Torneo si es configurable)

2.4. Operadores Genéticos Implementados

El ciclo evolutivo utiliza los siguientes operadores:

- **Inicialización:** Se implementaron los métodos Completa, Creciente y Ramped Half-and-Half (seleccionable desde la GUI) para generar la población inicial con diversidad estructural.
- **Selección:** Se implementaron varios métodos (Ruleta, Torneo Determinístico/Probabilístico, Estocástico Universal, Truncamiento, Restos, Ranking) seleccionables desde la GUI. El torneo es una opción robusta por defecto.
- **Cruce:** Se implementó el cruce estándar de GP: **Cruce de Subárbol**, donde se intercambian subárboles seleccionados aleatoriamente entre dos individuos padres.
- **Mutación:** Se implementaron cuatro tipos de mutación (seleccionables desde la GUI):
    - Mutación Terminal: Cambia un nodo terminal aleatorio por otro.

    - Mutación Funcional: Cambia un nodo de función aleatorio por otro de la misma aridad.
    - Mutación de Subárbol: Reemplaza un subárbol aleatorio por uno nuevo generado aleatoriamente.
    - Mutación Hoist: Reemplaza un subárbol aleatorio por uno de sus propios subárboles inmediatos.

- **Elitismo:** Permite preservar un porcentaje configurable de los mejores individuos de cada generación.
- **Control de Bloating:** Se aplica una **limitación estricta de profundidad máxima** después de las operaciones de cruce y mutación para prevenir el crecimiento descontrolado de los programas.

2.5. Arquitectura de la Aplicación

La aplicación se estructuró siguiendo un modelo básico Modelo-Vista-Controlador (aunque no estrictamente separado):

- **Modelo (modelo):** Contiene la lógica del problema y del AG.
    - IndividuoHormiga: Representa a la hormiga, su árbol genético y su estado en la simulación. Contiene la lógica de evaluar, ejecutarAccion, operadores de mutación/cruce a nivel individual.
    - PoblacionHormiga: Gestiona la colección de individuos, implementa el ciclo generacional, selección, llamadas a cruce/mutación a nivel poblacional.
    - AlgoritmoGenetico: Clase (o placeholder) que encapsula los parámetros del AG.
    - Nodo, Funcion, Terminal: Clases/Enums para la estructura del árbol.

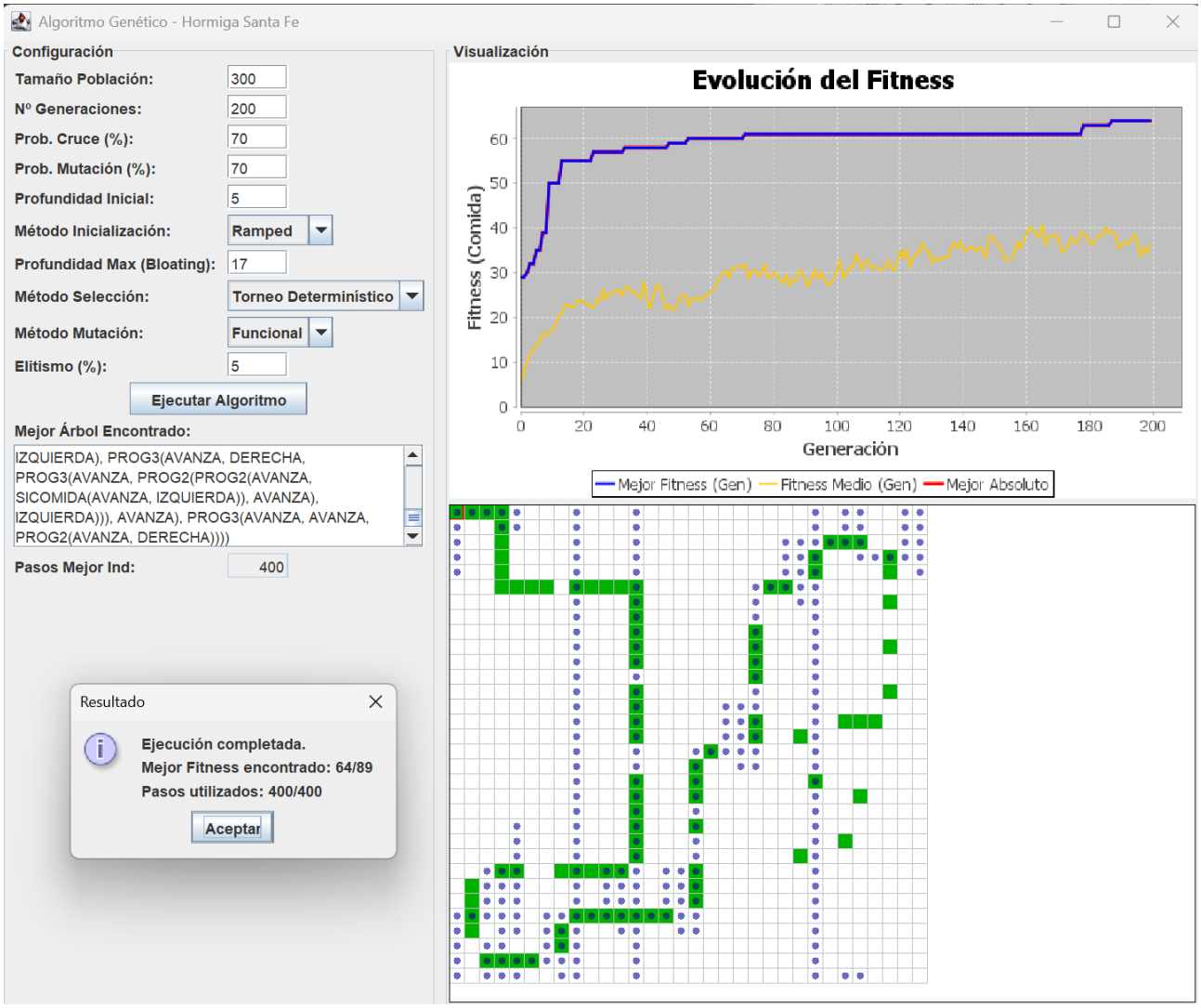
- **Vista (vista):** Responsable de la interfaz gráfica.
    - JFramePoblacionHormiga: Ventana principal (JFrame) que contiene los paneles de configuración y visualización.
    - PanelMapa: JPanel personalizado para dibujar el estado del tablero y la ruta.
    - Uso de JFreeChart para las gráficas de evolución.

- **Controlador (Integrado en la Vista):** La lógica de interacción (ActionListener del botón, SwingWorker) que lee los parámetros de la vista, configura el modelo (AlgoritmoGenetico), ejecuta el AG (PoblacionHormiga) y actualiza la vista (gráfica, mapa, área de texto) con los resultados. Se utiliza SwingWorker para mantener la responsividad de la GUI durante la ejecución del AG.

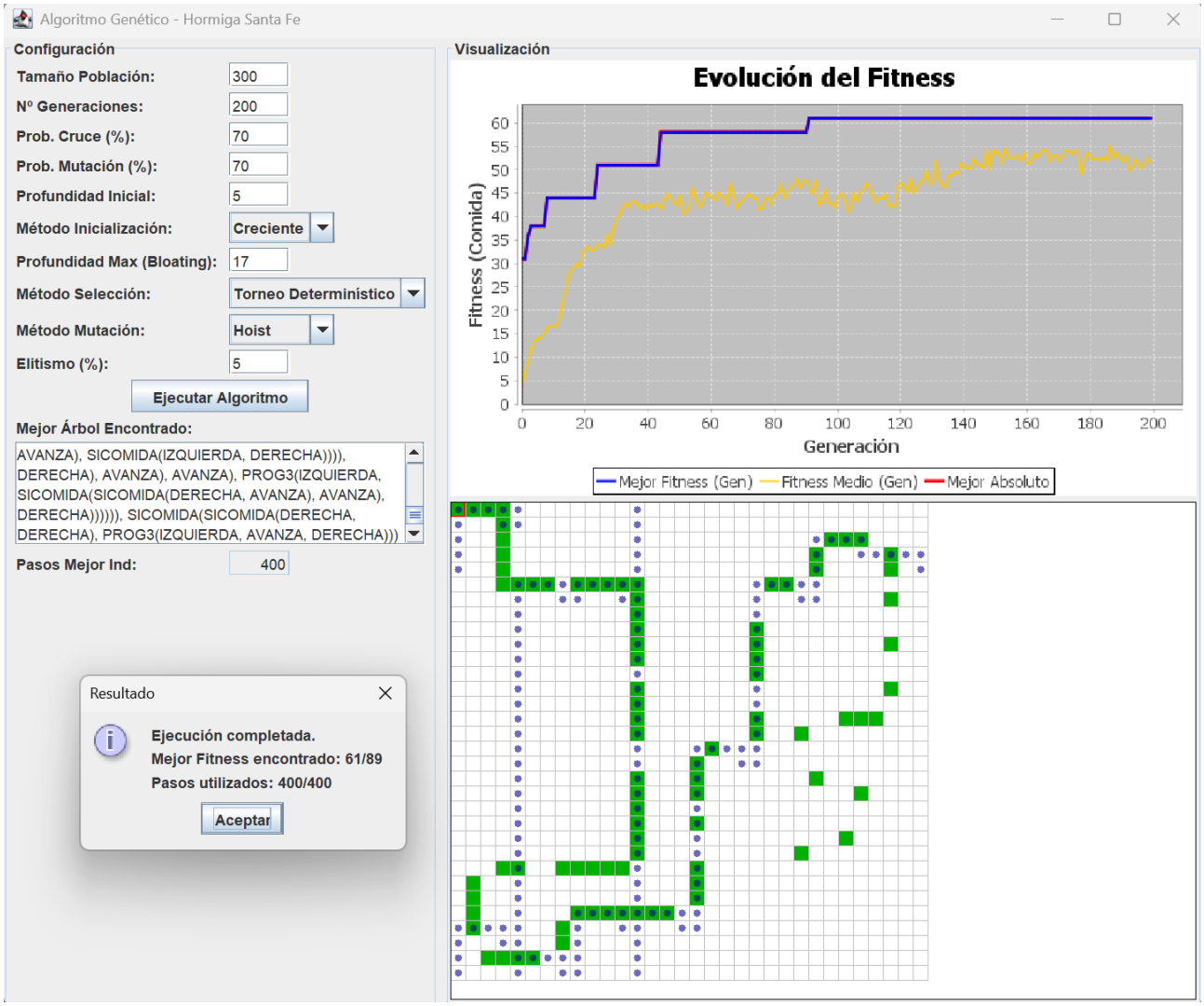
3. Resultados Experimentales (Aplicación Java Propia)

A continuación, se presentan los resultados de 5 ejecuciones representativas

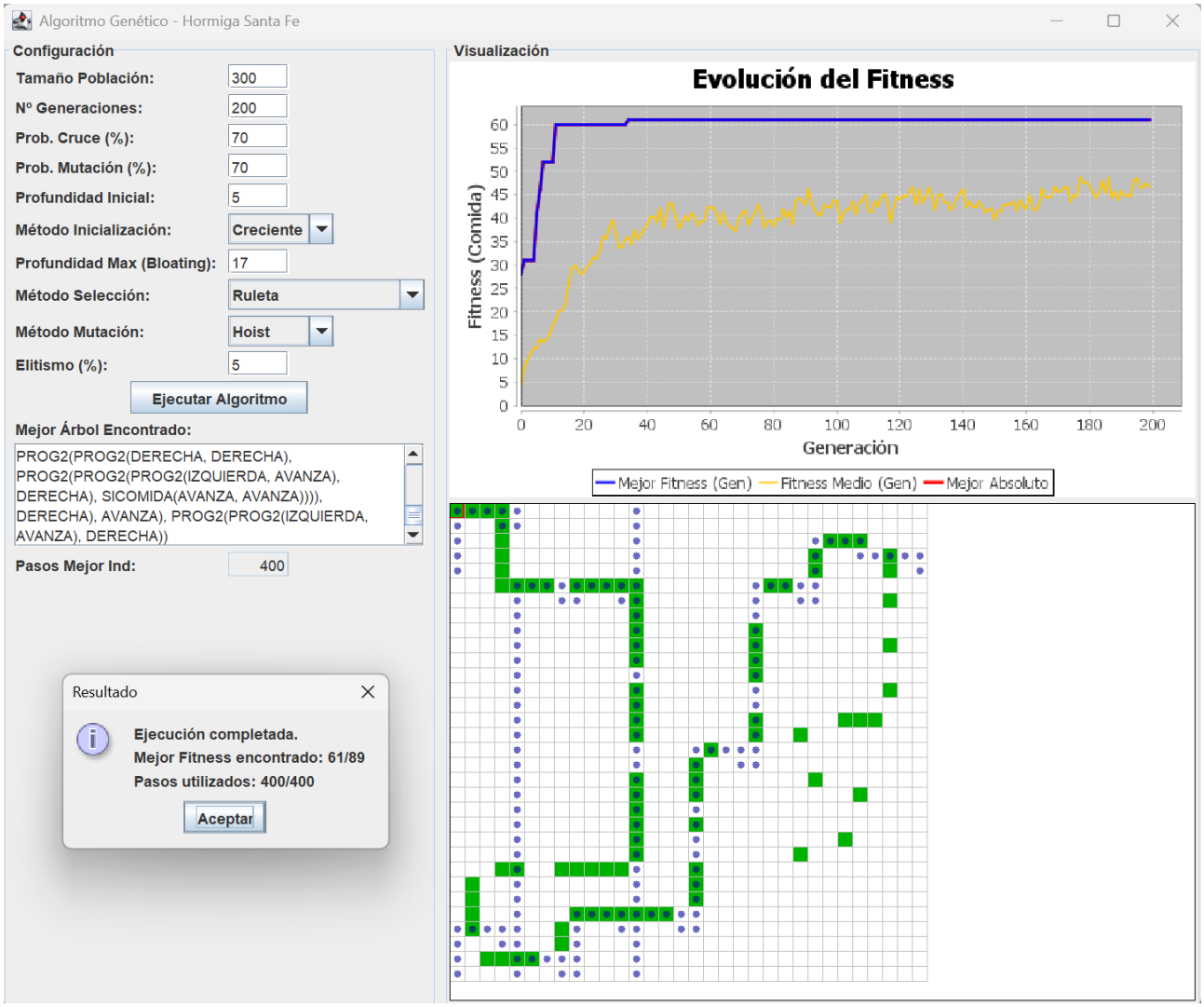
Ejecución #1


Algoritmo Genético - Hormiga Santa Fe
Configuración
Tamaño Población: 300
Nº Generaciones: 200
Prob. Cruce (%): 70
Prob. Mutación (%): 70
Profundidad Inicial: 5
Método Inicialización: Ramped
Profundidad Max (Bloating): 17
Método Selección: Torneo Determinístico
Método Mutación: Funcional
Elitismo (%): 5
Ejecutar Algoritmo
Mejor Árbol Encontrado:
IZQUIERDA), PROG3(AVANZA, DERECHA, PROG3(AVANZA, PROG2(PROG2(AVANZA, SICOMIDA(AVANZA, IZQUIERDA)), AVANZA), IZQUIERDA))), AVANZA), PROG3(AVANZA, AVANZA, PROG2(AVANZA, DERECHA))))
Pasos Mejor Ind: 400
Resultado
Ejecución completada.
Mejor Fitness encontrado: 64/89
Pasos utilizados: 400/400
Aceptar
Visualización
Evolución del Fitness
Fitness (Comida)
Generación
Mejor Fitness (Gen)
Fitness Medio (Gen)
Mejor Absoluto

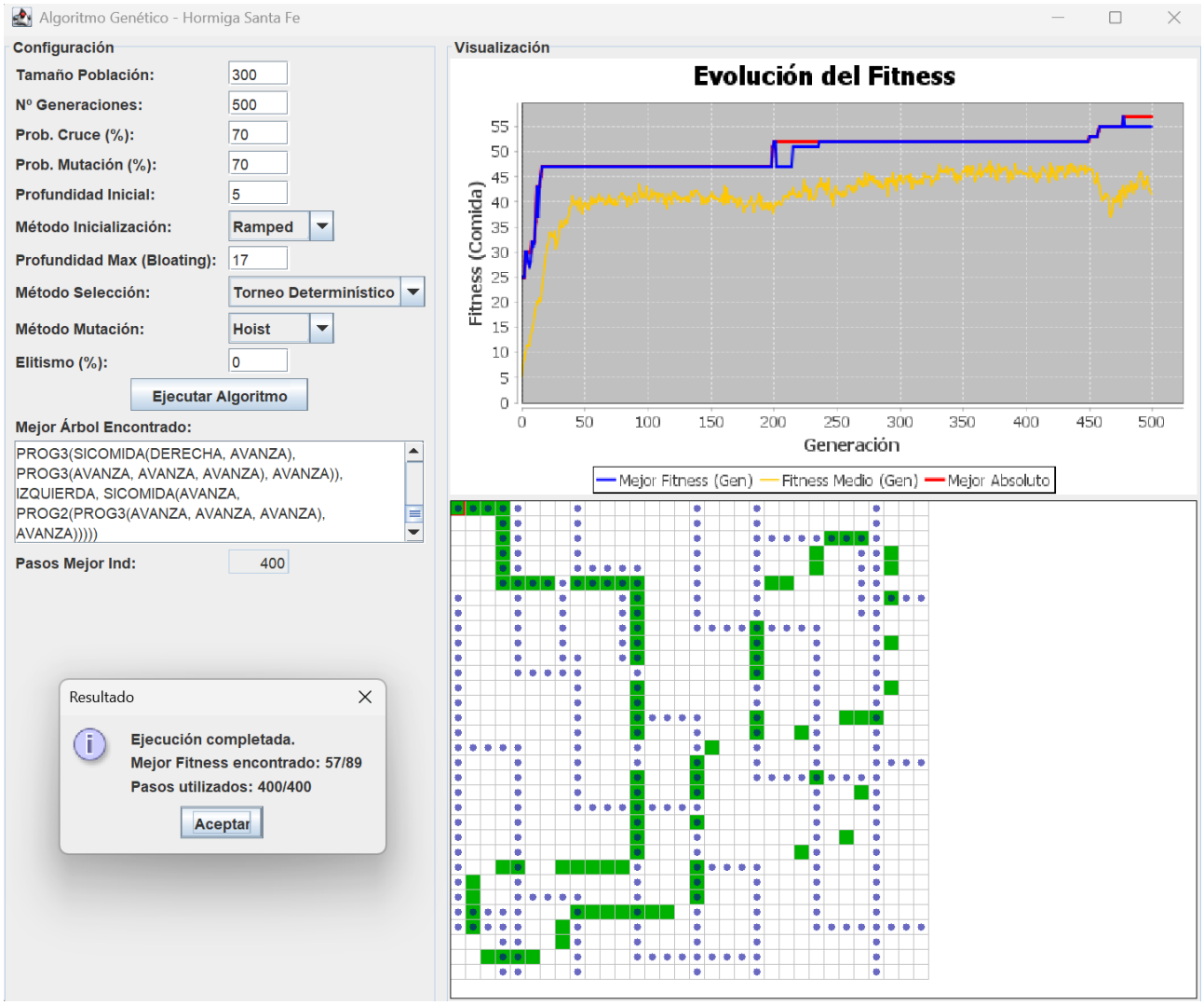
Ejecución #2


Algoritmo Genético - Hormiga Santa Fe
Configuración
Tamaño Población: 300
Nº Generaciones: 200
Prob. Cruce (%): 70
Prob. Mutación (%): 70
Profundidad Inicial: 5
Método Inicialización: Creciente
Profundidad Max (Bloating): 17
Método Selección: Torneo Deterministico
Método Mutación: Hoist
Elitismo (%): 5
Ejecutar Algoritmo
Mejor Árbol Encontrado:
AVANZA), SICOMIDA(IZQUIERDA, DERECHA))))), DERECHA), AVANZA), AVANZA), PROG3(IZQUIERDA, SICOMIDA(SICOMIDA(DERECHA, AVANZA), AVANZA), DERECHA))))))), SICOMIDA(SICOMIDA(DERECHA, DERECHA), PROG3(IZQUIERDA, AVANZA, DERECHA)))
Pasos Mejor Ind: 400
Resultado
Ejecución completada.
Mejor Fitness encontrado: 61/89
Pasos utilizados: 400/400
Aceptar
Visualización
Evolución del Fitness
Fitness (Comida)
Generación
Mejor Fitness (Gen)
Fitness Medio (Gen)
Mejor Absoluto

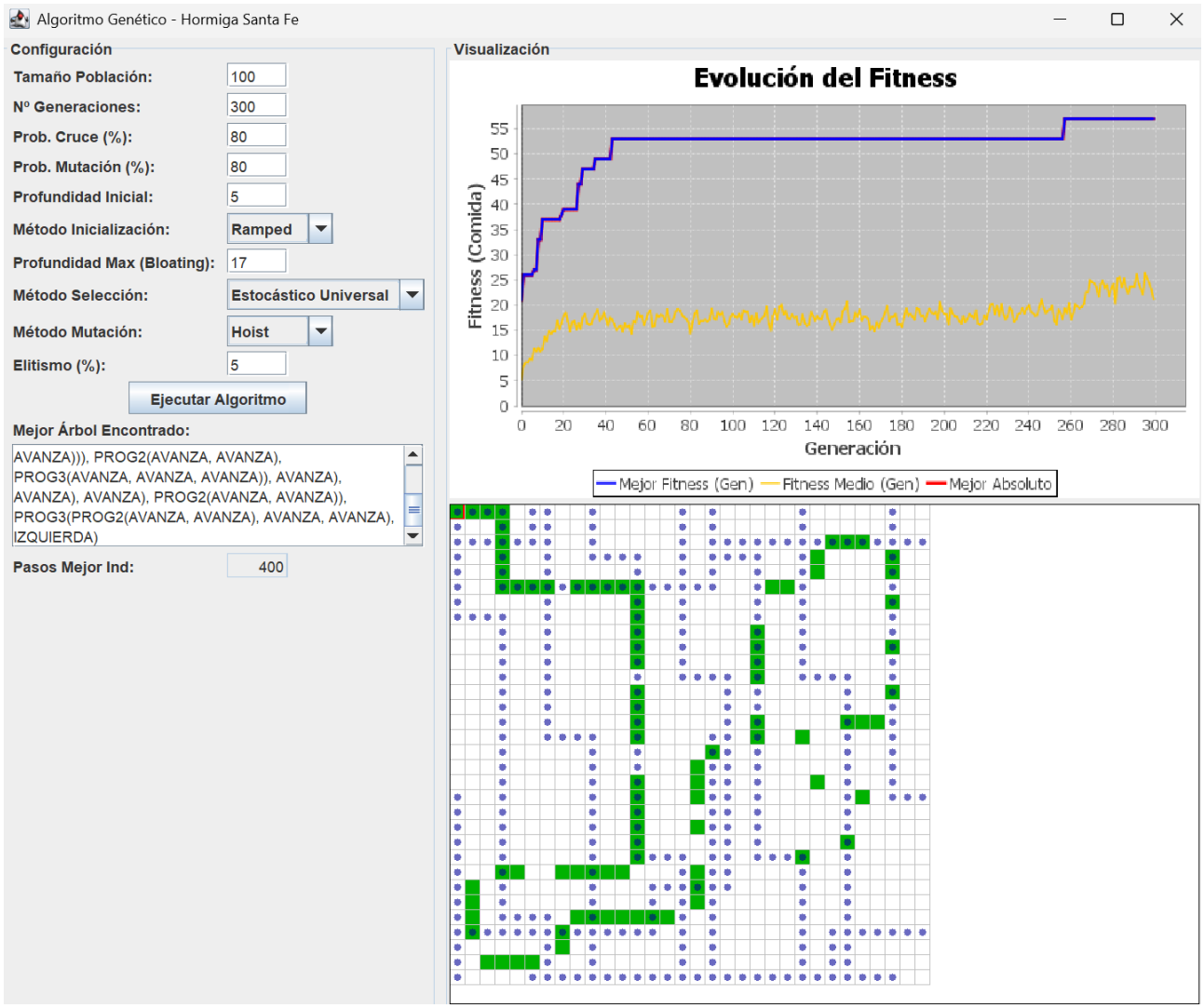
Ejecución #3


Algoritmo Genético - Hormiga Santa Fe
Configuración
Tamaño Población: 300
Nº Generaciones: 200
Prob. Cruce (%): 70
Prob. Mutación (%): 70
Profundidad Inicial: 5
Método Inicialización: Creciente
Profundidad Max (Bloating): 17
Método Selección: Ruleta
Método Mutación: Hoist
Elitismo (%): 5
Ejecutar Algoritmo
Mejor Árbol Encontrado:
PROG2(PROG2(DERECHA, DERECHA), PROG2(PROG2(PROG2(IZQUIERDA, AVANZA), DERECHA), SICOMIDA(AVANZA, AVANZA))), DERECHA), AVANZA), PROG2(PROG2(IZQUIERDA, AVANZA), DERECHA))
Pasos Mejor Ind: 400
Resultado
Ejecución completada.
Mejor Fitness encontrado: 61/89
Pasos utilizados: 400/400
Aceptar
Visualización
Evolución del Fitness
Fitness (Comida)
Generación
Mejor Fitness (Gen)
Fitness Medio (Gen)
Mejor Absoluto

Ejecución #4


Algoritmo Genético - Hormiga Santa Fe
Configuración
Tamaño Población: 300
Nº Generaciones: 500
Prob. Cruce (%): 70
Prob. Mutación (%): 70
Profundidad Inicial: 5
Método Inicialización: Ramped
Profundidad Max (Bloating): 17
Método Selección: Torneo Determinístico
Método Mutación: Hoist
Elitismo (%): 0
Ejecutar Algoritmo
Mejor Árbol Encontrado:
PROG3(SICOMIDA(DERECHA, AVANZA), PROG3(AVANZA, AVANZA, AVANZA), AVANZA)), IZQUIERDA, SICOMIDA(AVANZA, PROG2(PROG3(AVANZA, AVANZA, AVANZA), AVANZA)))))
Pasos Mejor Ind: 400
Resultado
Ejecución completada.
Mejor Fitness encontrado: 57/89
Pasos utilizados: 400/400
Aceptar
Visualización
Evolución del Fitness
Fitness (Comida)
Generación
Mejor Fitness (Gen)
Fitness Medio (Gen)
Mejor Absoluto

Ejecución #5



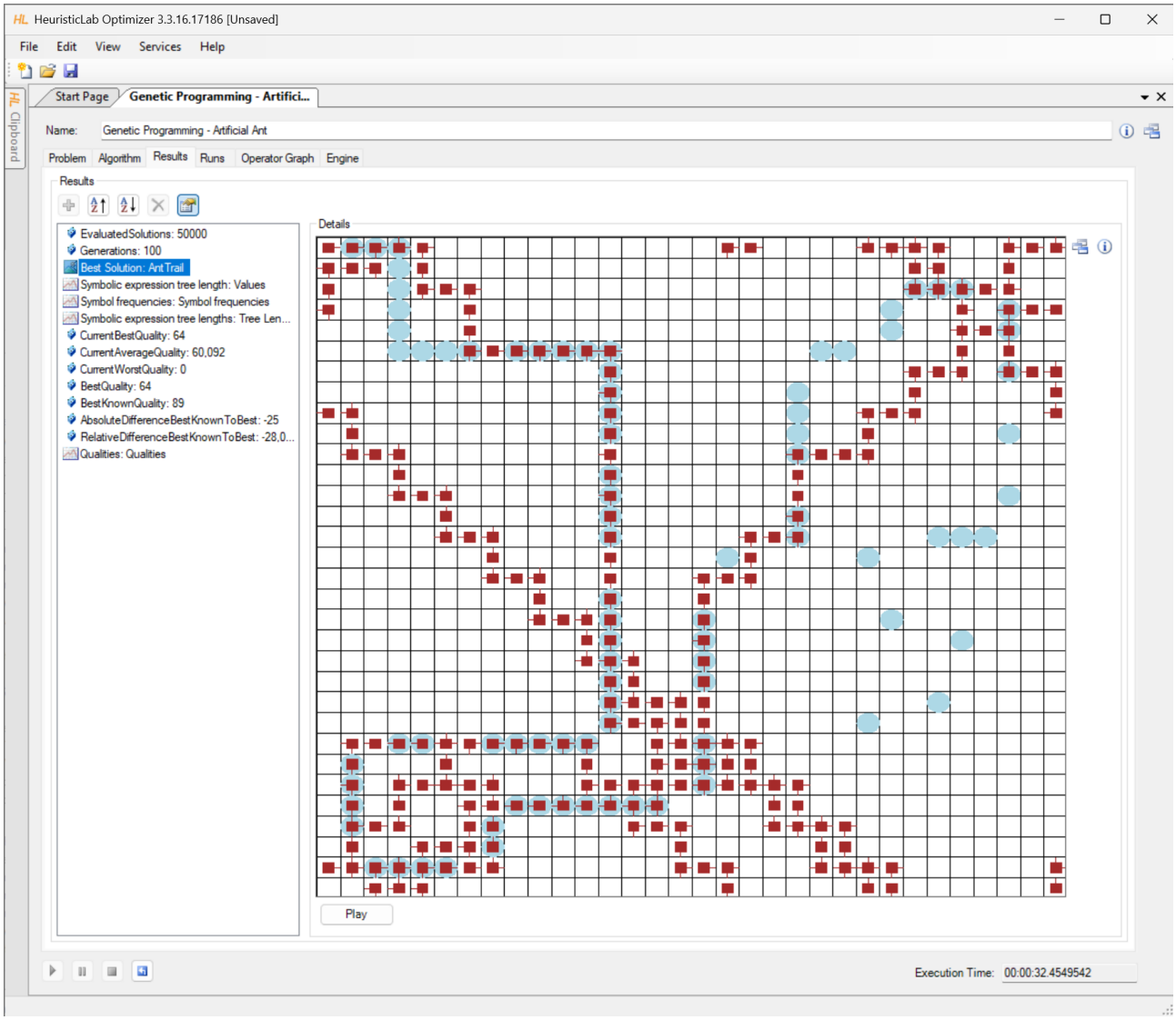
4. Resultados Experimentales (HeuristicLab)

Se utilizó el entorno HeuristicLab para modelar y resolver el mismo problema de la Hormiga (Rastro de Santa Fe) mediante Programación Genética.

4.1. Configuración en HeuristicLab

El problema se configuró utilizando los operadores y parámetros equivalentes a los de la aplicación propia, en la medida de lo posible.

4.2. Resultados Obtenidos (Problema Hormiga en HeuristicLab)



Sigue siendo un máximo de 64 de fitness, por lo que el problema para llegar a la solución optima seguramente sean unos parámetros incorrectos. Después de la entrega (con más tiempo) encontrare los parámetros adecuados.

5. Conclusiones

No se ha logrado alcanzar la solución óptima (89/89) tampoco en esta versión revisada, y me temo que es por la parte opcional de gramáticas evolutivas, que permitiría métodos de cruce y mutación más flexibles.

No obstante, se realizaron optimizaciones críticas sobre la versión inicial del algoritmo para corregir deficiencias en su lógica, rendimiento y control evolutivo.

1. Corrección del Intérprete de Programas:

- **Problema:** La lógica original invalidaba el funcionamiento del operador SICOMIDA, ya que las decisiones condicionales no se ejecutaban en tiempo real.

- **Solución:** Se implementó un **intérprete dinámico y recursivo** que evalúa las condiciones a cada paso, permitiendo un comportamiento reactivo y mejorando drásticamente el potencial de las soluciones.

2. Control de Bloating **mediante Presión de Parquedad:**

- **Problema:** Los programas crecían en tamaño sin control, generando soluciones ineficientes.
- **Solución:** Se introdujo una penalización por tamaño durante la selección (Presión de Parquedad). Ahora, a igual rendimiento, el algoritmo favorece a los individuos con los programas más cortos y elegantes.

3. Optimización del Rendimiento (Caché de Fitness):

- **Problema:** El algoritmo era muy lento al re-calcular el costoso fitness de cada individuo múltiples veces por generación.
- **Solución:** Se implementó una **caché** que almacena el fitness de cada individuo una vez por generación. Esto aceleró la ejecución de forma significativa, permitiendo realizar ejecuciones con poblaciones más grandes.

4. Mejoras en la Interfaz Gráfica:

- **Problema:** La gráfica del "Mejor Absoluto" mostraba datos incorrectos y la presentación del resultado final era poco práctica.
- **Solución:** Se corrigió el cálculo para una visualización correcta de la gráfica y se reemplazó la ventana emergente por un **panel de resultados fijo** en la interfaz principal, mejorando la usabilidad.