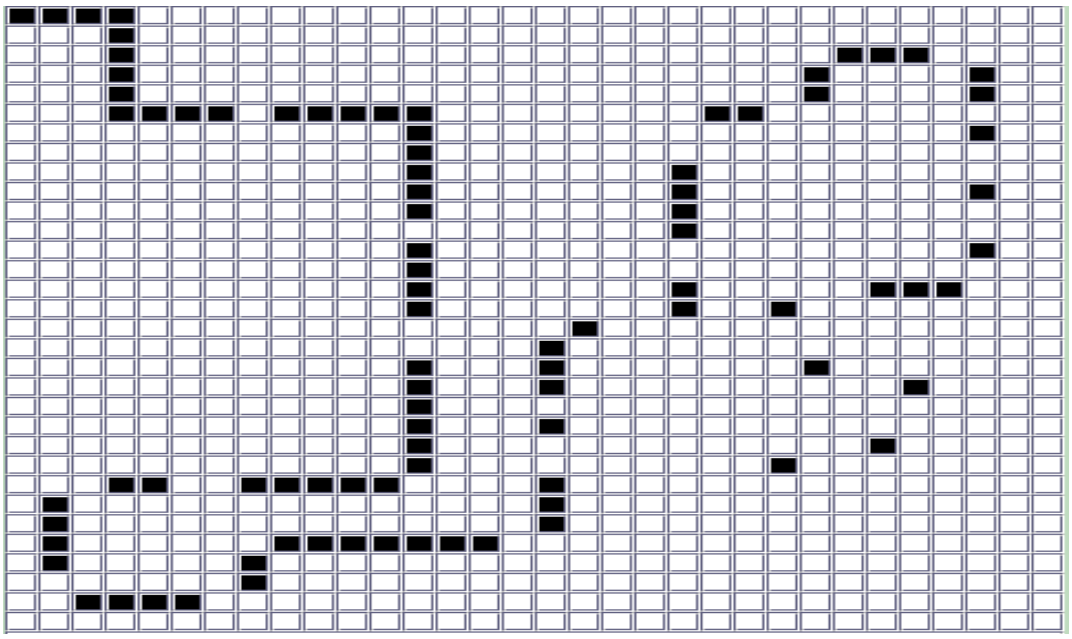
Programación evolutiva
Facultad de Informática
Curso 2015/2016

Práctica 3: Programación genética

En programación genética se hacen evolucionar programas representados en forma de árboles, que son los individuos con los que trabaja el algoritmo evolutivo.
La práctica consiste en el diseño de una hormiga artificial capaz de encontrar toda la comida situada a lo largo de un rastro irregular. El objetivo es utilizar programación genética para obtener el programa que realiza esta tarea. La hormiga artificial se mueve en un tablero (toroidal) de 32 x 32. La hormiga comienza en la esquina superior izquierda del tablero, identificada por las coordenadas (0,0), y mirando en dirección este.
Un modelo de rastro propuesto, con el que se han realizado diversos estudios, se conoce como "rastro de Santa Fe" , y tiene una forma irregular compuesta de 89 "bocados" de comida. El rastro no es recto y continuo, sino que presenta huecos de una o dos posiciones, que también pueden darse en los ángulos.



Nuestro problema requiere operaciones que permitan a la hormiga moverse hacia delante, girar a uno u otro lado y detectar comida a lo largo de rastro irregular. El conjunto de operandos (terminales) para este problema es:

Terminales = {AVANZA, DERECHA, IZQUIERDA}

donde AVANZA mueve la hormiga hacia delante en la dirección a la que mira en ese momento, DERECHA gira la hormiga 90° a la derecha, e IZQUIERDA la gira 90° a la izquierda.
Las funciones disponibles son:

Funciones = { *SIC*(a,b), *PROGN2* (a,b), *PROGN3* (a,b,c) }

La información que queremos procesar es la que la hormiga obtiene del mundo exterior a través de un sencillo sensor que detecta si hay comida en frente. Por ello incluiremos en el conjunto de operadores uno de selección condicional dependiente de la información del sensor.

En el conjunto de operadores incluiremos también conectivas que fuerzan la ejecución de sus argumentos en un determinado orden. Tomando el nombre de una conectiva de Lisp, PROGN, que causa este efecto, incluimos en el conjunto de operadores las conectivas PROGN2 y PROGN3, que toman dos y tres argumentos respectivamente. Por lo tanto disponemos de las siguientes funciones:

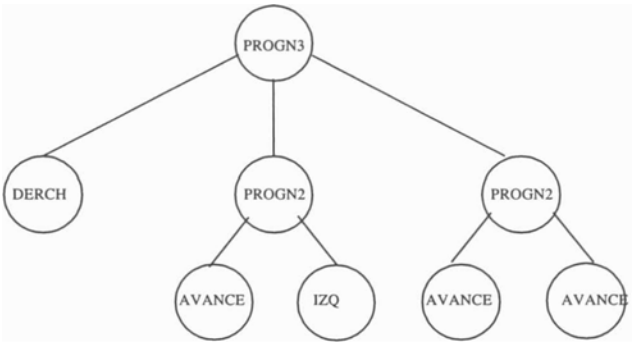
- ***SIC (a,b) :*** el operador SI_Comidadelante toma dos argumentos y ejecuta *a* si se detecta comida delante y *b* en otro caso.
- ***PROGN2(a,b)*** : evalúa *a*, luego *b*, y devuelve el valor de *b*.
- ***PROGN3(a,b,c)*** : evalúa *a, b* luego *c*, devolviendo el valor de *c*

Un par de ejemplos de programas:

(PROGN3 (SIC) (SIC) (PROGN3 (SIC) (SIC (SIC AVANZA DERECHA) (SIC DERECHA DERECHA)) AVANZA))

(PROGN3 (DERECHA) (PROGN2 (AVANZA) (IZQUIERDA)) (PROGN2 (AVANZA) (AVANZA))

Esta última expresión representa en forma de árbol:



El algoritmo original de Koza puede describirse en los siguientes puntos:

- ❑ Se genera un población inicial de programas aleatorios (árboles) usando el conjunto de funciones y terminales posibles. Estos árboles deben ser sintácticamente correctos. Limitamos la profundidad.
- ❑ Una medida natural de la aptitud para este problema es la cantidad de alimento comido por la hormiga dentro de un espacio de tiempo razonable al ejecutar el programa a evaluar. Se considera que cada operación de movimiento o giro consume una unidad de tiempo. En nuestra versión del problema limitaremos el tiempo a 400 pasos. El tablero se va actualizando a medida que desaparece la comida.
- ❑ Se aplican los operadores de selección, cruce y mutación para producir un cierto número de generaciones con una elección adecuada de parámetros para el algoritmo.
- ❑ Representación de los Individuos: Los individuos estarán formados, además de por los datos que se requieran para el algoritmo evolutivo, por un árbol que es la representación sintáctica de una función.
- ❑ Operador de Cruce: Intercambiar dos subárboles (elegidos aleatoriamente) entre los dos árboles padres.
- ❑ Operador de Mutación: de terminal, de función, de inicialización.
- ❑ Posible control del bloating para mejorar la adaptación y evitar programas muy largos
- ❑ Visualización de la ejecución de un programa: representar gráficamente la evolución de un juego en el tablero al aplicar un programa de buena adaptación.

Documentación a entregar

- ❑ Hay que enviar al campus virtual antes del **24 de mayo a las 12:00** un archivo comprimido con el código java de la aplicación (**proyecto en Eclipse** o **NetBeans)** cuyo nombre se corresponda con el nombre del grupo y las siglas **P3,** por ejemplo **G01P3.**
- ❑ En el archivo comprimido se incluirá una breve memoria que contenga el estudio de las gráficas y resultado obtenidos con cada función. Aquí se valorarán las conclusiones y observaciones que se consideren interesantes respecto al resultado obtenido, a las mejoras utilizadas y a los métodos propios
- ❑ El día de corrección será en la sesión de Laboratorio del **24 de mayo** y deberán estar presentes los dos integrantes del grupo. La práctica deberán conocerla a fondo los dos componentes del grupo pues se harán preguntas a ambos indistintamente.

La corrección será en la sesión de laboratorio del martes 24 de mayo.