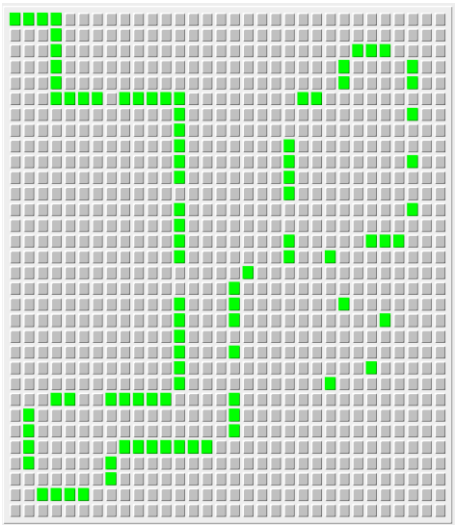
Programación Evolutiva:
Práctica 3.

Pablo Mac-Veigh
Jorge Sánchez
https://github.com/Pabsilon/PracticasPE

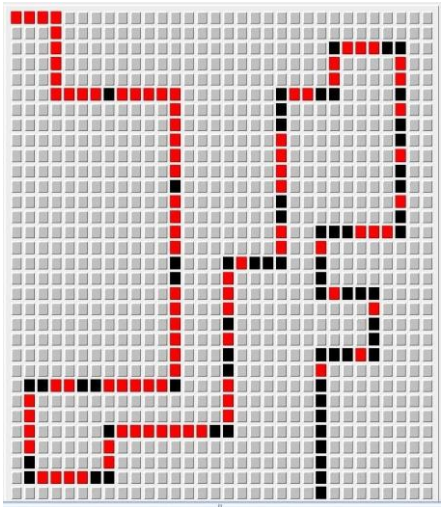
Hemos realizado una práctica en Java de programación genética, que representa en forma de árboles un programa de una hormiga artifical que busca un rastro: El rastro de Santa Fé.

Aquí el original de nuestro programa:



Las casillas verdes representan la comida.

Una vez ejecutado el problema, las casillas negras indicarán el camino tomado por la hormiga, y las casillas rojas, los fragmentos de comida que han sido comidos por la hormiga. He aquí una solución:



Datos de entrada:

- **Población**
- **Generaciones**
- **Profundidad**
- **Método de Creación**
- **Método de Selección**
- **Elitismo**
- **Método de Cruce**
- **Probabilidad de Cruce**
- **Probabilidad de Mutación**
- **Método de Mutación**
- **Bloating**
- **Método de Bloating**
- **Semilla**

Método de uso:

La interfaz es bastante similar a las prácticas anteriores, con la diferencia de que mostramos por defecto el mapa de Santa Fé en vez de la Gráfica, que es accesible desde los tabs de la esquina superior izquierda. Todos los parámetros de entrada son editables.

Población:

El número de individuos en cada generación del programa. Un número bajo resta diversidad, y un número alto merma el rendimiento del programa y devuelve un resultado conseguido más por fuerza bruta y menos por evolución.

Generaciones:

El número de iteraciones del algoritmo. Al igual que con la población, un número bajo no da mucha posibilidad de evolucionar, y un número bajo puede resultar en una búsqueda exhaustiva, aunque tienda a estancarse.

Profundidad:

La profundidad máxima de los árboles que generamos. Un valor muy bajo (<=2) reduce drásticamente el espacio de búsqueda, reduciendo el tiempo de ejecución, y posiblemente evitando llegar a una solución. Un valor muy alto (<=5) genera un árbol enorme, complicando muchísimo la búsqueda, el tiempo de ejecución y devolviendo un programa inecesariamente complicado. El valor más óptimo para este problema es 4.

Método de Creación:

Hemos creado tres métodos para la creación de árboles:

- Creciente: Tomamos nodos del conjunto completo (terminales y funciones) hasta llegar a la profundidad máxima y rellenamos.
- Completa: Tomamos únicamente nodos del conjunto de funciones y al llegar a la profundidad máxima rellenamos con nodos terminales.
- Ramped & Half: La mitad de la población es inicializada como creciente, y la otra mitad como completa.

Sin ninguna duda, la mejor forma de inicializar es con R&H, ya que creamos una diversidad mucho mayor que con cada método individualmente.

Método de Selección:

Hemos implementado 5 métodos de selección:

- Ruleta: Una selección aleatoria. La probabilidad de ser elegido es directamente proporcional a la aptitud del individuo.
- Ranking: La probabilidad de ser elegido se ve reducida en los individuos con peor aptitud. Utilizamos Beta = 1.5.
- Restos: Se multiplica la aptitud por el número de individuos y se redondea hacia el entero de abajo. Se añaden ese número de copias a la piscina genética de la siguiente generación.
- Truncamiento: Elegimos el top 25% de la población y rellenamos la nueva población únicamente con esos individuos.
- Torneo: Elegimos al azar n individuos y nos quedamos con el mejor. Repetimos hasta rellenar la población nueva.

En nuestra experiencia, los dos primeros métodos son demasiado aleatorios, y los dos siguientes merman mucho la diversidad. Al igual que en la primera práctica, el Torneo de 3 participantes es el mejor método para seleccionar individuos.

Elitismo:

El elitismo evita que se pierdan los mejores individuos de la población, ya que ningún método garantiza que se queden. Nosotros nos quedamos con el top 2% de la población. Se obtienen mejores resultados si está activado.

Método de Cruce:

Sólo tenemos el método de intercambio Implementado. Generamos un punto de corte en el árbol Padre y otro en el árbol Madre e intercambiamos.

Probabilidad de Cruce:

La probabilidad de que al elegir dos individuos, estos se crucen. Como en las otras dos prácticas, el mejor valor está en torno al 60%.

Probabilidad de Mutación:

La probabilidad, de después de cruzar la población, de que los hijos desarrollen cambios no heredados de los progenitores.

En prácticas anteriores, el valor óptimo estaba en torno al 5%. En este caso, como los fenotipos no tienen tantos valores, una mutación de 15%-20% da suficiente diversidad a la nueva población.

Método de Mutación:

Hemos implementado tres métodos de mutación:

- Árbol: Elige un subárbol y lo muta: Genera un nuevo árbol por el método creciente.
- Función: Elige una función y la muta: Puesto que sólo hay una función de 3 parámetros esta no muta, pero si mutan de forma aleatoria las de dos.
- Terminal: Elige un terminal y lo muta.

Por lo que hemos podido observar, la mutación de Árbol es la que mejor resultados obtiene.

Bloating:

El crecimiento casi exponencial de los programas generados es solucionado con al introducir una función que reduce el bloating.

Método de Bloating:

Hemos implementado dos métodos para reducir el bloating de los programas generados:

- Tarpeian: Calculamos la media de profundidad de los árboles de la población, y le ponemos aptitud 0 a los elementos que tengan más que la media.
- Penalización: Le restamos aptitud a los elementos que tienen mucha profundidad.

El método de Tarpeian funciona bastante bien y se obtiene un rendimiento mucho mejor que con penalización.

Semilla:

Al igual que en las prácticas anteriores, hemos añadido la posibilidad de introducir una semilla numérica que se utiliza para generar la población inicial.

Si la dejamos en 0 utiliza una semilla "Aleatoria" conseguida a través de System.currentTimeMillis(), que luego es mostrada en la GUI.

Resultados:

Los resultados del programa se muestran en el mapa (ejecución del mejor programa), en el panel de la derecha donde se muestra el código generado, y en el panel de la gráfica donde podemos observar la evolución de:

- Mejor absoluto: En azul, el mejor individuo conseguido hasta el momento
- Mejor de la generación: En rojo, el mejor individuo de una generación.
- Media de la generación: La aptitud media de los individuos de la generación.

Estructura del programa generado:

Se aplican estos métodos sobre un árbol que se traduce a un programa. Cada nodo de los árboles puede ser:

- Terminal: Hay tres terminales: "Avanza", "Derecha", "Izquierda". Como su nombre indican, no tienen hijos en el árbol, y su nombre es bastante representativo de la acción.
- Funciones: Hay tres funciones: "SIC(a,b)", "PROGN2(a,b)", "PROGN3(a,b,c)". Estos nodos tienen dos (sic, progn2) hijos o tres (progn3) que pueden a su vez ser funciones o terminales.
    - SIC(a,b): "Si comida delante ejecuta a, sino, b".
    - PROGN2(a,b): evalua a, luego b, y devuelve b.
    - PROGN3(a,b,c): evalua a, b, luego c, y devuelve el valor de c.

Resultados obtenidos:

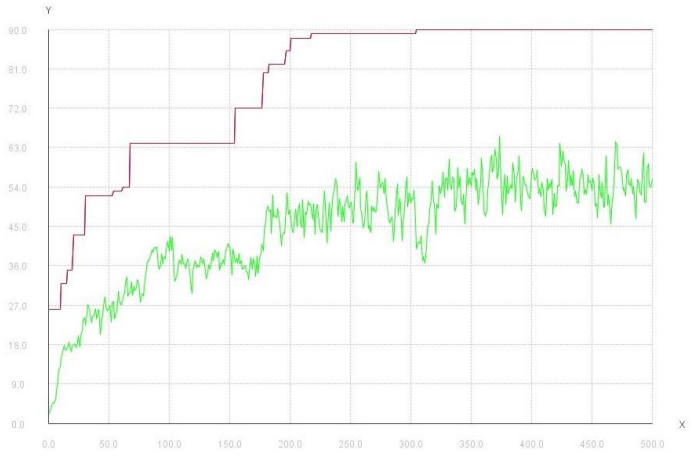
Hemos encontrado el resultado óptimo con los siguientes parámetros:

Población: 200 ; Generaciones: 500 ; Profundidad 4 ; Método de Creación: RampedHalf ; Método de Selección : Torneo de 3 ; Elitismo: Si ; Método de Cruce: Intercambio ; Probabilidad de Cruce: 60 ; Probabilidad de Mutación : 15 ; Método de mutación : Árbol ; Bloating: Si ; Método de Bloating: Penalización.

El resultado obtenido es el de la segunda imagen de la primera página; y el programa generado es:

```
(SIC     (AVANZA)
         (PROGN3        (PROGN3        (GIRA_IZQUIERDA)
                                       (SIC     (GIRA_IZQUIERDA)
                                                (GIRA_DERECHA))
                                                        (GIRA_DERECHA))
                        (SIC     (PROGN2        (AVANZA)
                                                (AVANZA))
                                 (SIC     (AVANZA)
                                          (GIRA_IZQUIERDA)))
                        (AVANZA)))
```

La gráfica resultante de la evolución es la siguiente:

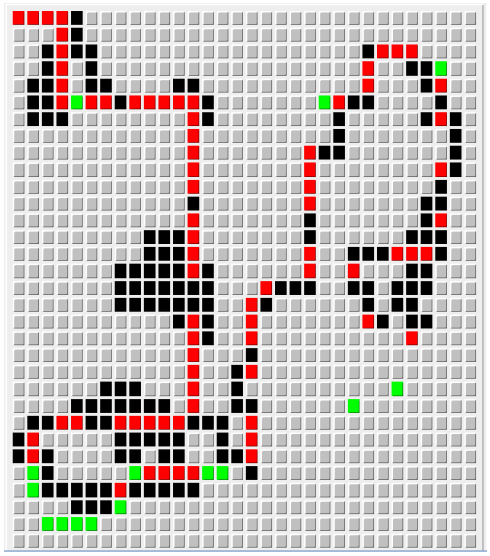


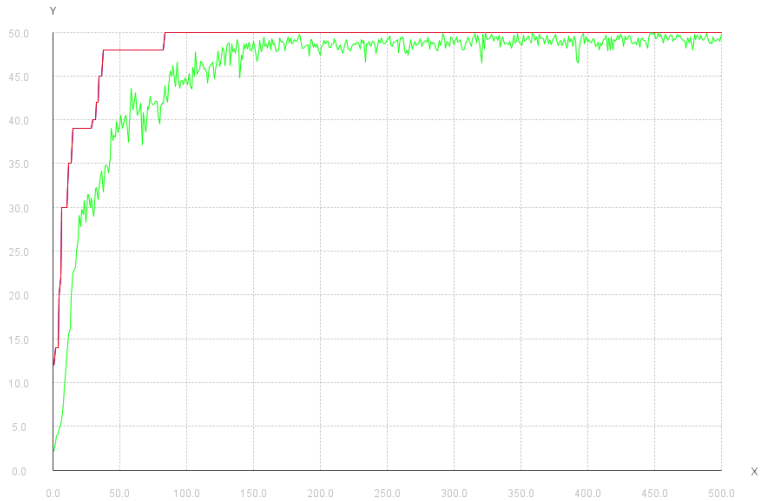
Donde se puede ver una mejora casi constante tanto del mejor (absoluto y de generación, ya que hay elitismo) como de la media, encontrando el mejor resultado justo pasadas las 300 generaciones.

Otros valores:

Todos estos valores son cambiando un único parámetro de los anteriores, para poder analizar los distintos resultados partiendo de la misma base.

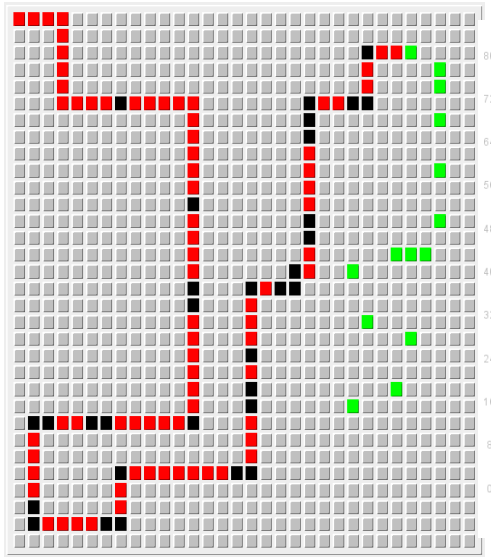
- Bloating Tarpeian.

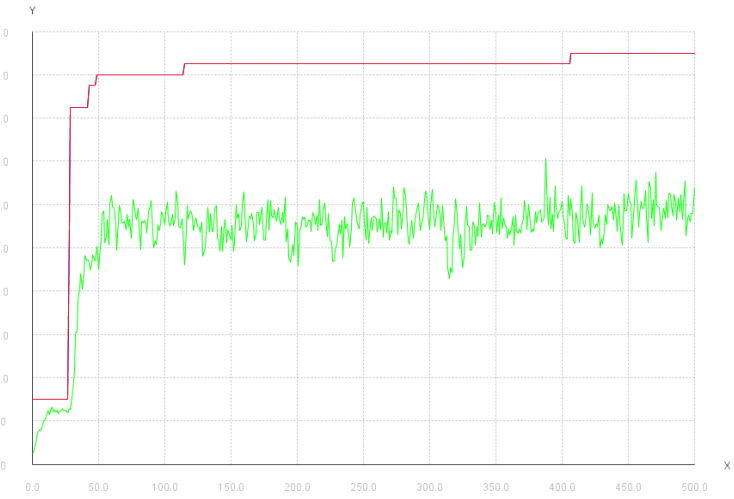




La complejidad del problema aumenta de forma considerable; el tiempo de ejecución se multiplica, y el progreso de mejor se estanca muy rápido – pero la media tiende al mejor valor.

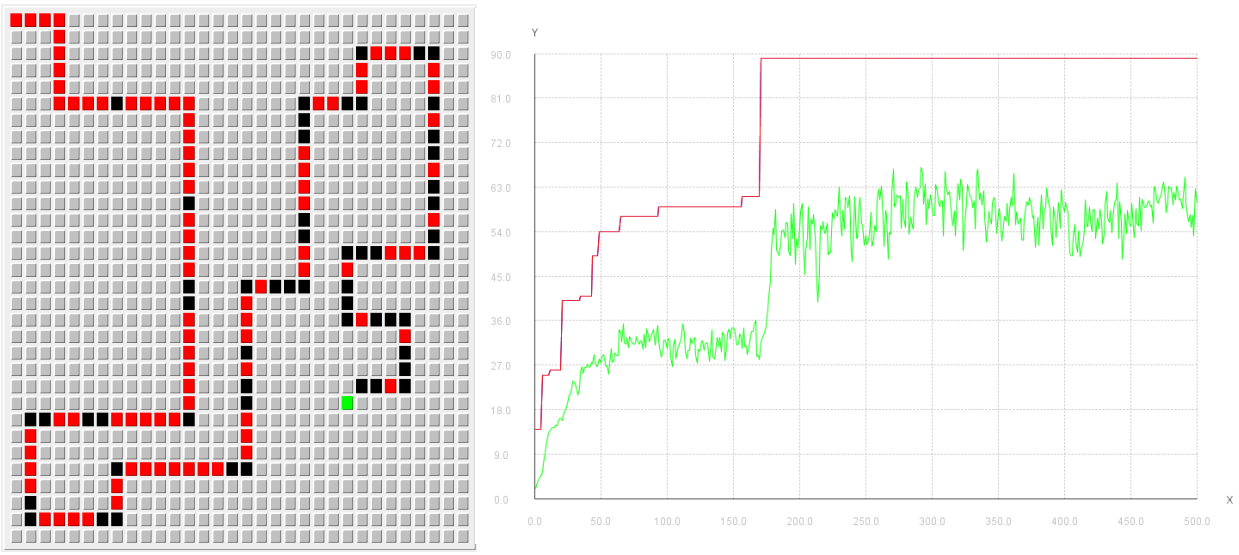
- Mutación de Funcion.





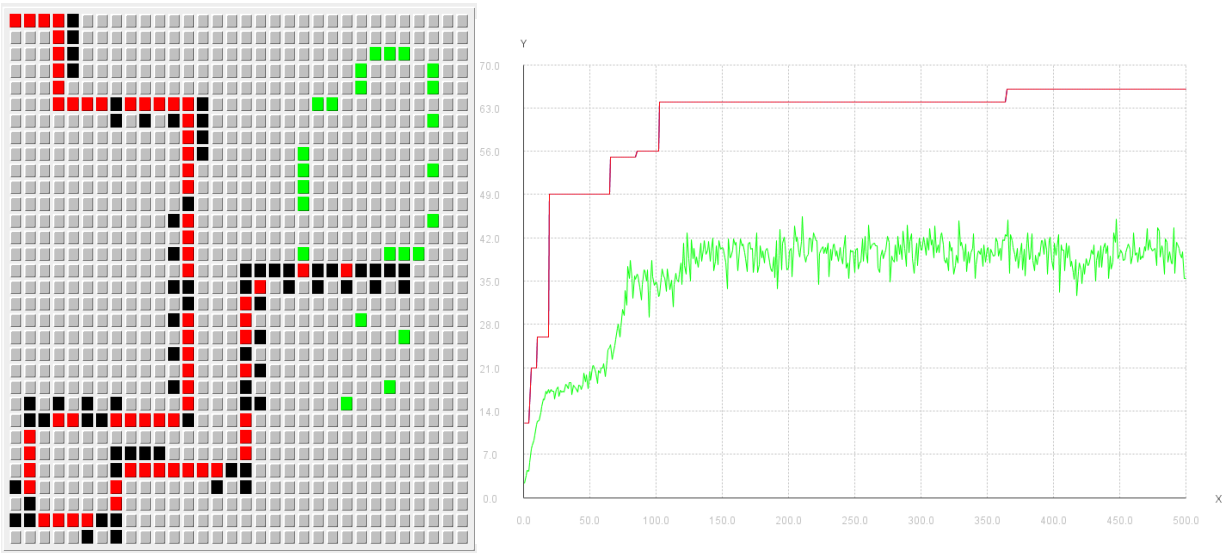
Obtenemos un tiempo de ejecución más corto, y el programa devuelto es bastante simple. Vemos una repentina mejora de la aptitud, seguido de un estancamiento de la media.

- Mutación de Terminal



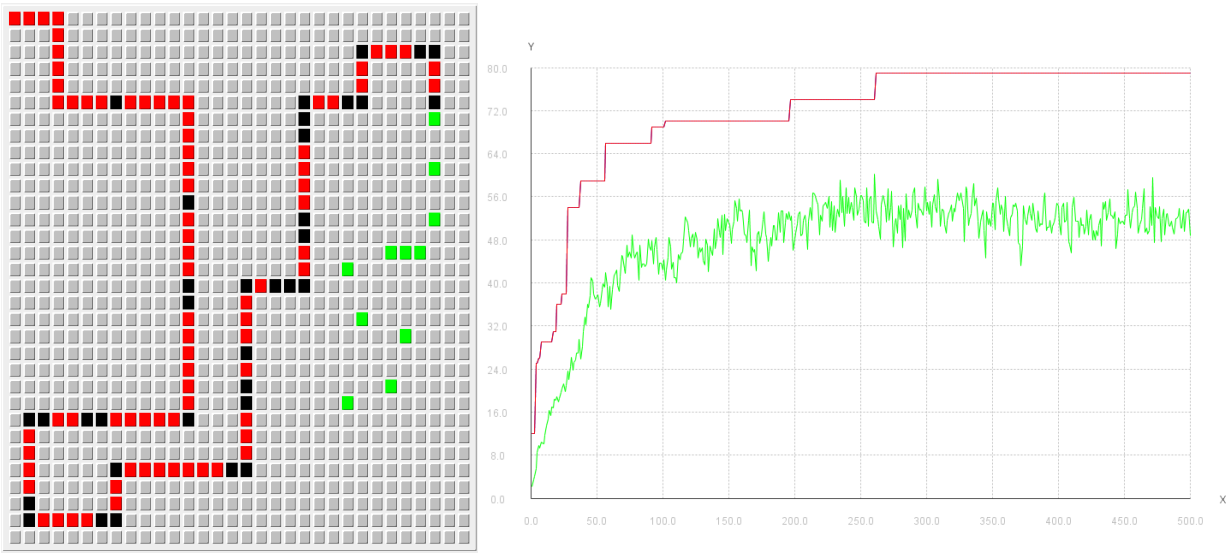
Obtuve este resultado repentinamente. Podemos observar que había una lenta evolución, y encontramos un resultado que da un salto enorme tanto en el mejor individuo como en la media.

- Selección por ranking.



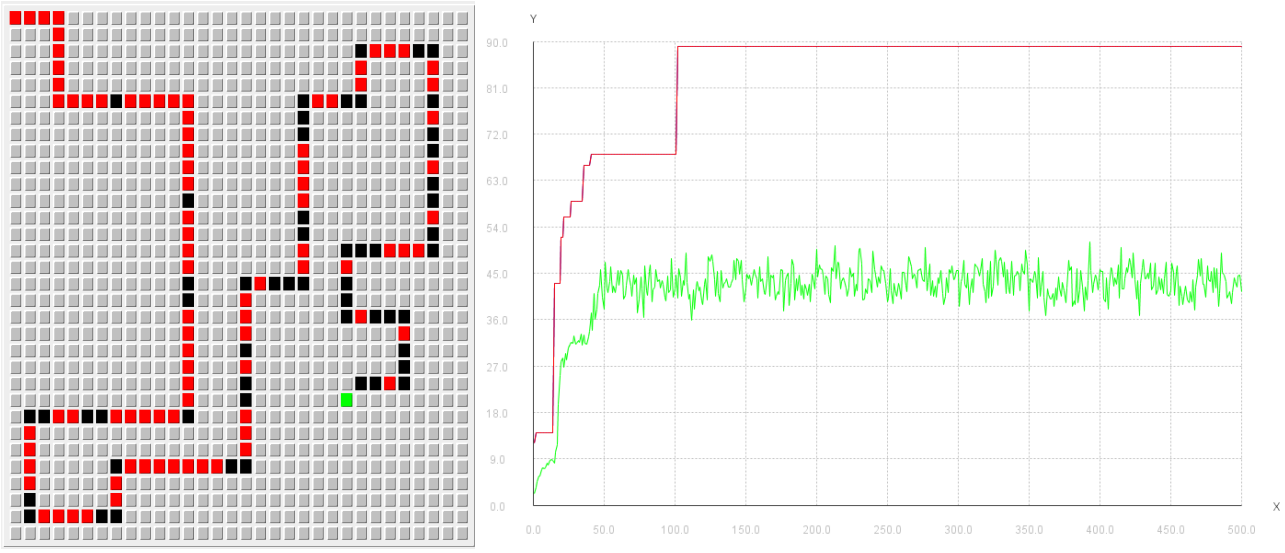
La selección por ranking ya nos hace perder una “solución limpia”. El programa solución es mucho más complejo que los anteriores. Vemos una leve mejora del mejor individuo en las primeras 100 generaciones, pero luego se estanca de forma considerable.

- Selección por restos.



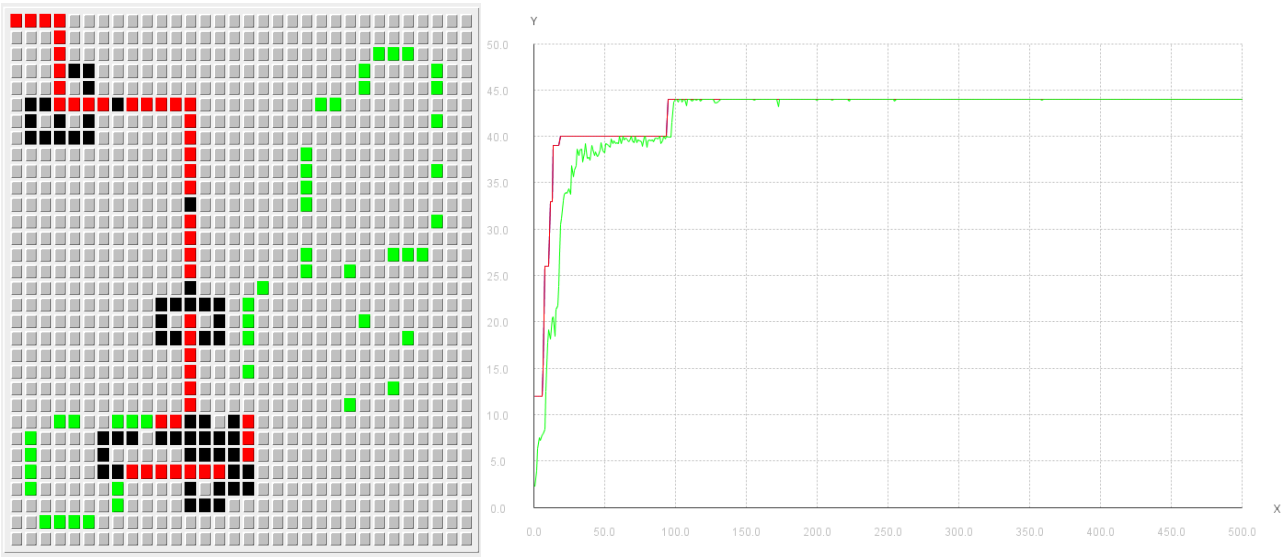
Se obtiene un resultado bastante aceptable. El programa solución es relativamente sencillo, y la gráfica evoluciona de forma similar a la mejor solución encontrada.

- Selección por ruleta.



Obtenemos una respuesta muy cercana a la óptima, salvo por que falta un bocado. Como vemos en la gráfica, obtenemos este resultado muy pronto. También se observa que la media se estanca muy pronto, y por cómo es la selección por ruleta, puede haber sido suerte.

- Truncamiento:



El tiempo de ejecución es terrible. La solución está lejos de ser aceptable. Al copiar siempre al top 25% de la población, esta se estanca, convergiendo así la media, el mejor valor de la generación y el mejor valor absoluto.