PRÁCTICA 3

PROGRAMACIÓN EVOLUTIVA

Grupo 2

Javier Martín-Tesorero Ruiz

Clara Martín Navas

Introducción

En esta práctica se aborda la implementación de algoritmos evolutivos para resolver problemas complejos mediante dos enfoques principales:

1. **Programación genética:** enfocada en la evolución de programas y estructuras funcionales.
2. **Gramáticas evolutivas:** que combinan la flexibilidad de los algoritmos evolutivos con la expresividad de las gramáticas formales.

El objetivo principal es desarrollar soluciones optimizadas para problemas específicos, como el caso de la hormiga artificial, evaluando diferentes operadores genéticos y configuraciones para determinar las estrategias más efectivas.

1. Representación de la solución

1.1 Estructura del cromosoma

Nuestra implementación utiliza una representación basada en árboles, donde los cromosomas son programas ejecutables estructurados jerárquicamente. La estructura básica se compone de:

- **Interfaz Node:** Define el contrato básico para los nodos del árbol con métodos como getValor(), getHijos() y clone().
- **TerminalNode:** Implementa la interfaz Node y representa acciones básicas (terminales) que puede realizar la hormiga:
    - **"AVANZA":** Mueve la hormiga una celda en la dirección actual.
    - **"DERECHA":** Gira la hormiga 90° a la derecha.
    - **"IZQUIERDA":** Gira la hormiga 90° a la izquierda.
- **FunctionNode:** Implementa la interfaz Node y representa nodos no terminales con aridad específica:
    - **"SICOMIDA":** Función condicional con aridad 2 (si hay comida adelante, ejecuta el primer hijo; si no, el segundo).
    - **"PROG2":** Secuencia de 2 instrucciones (ejecuta los dos hijos en orden).
    - **"PROG3":** Secuencia de 3 instrucciones (ejecuta los tres hijos en orden).
- **Estado:** Clase que encapsula el estado actual de la hormiga, incluyendo:

- **Posición** (x, y)
- **Dirección** (NORTE, SUR, ESTE, OESTE)
- **Métodos para manipular y consultar este estado**

Esta estructura jerárquica permite representar programas complejos que combinan decisiones (SICOMIDA) y secuencias de acciones (PROG2, PROG3) con las operaciones básicas de movimiento (AVANZA, DERECHA, IZQUIERDA).

1.2 Clase Árbol

La clase Árbol encapsula la estructura completa del cromosoma y proporciona métodos para:

- Manipular la estructura del árbol (añadir/eliminar/reemplazar nodos)
- Recorrer el árbol para recolectar diferentes tipos de nodos
- Calcular propiedades como profundidad y tamaño
- Generar subárboles aleatorios usando métodos completos o crecientes
- Ejecutar el programa representado por el árbol

Esta representación ofrece la flexibilidad necesaria para evolucionar programas complejos mientras mantiene una estructura que permite aplicar operadores genéticos de forma eficiente.

1.3 Gramáticas Evolutivas

En el enfoque de gramáticas evolutivas, complementamos la representación en árbol con una gramática formal que define la sintaxis válida de los programas. La gramática se define como: **G = {N, T, P, S}**

Donde:

- N = {<expr>, <acción>, <condición>, <prog>} (Símbolos no terminales)
- T = {AVANZA, DERECHA, IZQUIERDA, SICOMIDA, PROG2, PROG3} (Símbolos terminales, correspondientes a los nodos de nuestra implementación)
- S = <expr> (Símbolo inicial)
- P = Conjunto de reglas de producción:
  - <expr> ::= <acción> | <condición> | <prog>
  - <acción> ::= AVANZA | DERECHA | IZQUIERDA
  - <condición> ::= SICOMIDA(<expr>, <expr>)
  - <prog> ::= PROG2(<expr>, <expr>) | PROG3(<expr>, <expr>, <expr>)

Los cromosomas en este enfoque son cadenas de enteros que se mapean a programas válidos según esta gramática, garantizando así la corrección sintáctica de todas las soluciones generadas.

2. Inicialización de la Población

La inicialización de la población es una fase crítica en algoritmos evolutivos, ya que determina el punto de partida para la exploración del espacio de búsqueda. Una inicialización eficaz debe proporcionar suficiente diversidad genética para permitir una amplia exploración mientras mantiene individuos de calidad razonable. En nuestra implementación para el problema de la hormiga artificial, hemos implementado tres métodos de inicialización principales para los árboles que representan los programas de control.

2.1 Método completo

El método completo (full) genera árboles perfectamente balanceados donde todos los nodos internos tienen el número máximo de hijos hasta alcanzar la profundidad máxima predefinida.

Algoritmo:

1. Se establece una profundidad máxima *D*
2. Se crea un nodo raíz aleatorio de tipo función
3. Para cada nivel desde la raíz hasta *D-1*:
    - Se añaden únicamente nodos función como hijos
4. Al llegar al nivel *D* (hojas):
    - Se añaden únicamente nodos terminales

Características:

- Todos los nodos en niveles inferiores a la profundidad máxima son nodos función
- Todos los nodos en el nivel de profundidad máxima son terminales
- Todos los caminos desde la raíz hasta cualquier hoja tienen la misma longitud (igual a *D*)
- La estructura resultante es similar a un árbol binario completo (o *n*-ario completo según la aridad de las funciones)

2.2 Método creciente

El método creciente (grow) permite la creación de árboles de forma irregular seleccionando aleatoriamente entre nodos función y terminales hasta alcanzar la profundidad máxima.

Algoritmo:

1. Se establece una profundidad máxima *D*
2. Para cada nodo en profundidad menor que *D*:
    - Se elige aleatoriamente entre crear un nodo función o terminal
3. Al llegar al nivel *D* (profundidad máxima):
    - Se crean únicamente nodos terminales

Características:

- Los árboles tienen formas irregulares y variadas
- Las ramas pueden tener diferentes longitudes
- Mayor diversidad estructural en la población inicial
- Mayor probabilidad de generar árboles pequeños y simples

2.3 Método ramped-and-half

El método ramped-and-half combina los enfoques completo y creciente para maximizar la diversidad estructural de la población inicial.

Algoritmo:

1. Se divide la población en *R* grupos (donde *R* es la profundidad máxima menos la mínima)
2. Para cada grupo *i*:
    - Se asigna una profundidad máxima *i + profMin*
    - La mitad de los individuos se generan con el método completo
    - La otra mitad se genera con el método creciente

Características:

- Máxima diversidad estructural en la población inicial
- Incluye árboles de diferentes tamaños y formas
- Equilibrio entre soluciones simples y complejas
- Mejor cobertura del espacio de búsqueda inicial

3. Operadores Genéticos

3.1 Selección

Se implementaron diversos métodos de selección para evaluar su impacto en la convergencia y diversidad poblacional:

- **Ruleta:** Asigna probabilidades proporcionales al fitness de cada individuo.
- **Torneos determinísticos:** Selecciona el mejor individuo de un grupo aleatorio.
- **Torneos probabilísticos:** Similar al anterior, pero con probabilidad de seleccionar individuos con menor fitness.
- **Selección estocástica universal:** Muestreo sistemático para mantener diversidad.
- **Truncamiento:** Selecciona directamente un porcentaje de los mejores individuos.
- **Restos:** Combina selección determinística y probabilística.
- **Ranking:** Asigna probabilidades basadas en la posición ordenada por fitness.

Mejor resultado: Los experimentos mostraron que la selección por torneo con tamaño 5-7 proporcionó el mejor equilibrio entre presión selectiva y diversidad poblacional. Este método favorece a los individuos con mayor fitness sin eliminar completamente la diversidad genética, lo que es crucial para evitar la convergencia prematura.

3.2 Cruce

En nuestra implementación, el principal operador de cruce es el intercambio de subárboles:

Cruce de subárboles: Intercambia subárboles completos entre dos individuos progenitores. El procedimiento es:

1. Seleccionar un nodo aleatorio en cada padre
2. Intercambiar los subárboles que tienen como raíz esos nodos
3. Verificar que los nuevos árboles no excedan la profundidad máxima permitida

Para gramáticas evolutivas, se implementaron dos operadores adicionales:

Cruce monopunto: Intercambia segmentos de la cadena de codones

1. Selecciona un punto de corte aleatorio en las cadenas de enteros de ambos padres

2. Intercambia los segmentos posteriores al punto de corte
3. Verifica y repara la validez gramatical si es necesario

Cruce uniforme: Intercambia codones individuales con cierta probabilidad:

1. Para cada posición en las cadenas de enteros, decide aleatoriamente de qué padre tomar el valor
2. Construye un nuevo individuo combinando estos valores
3. Verifica y ajusta la gramática para garantizar que el resultado sea válido

Conclusión: El cruce de subárboles demostró ser el más efectivo para programación genética tradicional, mientras que, para gramáticas evolutivas, el cruce monopunto ofreció un mejor equilibrio entre exploración y explotación del espacio de búsqueda.

3.3 Mutación

Basándonos en el código proporcionado, se implementaron cuatro operadores de mutación principales:

ArbolSubarbol: Reemplaza un subárbol completo por uno generado aleatoriamente.

Funcional: Cambia un nodo función por otro nodo función de la misma aridad.

Terminal: Cambia un nodo terminal por otro nodo terminal.

Permutación:

Para gramáticas evolutivas, se implementan también:

Mutación Gramática: Modifica un codón específico, por otro valor valido (0 – 255).

Conclusión: La combinación de estos operadores de mutación, aplicados con diferentes probabilidades, proporciona un equilibrio entre exploración global (ArbolSubarbol) y refinamiento local (Funcional y Terminal). La mutación de subárbol permite grandes saltos en el espacio de búsqueda, mientras que las mutaciones funcional y terminal realizan ajustes más sutiles, permitiendo un refinamiento gradual de las soluciones.

4. Fitness

La función de fitness para el problema de la hormiga artificial se basa en la cantidad de comida recolectada en un número limitado de pasos. El entorno consiste en una cuadrícula con piezas de comida distribuidas siguiendo un "rastro de Santa Fe".

Formalmente: Fitness = Número de piezas de comida recolectadas

Con las siguientes consideraciones:

- La hormiga tiene un límite de pasos (configurable entre 400 y 800)
- El fitness máximo teórico es 89 (todas las piezas de comida)

- Se implementa un sistema de caché para evitar recalcular el fitness de individuos no modificados (mediante el método disableCache() que invalida la caché cuando el individuo sufre cambios)

5. Bloating

6. Gramáticas Evolutivas

Las gramáticas evolutivas extienden nuestro enfoque básico de programación genética incorporando restricciones sintácticas. La implementación incluye:

- **Definición gramatical**: Especificación formal de las reglas de producción que definen programas válidos.
- **Genotipo:** Representación como cadena de enteros que codifica las elecciones de reglas de producción.
- **Proceso de mapeo:** Algoritmo que traduce el genotipo a un fenotipo (programa ejecutable).
- **Operadores específicos:** Adaptación de los operadores genéticos para mantener la validez gramatical.

Este enfoque ofrece varias ventajas:

1. Garantiza que todas las soluciones sean sintácticamente correctas
2. Reduce el espacio de búsqueda a soluciones válidas
3. Permite incorporar conocimiento del dominio en la definición de la gramática
4. Facilita la evolución de soluciones con estructuras complejas

7. Resultados y Gráficas

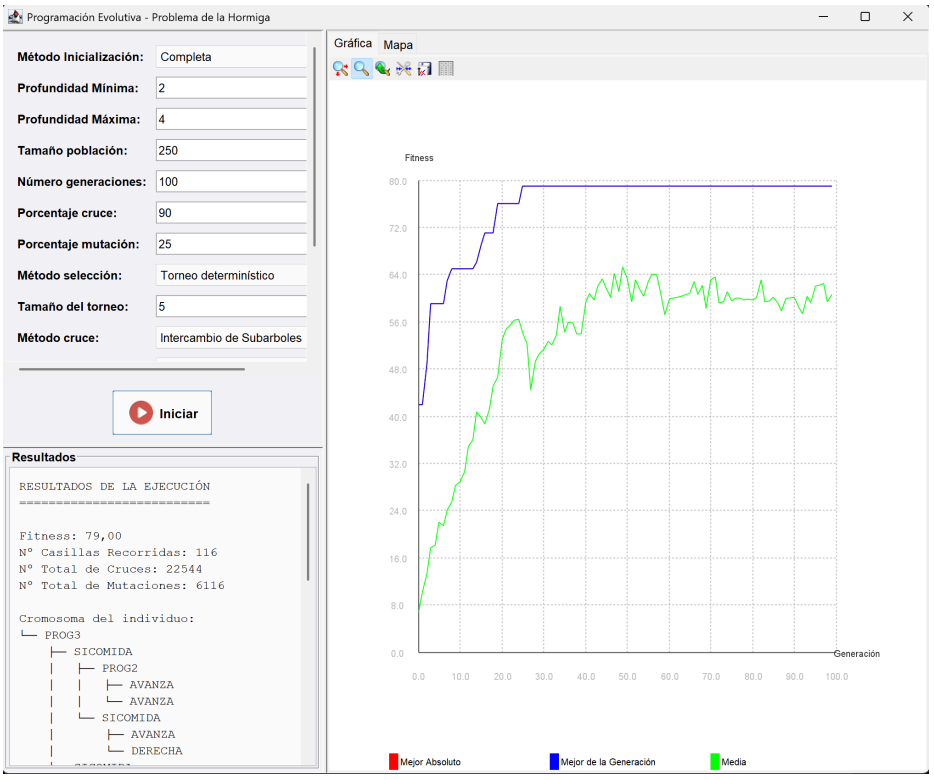
Se realizaron múltiples experimentos con diferentes configuraciones para evaluar el rendimiento de los diversos operadores y enfoques. A continuación, se presentan los modelos más relevantes:

Modelo 1: Configuración Básica con Mutación de Terminales

- Parámetros:
    - Tamaño de población: 250
    - Generaciones máximas: 100
    - Pasos máximos: 400
    - Cruce: Subárboles (probabilidad 90%)
    - Mutación: Terminal (probabilidad 25%)
    - Selección: Torneo (tamaño 5)

- Elitismo: 2%

Resultados:



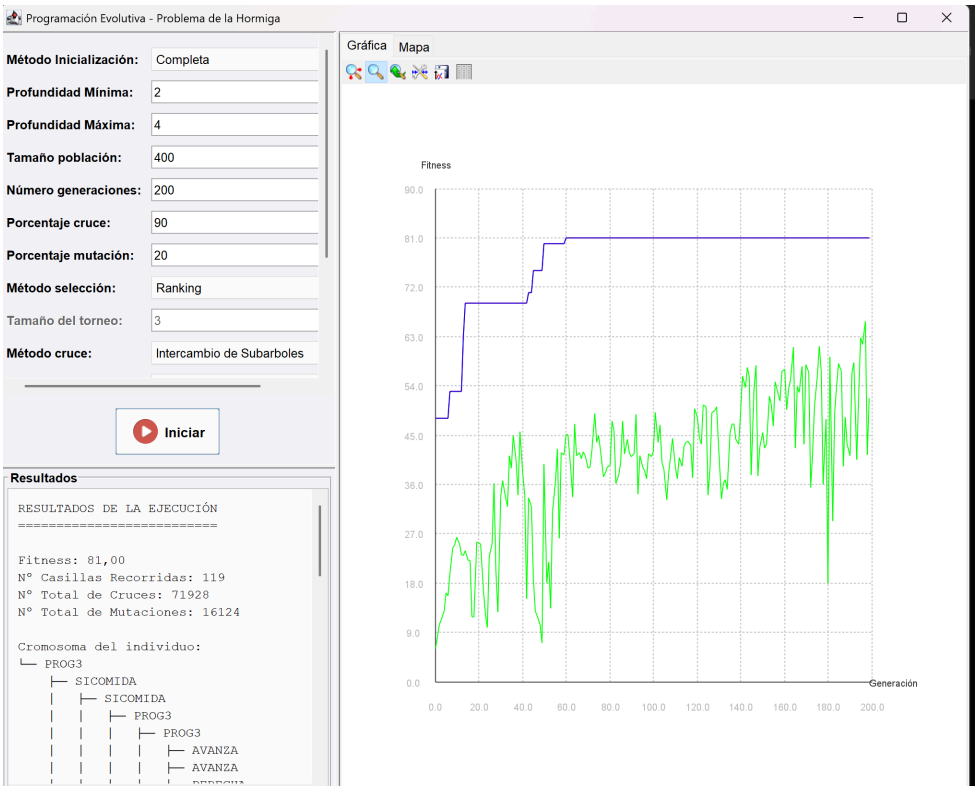
Resultados: Esta configuración mostró una convergencia moderada pero estable. La mutación de terminales permitió un refinamiento gradual de las soluciones sin alterar drásticamente su estructura, lo que resultó en una exploración controlada del espacio de búsqueda.

Modelo 2: Configuración Exploratoria con Mutación de Subárboles

- Parámetros:
  - Tamaño de población: 400
  - Generaciones máximas: 200
  - Pasos máximos: 600
  - Cruce: Subárboles (probabilidad 90%)
  - Mutación: ArbolSubarbol (probabilidad 20%)
  - Selección: Ranking
  - Elitismo: 5%

Resultados: Esta configuración, con mayor componente exploratorio debido a la mutación de subárboles, mostró mejores resultados a largo plazo pero con mayor

variabilidad inicial. La selección por ranking ayudó a mantener la diversidad poblacional, permitiendo que soluciones prometedoras pero subóptimas sobrevivieran y contribuyeran a la evolución



Modelo 3: Exploración Grande con Amplia Presión Selectiva

- Parámetros:
    - Tamaño de población: 800
    - Generaciones máximas: 100
    - Pasos máximos: 400
    - Cruce: Monopunto (probabilidad 90%)
    - Mutación: ArbolSubarbol (probabilidad 25%)
    - Selección: Torneo (tamaño 7)
    - Elitismo: 2 individuos

Resultados: Esta configuración con alta presión selectiva y una población grande logró explorar efectivamente el espacio de soluciones, encontrando rápidamente individuos de alta calidad. Sin embargo, la convergencia prematura fue un riesgo debido a la alta presión selectiva.

Modelo 4: Gramáticas Evolutivas con Cruce Monopunto

- Parámetros:
    - Tamaño de población: 350
    - Generaciones máximas: 180
    - Pasos máximos: 550
    - Cruce: Monopunto (probabilidad 90%)
    - Mutación: Puntual (probabilidad 15%)
    - Selección: Torneo (tamaño 4)
    - Elitismo: 2 individuos

Resultados: Este modelo, basado en gramáticas evolutivas, mostró un rendimiento competitivo con la ventaja adicional de garantizar la validez sintáctica de todas las soluciones. La representación gramatical y el cruce monopunto permitieron una exploración eficiente del espacio de soluciones válidas.

Modelo 5: Gramáticas Evolutivas con Cruce Uniforme

- Parámetros:
    - Tamaño de población: 350
    - Generaciones máximas: 180
    - Pasos máximos: 550
    - Cruce: Uniforme (probabilidad 90%)
    - Mutación: Puntual (probabilidad 15%)
    - Selección: Torneo (tamaño 4)
    - Elitismo: 2 individuos

Resultados: Este modelo, similar al anterior, pero utilizando cruce uniforme, mostró un rendimiento ligeramente inferior. El cruce uniforme tendió a generar más disrupción en las soluciones, lo que dificultó la preservación de bloques constructivos efectivos.

8. HeuristicLab

Realizamos también experimentos utilizando la plataforma HeuristicLab para comparar con nuestra implementación propia. A continuación, se presentan las configuraciones más relevantes.

- Population Size: 1000
- Maximum Generations:
- SetSeedRandomly: True
- Analyzer: MultiAnalyzer
- Elites: 1
- Pasos máximos de la hormiga: 600
- Crossover: SubtreeSwappingCrossover
- Crossover Probability: 90%

Modelo 1: Configuración Clásica Equilibrada

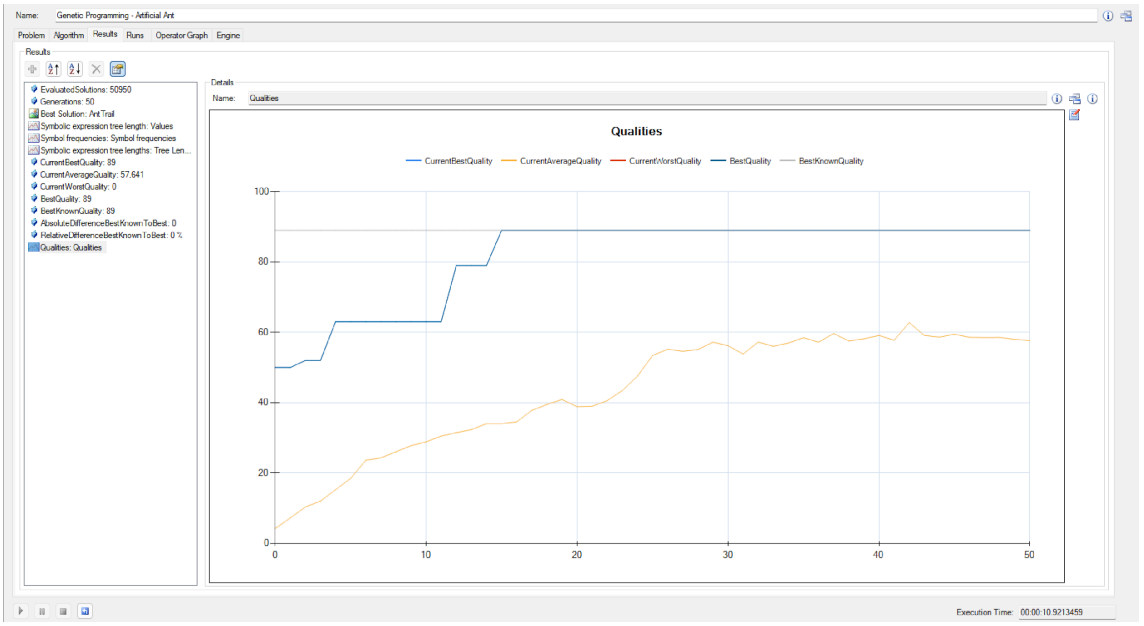
Mutación:

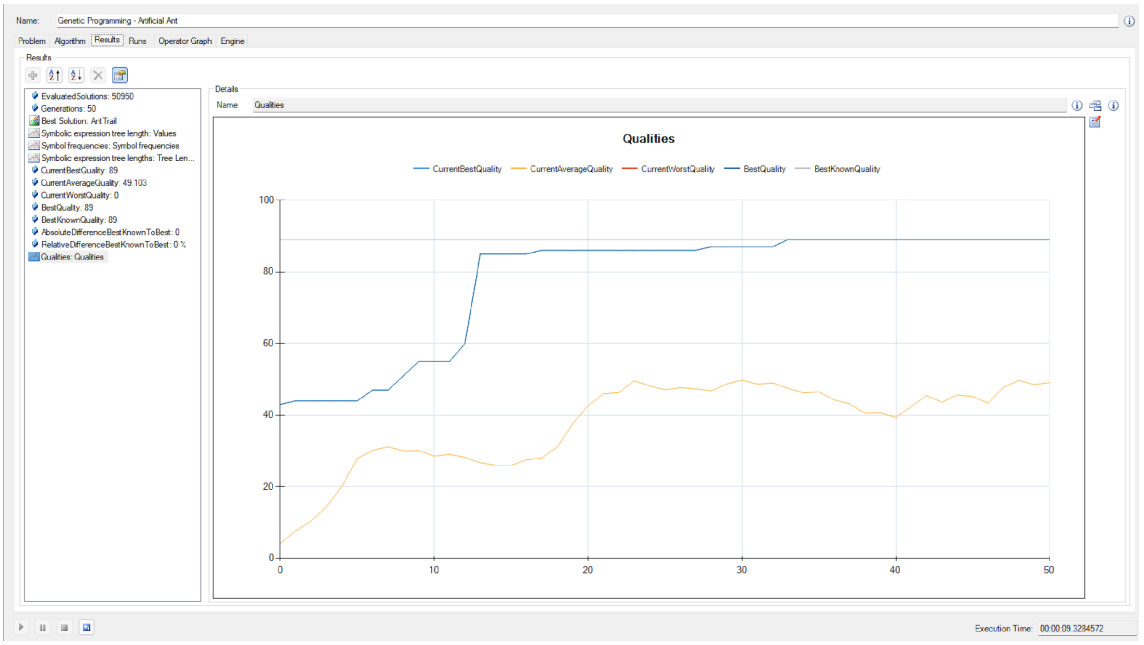
- ChangeNodeTypeManipulation (40% de probabilidad)

Selección:

- TournamentSelector con tamaño 5

Justificación: Esta configuración representa el enfoque clásico en programación genética. La mutación de reemplazo de ramas permite una exploración considerable del espacio de búsqueda mientras que la mutación de cambio de tipo de nodo realiza ajustes más sutiles. El selector de torneo con tamaño intermedio proporciona una presión selectiva moderada, balanceando exploración y explotación.





Modelo 2: Configuración Exploratoria con Simplificación

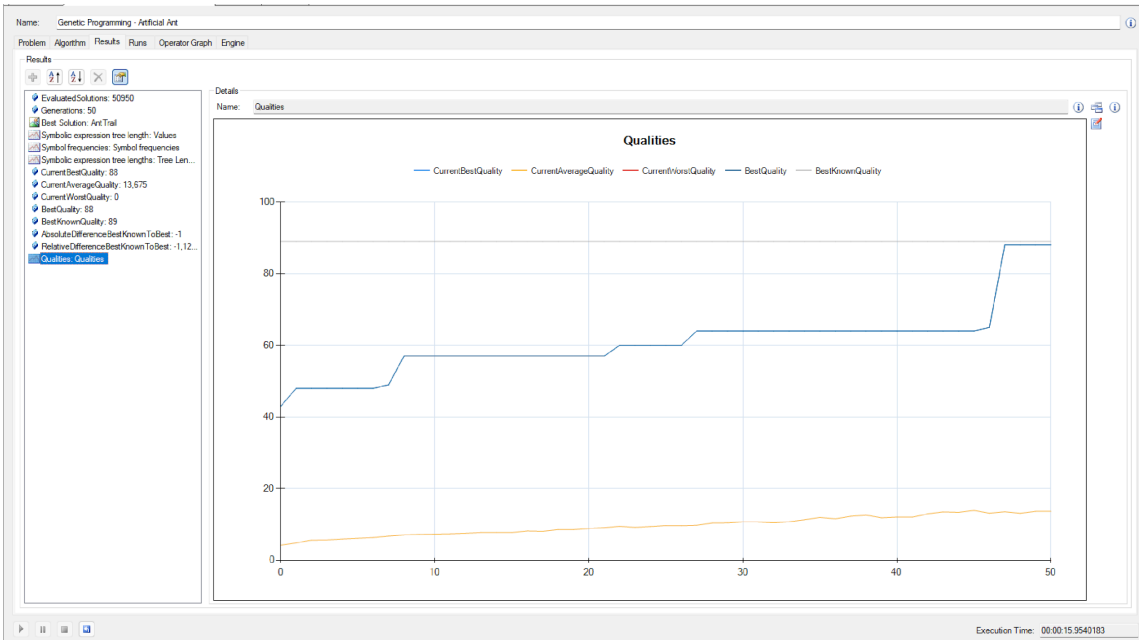
Mutación:

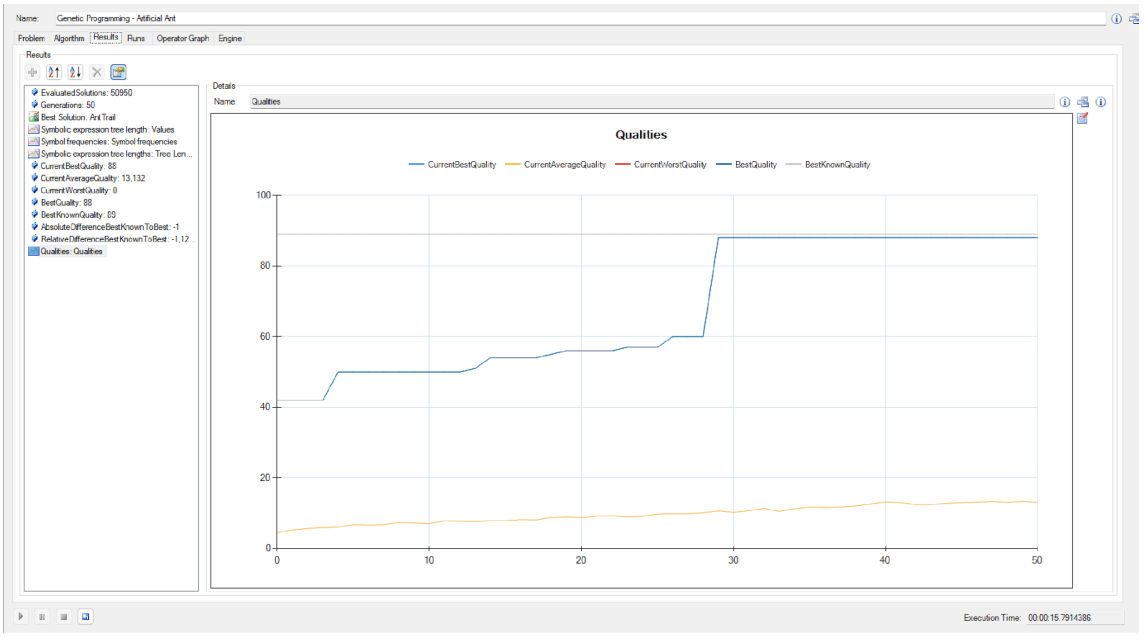
- RemoveBranchManipulation (30% de probabilidad)

Selección:

- LinearRankSelector (presión selectiva media)

Justificación: Esta configuración favorece la exploración y la simplificación de programas. El selector por rango lineal mantiene diversidad en la población al basarse en las posiciones relativas y no en los valores absolutos de fitness.





Modelo 3: Configuración con Subrutinas (ADFs)

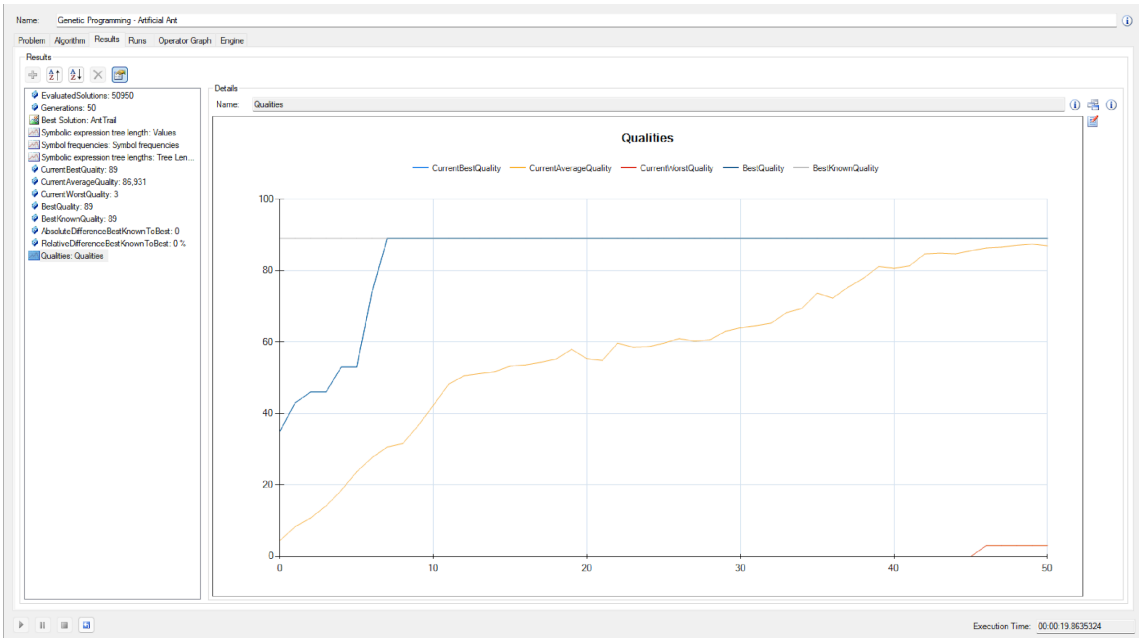
Mutación:

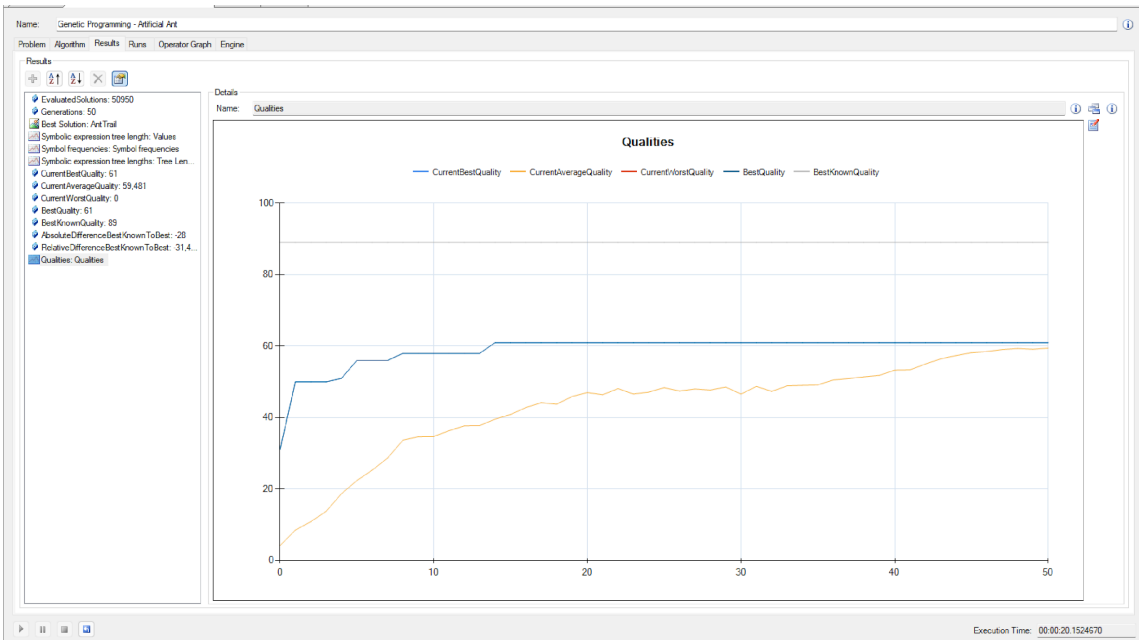
- ReplaceBranchManipulation (40% de probabilidad)
- SubroutineCreator (20% de probabilidad)
- SubroutineDuplicater (20% de probabilidad)
- ChangeNodeTypeManipulation (20% de probabilidad)

Selección:

- TournamentSelector con tamaño 7

Justificación: Esta configuración explora el potencial de las Funciones Automáticamente Definidas (ADFs) para resolver el problema de la hormiga artificial. Al permitir la creación y duplicación de subrutinas, facilitamos la reutilización de patrones de movimiento efectivos. El selector de torneo con un tamaño mayor proporciona una fuerte presión selectiva para favorecer a los mejores individuos, compensando la mayor complejidad de búsqueda introducida por las subrutinas.





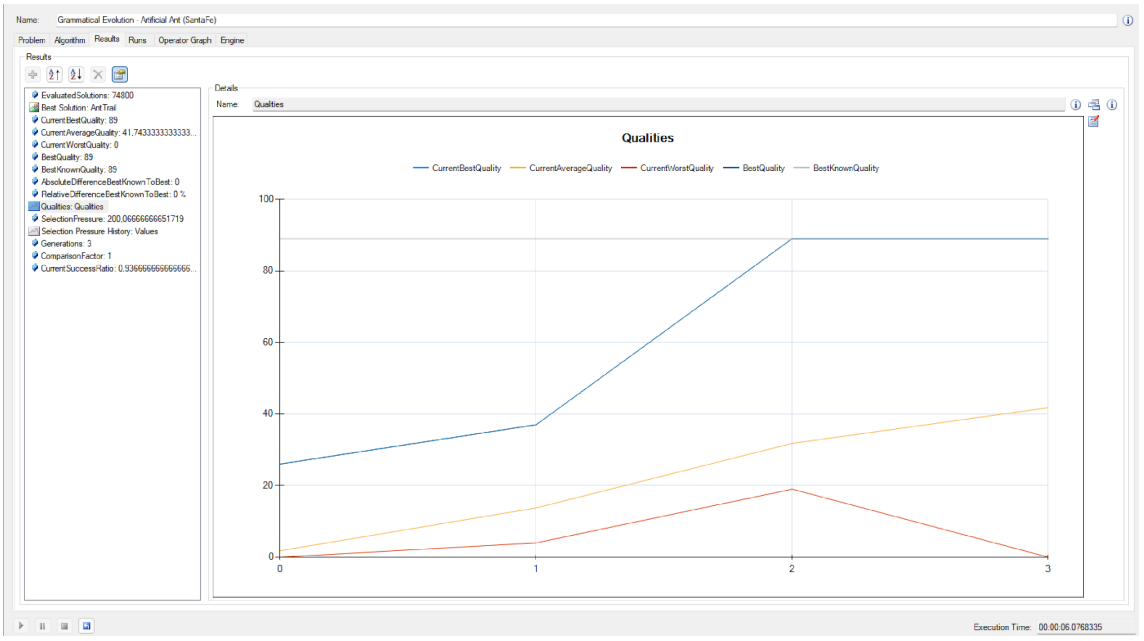
Modelo 4: Gramáticas con exploración equilibrada

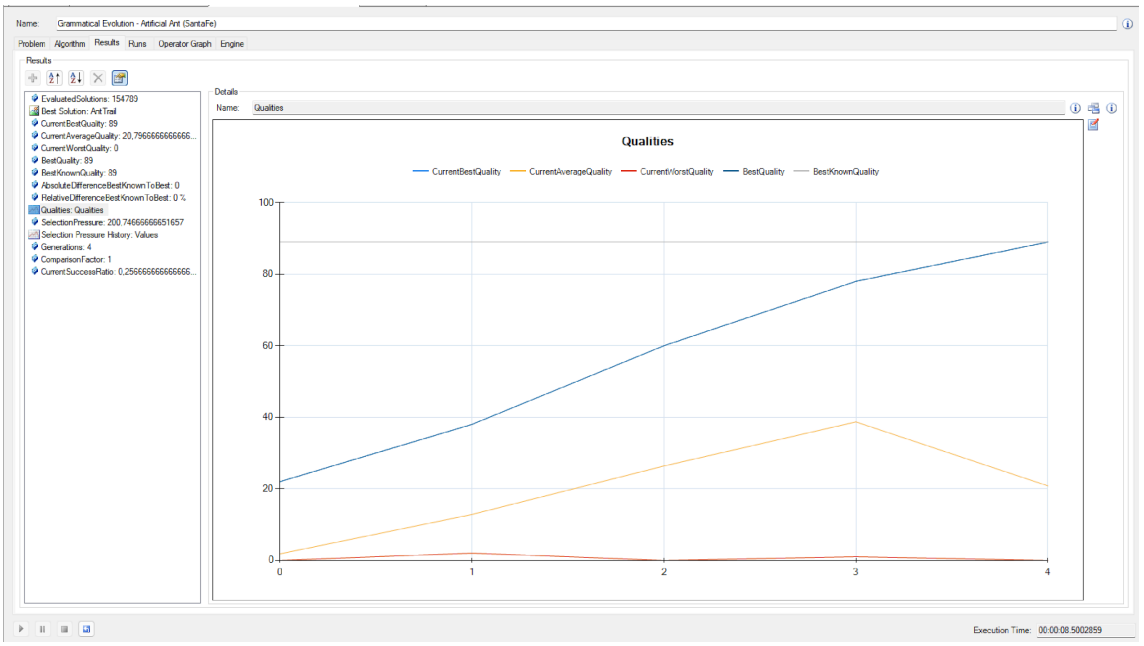
- ComparisonFactorModifier: LinearDiscreteDoubleValueModifier
- Crossover: SinglePointCrossover
- Elites: 1

- MaximumGenerations: 50
- MaximumSelectionPressure: 200
- MutationProbability: 10%
- Mutator: UniformOnePositionManipulator
- PopulationSize: 300
- SelectedParents: 200
- Selector: TournamentSelector (tamaño: 7)

Justificación: Esta configuración utiliza una selección por torneo más intensiva que favorecerá a los mejores individuos, con una tasa de mutación moderada del 10% para permitir cierta exploración del espacio de búsqueda.

- **TournamentSelector (tamaño 7):** Proporciona una presión selectiva alta pero no extrema, favoreciendo individuos con buen fitness pero manteniendo cierta diversidad.
- **MutationProbability 10%:** Un valor intermedio que permite suficiente variación genética sin destruir excesivamente las soluciones existentes.
- **UniformOnePositionManipulator:** Modifica una única posición en la gramática, lo que permite cambios más controlados y menos disruptivos.
- **SinglePointCrossover:** Cruza simple que intercambia una sección entre dos padres, manteniendo bloques cohesivos de código que pueden estar resolviendo partes específicas del problema.
- **PopulationSize 400:** Tamaño suficiente para mantener diversidad genética sin requerir recursos excesivos.
- **LinearDiscreteDoubleValueModifier:** Ofrece una relación lineal para modificaciones de valor, lo que facilita ajustes proporcionales y predecibles.



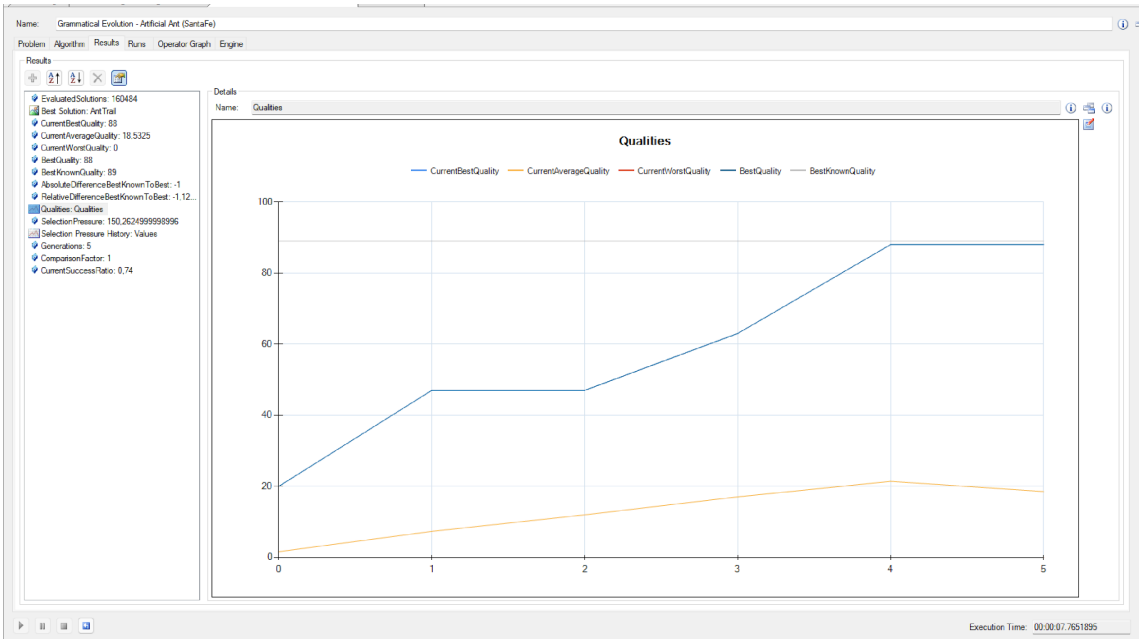


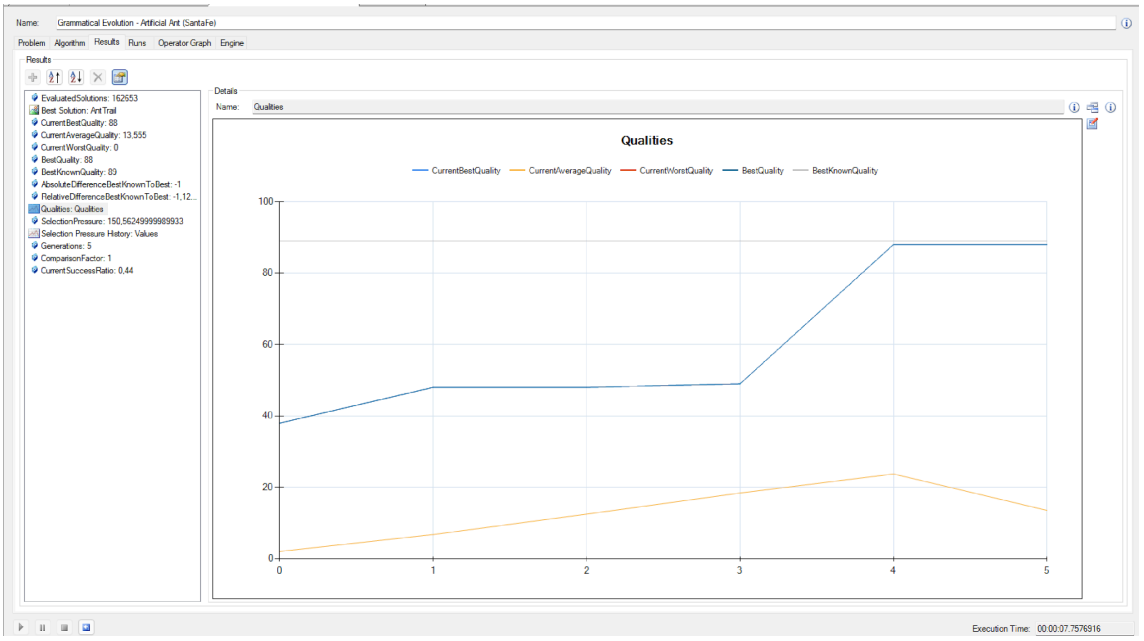
Modelo 5: Alta diversidad con enfoque en preservación genética

- ComparisonFactorModifier: ExponentialDiscreteDoubleValueModifier
- Crossover: RoundedLocalCrossover
- Elites: 3
- MaximumGenerations: 50
- MaximumSelectionPressure: 150
- MutationProbability: 5%
- Mutator: UniformSomePositionsManipulator

- PopulationSize: 400
- SelectedParents: 250
- Selector: GenderSpecificSelection
    - FemaleSelector: ProportionalSelector
    - MaleSelector: RandomSelector

Esta segunda configuración aprovecha el selector específico de género para mantener mayor diversidad, combinando la selección proporcional al fitness con un selector de diversidad. El crossover local y la mutación en varias posiciones permiten una exploración más fina del espacio gramatical, mientras que una población más grande con más élites preserva las mejores soluciones encontradas.



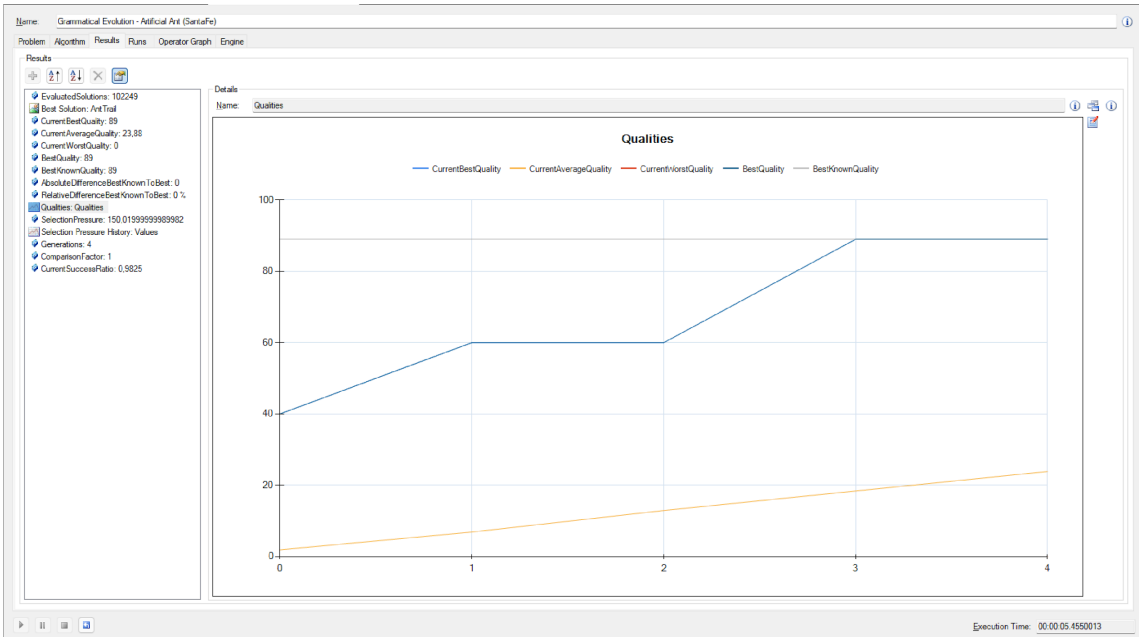


Justificación: Esta configuración aprovecha el selector específico de género para mantener mayor diversidad, combinando la selección proporcional al fitness con un selector de diversidad. El crossover local y la mutación en varias posiciones permiten una exploración más fina del espacio gramatical, mientras que una población más grande con más élites preserva las mejores soluciones encontradas.

- **GenderSpecificSelection:** Permite usar estrategias de selección diferentes para "machos" y "hembras", lo que aumenta la diversidad al promover diferentes características.
- **ExponentialDiscreteDoubleValueModifier:** Permite cambios más drásticos en ciertos valores, aumentando la capacidad de exploración en regiones distantes del espacio de búsqueda.
- **RoundedLocalCrossover:** Intercambia material genético de manera más localizada, lo que preserva mejor los bloques constructivos evolutivos.
- **UniformSomePositionsManipulator:** A diferencia del manipulador de una posición, este modifica varias posiciones, permitiendo cambios coordinados en diferentes partes de la gramática.
- **Elites 3**: Preserva más soluciones élite entre generaciones, asegurando que no se pierdan buenas soluciones ya encontradas.
- **OffspringSelectionBeforeMutation:** Realizar la selección antes de la mutación mantiene mayor integridad en el material genético seleccionado antes de introducir variaciones.

- **PopulationSize 400**: Una población mayor permite explorar más ampliamente el espacio de soluciones y mantener mayor diversidad genética.

Curiosidades: Respecto al programa de Gramáticas Evolutivas cabe destacar que hay un tipo de cruce, **RoundedHeuristicCrossover**, el cual hace uso de heurísticas para ver cuál es el mejor individuo en el cruce. Su funcionamiento es muy buena con casi cualquier combinación de parámetros, aumentando altamente el porcentaje de mutación (40%).



Además, hay un parámetro, MaximumSelectionPressure, el cuál esta como condición de finalización del algoritmo, si el algoritmo no ha mejorado ese tanto por ciento especificado (un valor de 150 indica que mejore al menos un 50%) con respecto la generación anterior, finaliza el algoritmo. Por eso hay tan pocas generaciones en las gráficas de las gramáticas

9. Bloating

Para el bloating hemos implementado tres técnicas explicadas en las diapositivas de la asignatura: el método tarpeian, la penalización bien fundamentada y la poda de árboles. La primera funciona tanto para gramáticas como para árboles, las otras dos solo funcionan para árboles.

El método tarpeian consiste en penalizar con probabilidad del 50% a los individuos cuyo tamaño supera la media. El tamaño es la cantidad de nodos que tienen (el

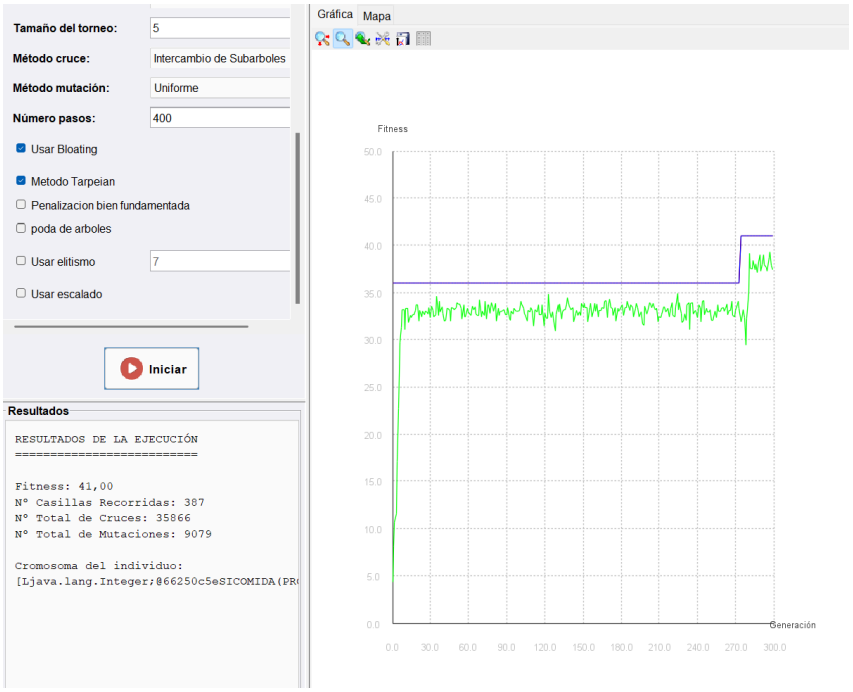
número de funciones o terminales). Hace que las soluciones que se encuentren sean más cortas y las ejecuciones menos lentas.
La penalización bien fundamentada consiste en penalizar los fitness según su varianza y covarianza, con la siguiente fórmula f'(x) = f(x) + k * nodos-en(x) kt = Covarianza(l,f) / Varianza(l). Hace que los tamaños no varíen mucho.
La poda de árboles se hace a la hora de cruzar, cuando un árbol supera la profundidad máxima establecida. Si eso sucede, la función de poda corta todos los nodos que hay por encima de la profundidad, y los convierte en terminales aleatorios. Así conseguimos no excedernos y hacer árboles más limitados en tamaño.

10. GRAMÁTICAS

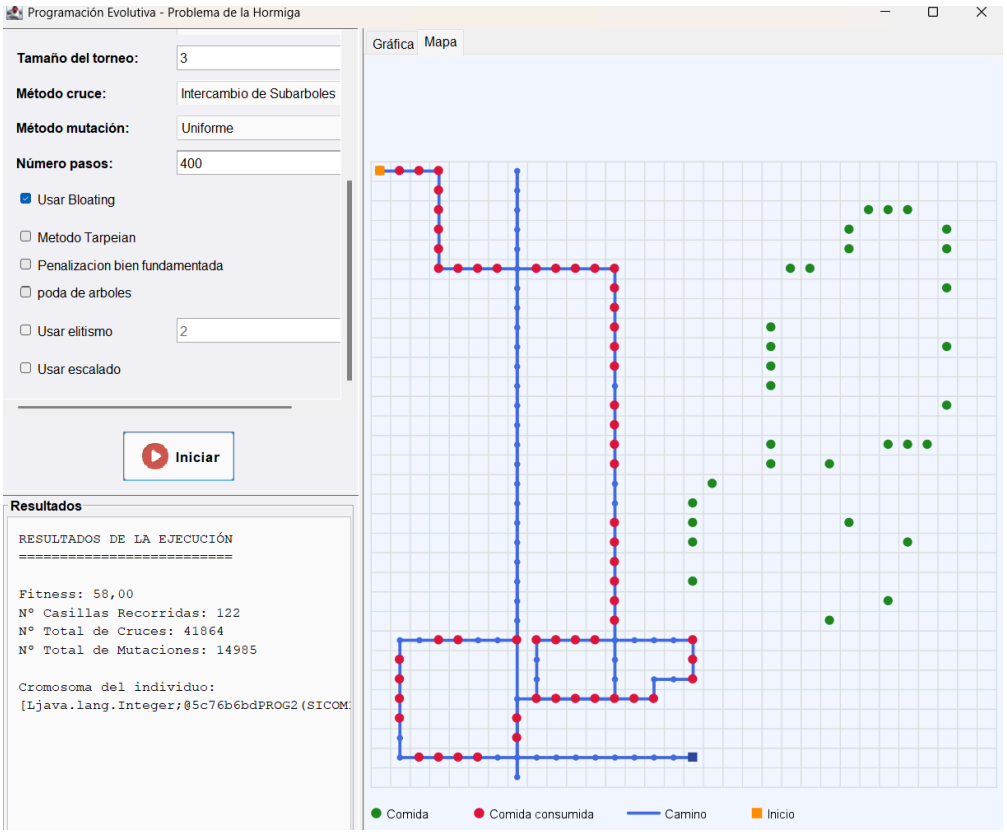
Para los individuos que funcionan con gramáticas hemos decidido implementar la siguiente lógica:
<prog> ::= <func> | <term>
<func> ::= SICOMIDA(<prog>,<prog>) | PROG2(<prog>,<prog>) | PROG3(<prog>,<prog>,<prog>)
<term> ::= AVANZA | DERECHA | IZQUIERDA

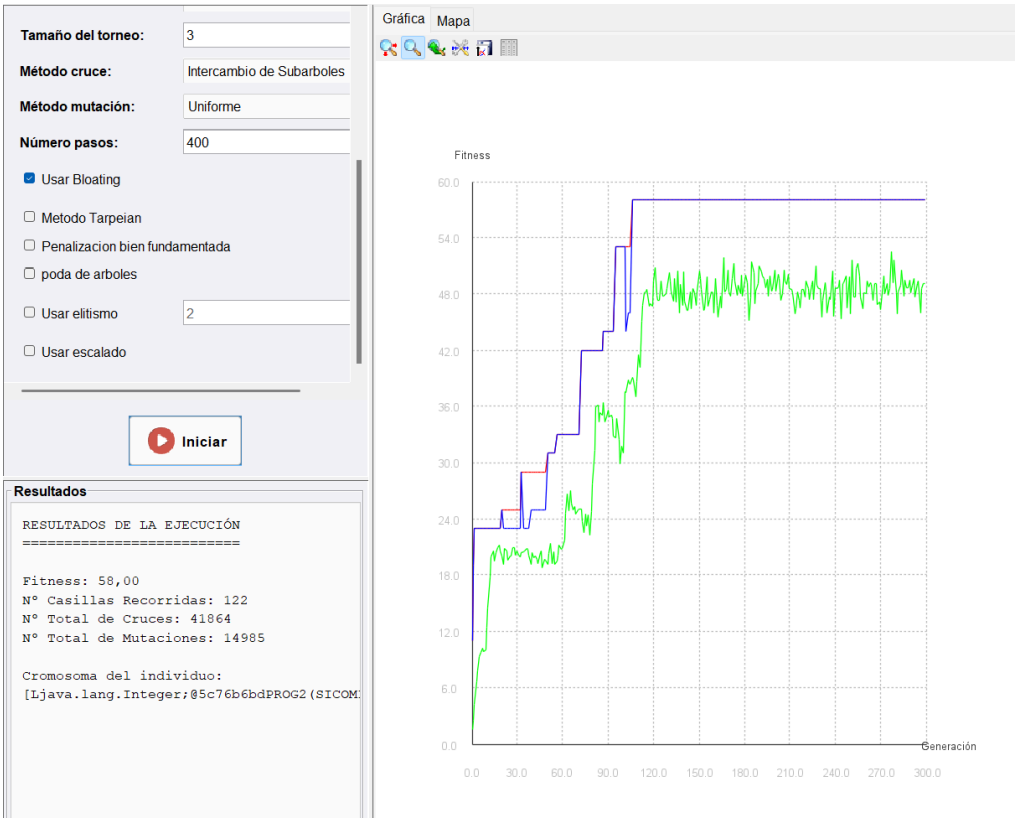
El problema con las gramáticas es que no llegan nunca a la mejor solución, pese a que su funcionamiento parece ser correcto. La función fitness es exactamente igual que la de los árboles: por cada "nodo", si es una función ejecuta a sus hijos, y si es un terminal mueve a la hormiga, devolviendo el número de comida recogida. Tiende a converger muy deprisa hacia los mismos valores, produciendo poblaciones donde todos los individuos son iguales, lo que hace que se estanque en ocasiones. Los mejores resultados se obtienen con torneo determinístico de 5, mutación entre el 15 y el 25%, y cruce de 60-70%. Usamos wraps que pueden ser modificados desde la GUI, y un cromosoma de tamaño 15. Como métodos de mutación tenemos el uniforme y el monopunto, y el cruce es el de intercambio por un punto en ambos cromosomas.

Ejemplo de “estancamiento”:

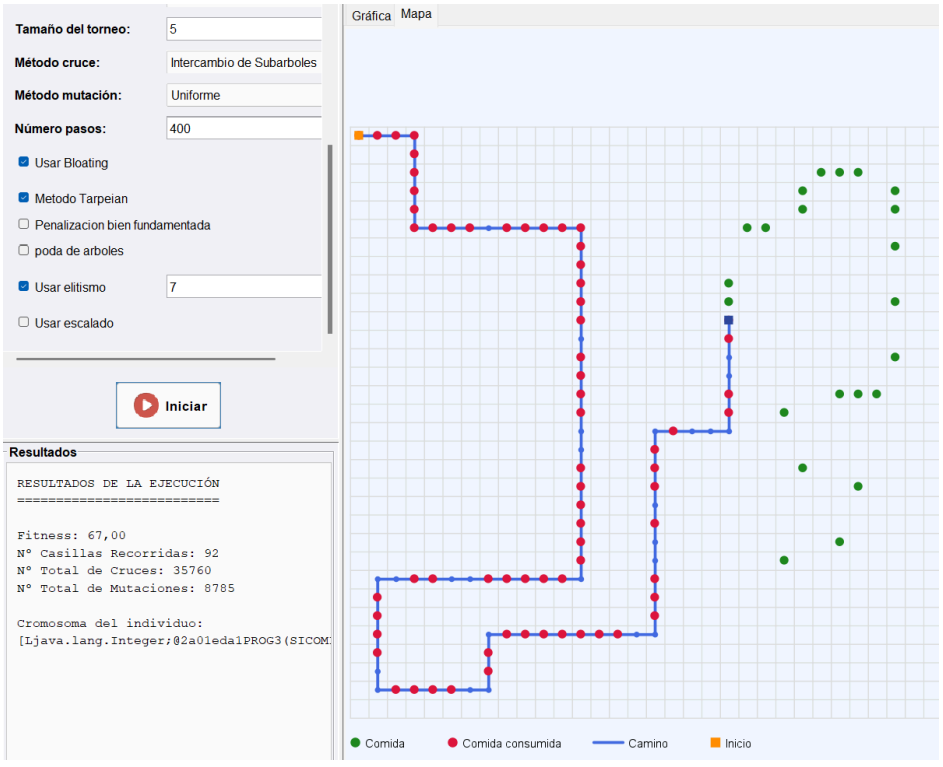


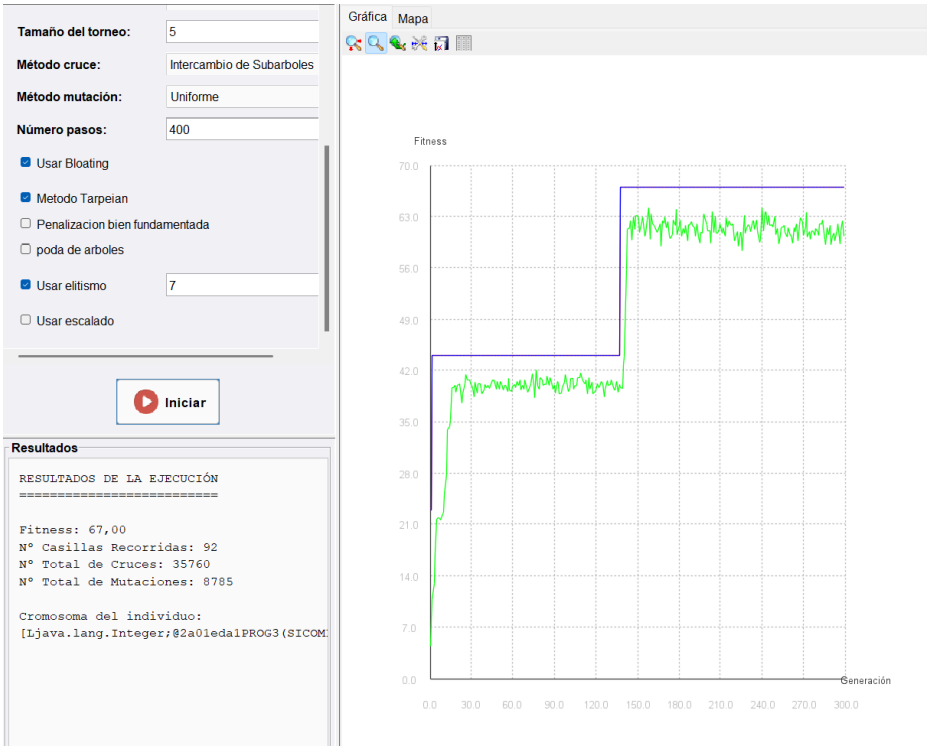
Ejemplo de ejecución más evolutivo:





Ejemplo con elitismo:





11. Análisis Comparativo y Conclusiones

A partir de los experimentos realizados, podemos extraer las siguientes conclusiones:

1. Sobre los operadores de selección: Los métodos basados en torneos con tamaño moderado (3-5) proporcionaron el mejor equilibrio entre presión selectiva y diversidad. La selección por ranking también mostró buenos resultados al preservar mejor la diversidad genética.

2. Sobre los operadores de cruce: El cruce de subárboles demostró ser el más efectivo para programación genética, mientras que para gramáticas evolutivas el cruce monopunto mostró mejores resultados que el uniforme.

3. Sobre los operadores de mutación: La combinación de múltiples operadores produjo mejores resultados que el uso de un único operador. La mutación de subárboles proporciona exploración global, mientras que las mutaciones funcional y terminal permiten un refinamiento local.

4. Sobre el número de pasos: Se observó una correlación positiva entre el número máximo de pasos permitidos y la calidad de las soluciones. Con 600 pasos se obtuvieron los mejores resultados, aunque esto aumenta el tiempo de evaluación.

5. Sobre los enfoques comparados: El Modelo 3 (mutaciones mixtas) mostró el mejor rendimiento global, seguido por el Modelo 4 (gramáticas evolutivas con cruce monopunto). Las gramáticas evolutivas ofrecen la ventaja adicional de garantizar la validez sintáctica.

12. REPARTO DE TAREAS

El trabajo ha sido repartido a partes iguales, con la siguiente distribución:
Javier: vistas de las ventanas y GUI, modificaciones en clase árbol, fitness y algoritmo genético, cambios en cruces, individuos y mutaciones, depuración y corrección de errores, heuristic lab y trabajo en la memoria.
Clara: modificaciones en clase árbol, fitness y algoritmo genético, cambios en cruces, individuos y mutaciones, depuración y corrección de errores, creación de gramáticas y sus clases y trabajo en la memoria.