TEAM_GETPPID__
www.wuolah.com/student/TEAM_GETPPID__
32512

Metahuristica Tema 4.pdf

Resúmenes Metaheurística

3° Metaheurísticas

Grado en Ingeniería Informática

Escuela Politécnica Superior de Córdoba
UCO - Universidad de Córdoba

2019

Tema 4: Metaheurísticas basadas en poblaciones. Programación genética

Contenido

1. Introducción a la Programación Genética

La Programación Genética es una metaheurística poblacional cuya estructura es básicamente igual a la de un AG. La característica diferencial de GP es que los individuos representan "entidades ejecutables", que solucionan un determinado problema (la representación de las soluciones). Estas entidades pueden ser:

- Código fuente para interpretar o compilar
- Estructuras directamente ejecutables

La disciplina data de 1985, aunque el gran difusor ha sido John Koza a principios de la década de los 90.

1.2 Características básicas

Los algoritmos de GP tienen la estructura básica de otros EAs. El esquema algorítmico más popular es el de estado estacionario, u otros en los que se aplique un elitismo fuerte, dado que los operadores son muy disruptivos.

```
Algoritmo GP
Inicio
  Crear población inicial
  Mientras (no criterio_parada()) hacer
    Seleccionar padres
    Crear hijos aplicando operadores
    Actualizar población
  Fin Mientras
Fin Algoritmo GP
```

2. Variantes de la Programación Genética

2.1 Canonical Genetic Programming

Esta es la estructura descrita por John Koza en el paper y también en el libro Genetic Programming, publicado en 1992.

2.1.1 Fundamentos

- **Individuo**: Una o más expresiones expresadas como árboles
- **Nodos del árbol de expresión**:
  - Terminales o nodos hoja (variables, constantes, funciones sin argumentos).
  - Funciones o nodos internos (operadores – principalmente unarios o binarios)
- El árbol hace la doble función de genotipo y fenotipo:
  - **Genotipo**: experimenta transformaciones genéticas
  - **Fenotipo**: El recorrido del árbol genera la expresión asociada a éste

2.1.2 Árboles de expresión.

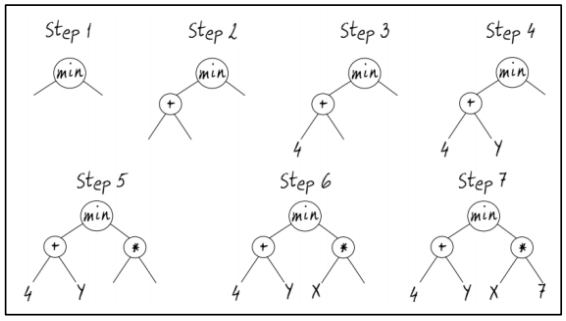
El cromosoma como tal codifica la expresión del árbol utilizando cualquier recorrido sobre él, habitualmente el preorden. Los cromosomas en forma de árboles son estructuras no lineales. Los árboles en PG pueden variar en profundidad y anchura.

- La evaluación de los árboles consiste en ejecutar el programa representado por el árbol.
- La ejecución se hace de forma distinta según la plataforma de desarrollo:
    - La formulación original de Koza utilizaba expresiones S en el lenguaje LISP.
    - Existen implementaciones en C, C++, Java...
    - En principio, para poder evaluar los árboles de expresiones, es necesario una evaluación recursiva, consistente en dar valor a los atributos de las hojas e ir subiendo en el árbol hasta devolver el valor asociado a la raíz. Sin embargo, se puede implementar de forma iterativa haciendo uso de una o más pilas para ir operando con la representación en preorden del árbol.
- La implementación influye en el coste y rendimiento del algoritmo

2.1.3 Inicialización de la población

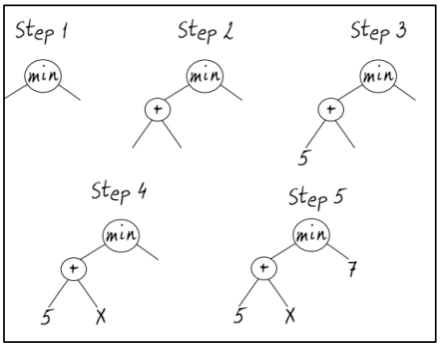
Se basa en una generación aleatoria de distintos programas (árboles de expresiones). La profundidad de un árbol se calcula como el número de nodos existentes desde la raíz hasta la hoja más lejana. Las formas más conocidas de generación son:

- **Full Method**: genera árboles cuyas hojas están a la misma profundidad. Que los árboles tengan la misma profundidad no quiere decir que tengan la misma forma (sólo ocurre si las funciones tienen la misma aridad).



Creación con Full Method

- **Grow Method**: genera árboles de cualquier forma con la única restricción de parar al alcanzar la profundidad máxima.
- El método **Half and Half** construye un 50% de árboles con el método Full y el otro 50% con el método Grow.



Creación con Grow Method

2.1.4 Operadores genéticos: Cruce

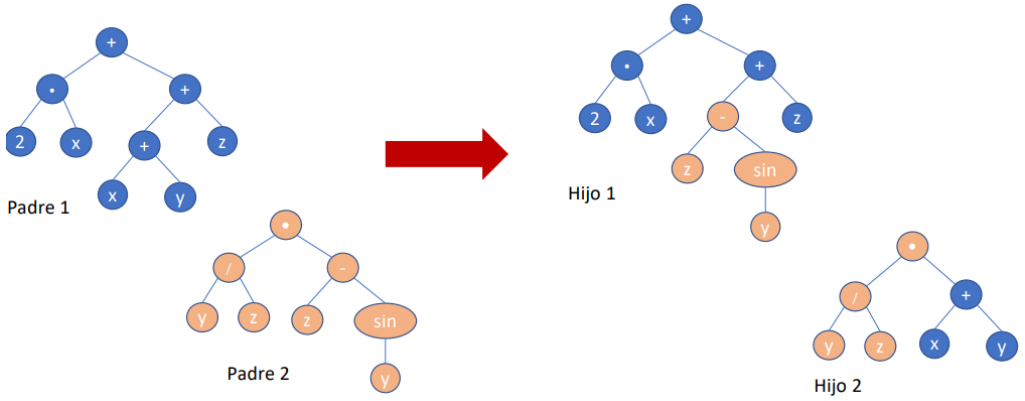
Se escogen aleatoriamente dos aristas y se intercambian los subárboles. Es decir, se selecciona un nodo de un padre de manera aleatoria. Una vez seleccionado el punto de corte en el primer padre, se hace una elección aleatoria dirigida en el segundo padre para verificar restricciones como:

- Cruzar con un subárbol que evalúe a un tipo de dato compatible
- Cruzar con un subárbol de tamaño adecuado

El cruce es un operador muy disruptivo: Los árboles tienden a crecer indefinidamente si no se limita el tamaño de la operación de cruce.

El operador de cruce en programación genética de dos individuos iguales no genera dos descendientes iguales como en los algoritmos genéticos. Esto provoca:

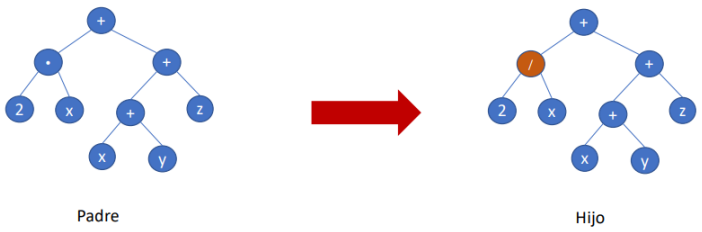
- Esto provoca que el cruce de dos árboles muy parecidos con buen fitness (cuando el algoritmo está próximo a converger) no necesariamente produzca descendientes adecuados.
- Así, el algoritmo explora bien el espacio pero no lo explota adecuadamente.



2.1.5 Operadores genéticos: Mutación

Existen distintas posibilidades, que producen una mayor o menor alteración en el descendiente:

- Mutación en un punto: Se escoge un nodo y se cambia su valor por otro del mismo tipo.
- Mutación por subárbol aleatorio: Se escoge una arista y se substituye el subárbol conectado a ella por otro generado aleatoriamente.

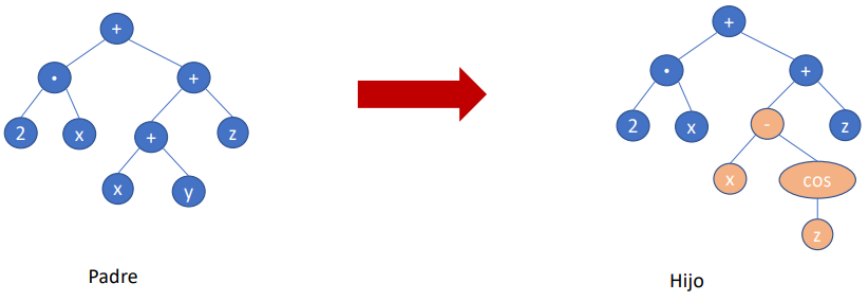


Tiene menor importancia que en AGs puesto que el

operador de cruce PG se basta para mantener la diversidad. Se define una lista de terminales que pueden intercambiarse entre sí y se reemplaza un nodo escogido al azar por otro de la lista: variable por variable, función por función, constante por constante, etc.

Mutación por Subárbol Aleatorio

Se escoge una arista aleatoria y se reemplaza el subárbol que cuelga de ella por otro generado aleatoriamente, sin superar el tamaño máximo.



Otros operadores de mutación

- **Permutación**: Se reemplaza un subárbol (lista de argumentos a un nodo función) por una permutación de estos.
- **Edición**: Simplifica una expresión reemplazándola por otra más sencilla.
- **Encapsulación**: Consiste en dar un nombre a un subárbol, convirtiéndolo en una nueva función que puede intervenir en otros árboles

2.2 Strongly-Typed Genetic Programming

El problema que tenía la GP canónica es que no permite la combinación de tipos de datos. Se pueden establecer simplificaciones (por ejemplo, 1 y 0 para los valores lógicos) pero, en general, se complica la resolución de algunos problemas. Tampoco se permite la sobrecarga de operadores. David Montana (1994) propone STGP para solucionar los problemas anteriores.

2.2.1 Fundamentos

Los individuos en STGP son similares a los individuos en CGP, es decir, son árboles de instrucciones. Los nodos definidos en STGP están tipados:

- Los argumentos que reciben (hijos) han de ser de un determinado tipo.
- El valor que devuelven es de un tipo determinado (no tiene por qué ser el mismo que el de los argumentos de entrada)
- Los nodos función admiten sobrecarga, es decir, un mismo nodo puede devolver un resultado distinto según los nodos sean de un tipo o de otro.

- La clave de STGP está en la tabla de compatibilidades que se establece al definir los nodos función y los nodos terminales. Esta tabla contiene:
    - Número de argumentos
    - Tipo de dato para cada uno de los argumentos
    - Tipo devuelto
- A partir de esa tabla, se puede construir una segunda en la que se incluyen los nodos que pueden actuar como hijos de cada uno de los nodos existentes. Esto es lo que permite manipular los árboles para que siempre resulten correctos.

2.2.2 Creación de árboles y operadores genéticos

El proceso de creación de árboles en STGP es muy parecido al de CGP. La única diferencia es que, los nodos se eligen al azar de entre los que son considerados como compatibles de un nodo dado. El cruce y mutación son exactamente iguales, con una pequeña variación:

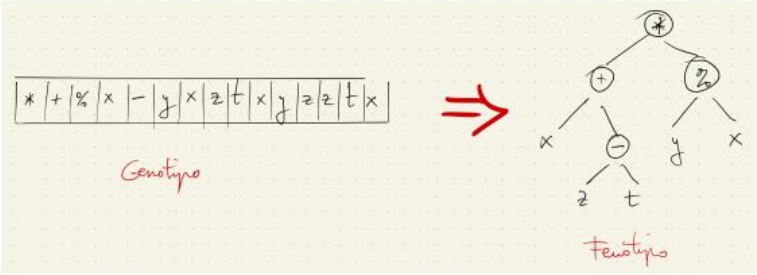
- Para el **cruce**, se eligen ramas cuyos tipos devueltos sean iguales o compatibles con el lugar de destino.
- Para la **mutación**, se cambian unas funciones por otras con el mismo prototipo y, cuando hay que generar un subárbol se hace usando la tabla de compatibilidad de funciones.

2.3 Gene Expression Programming (GEP)

2.3.1 Introducción

Paradigma propuesto por Candida Ferreira (2002). Presenta una doble codificación:

- Genotipo lineal (parecido a los individuos de AG con codificación entera)
- Fenotipo árbol de instrucciones (ejecutable y evaluable)



2.3.2 Codificación

Los cromosomas en GEP presentan dos zonas

- **Cabeza** (head) – Desde el extremo izquierdo hasta la **posición h**
- **Cola** (tail) – Desde la **posición h+1 hasta la posición h+t**

Los elementos de la cola sólo pueden ser terminales, los demás pueden ser funciones o terminales. Los valores h y t se denominan tamaño de cabeza y tamaño de cola, y se definen para garantizar que la traducción del genotipo al fenotipo producirá siempre un árbol de instrucciones válido.

$$t = h\,(n_{max} - 1) + 1$$

2.3.3 Decodificación

Para pasar del genotipo al fenotipo se va recorriendo el cromosoma lineal, y con su contenido se van rellenando los nodos del árbol por capas: primero la raíz, segundo la capa bajo la raíz y así hasta completar el árbol.

2.3.4 Operadores genéticos

Existen **múltiples operadores genéticos** en GEP: replicación, recombinación, mutación, transposición e inversión. Todos los operadores **se aplican con una determinada probabilidad** durante la fase de reproducción.

3. Representaciones híbridas en GP: Modelo GA-P