Trabajo Final
Memoria
Teoría de autómatas y lenguajes formales
Universidad de Cádiz

June 1, 2021

Alumnos: Alejandro Serrano Fernández
Pedro Antonio Navas Luque

1 Lenguaje

Para el diseño del lenguaje hemos decidido hacerlo como si de una lista de componentes se tratara, es decir, cada algoritmo genético irá acompañado de los elementos que lo forman con sus respectivos atributos o especificaciones.

De la misma manera declararemos la política de intercambio que sera común para todos los algoritmos genéticos declarados anteriormente.

Además de que hemos decidido hacerlo en inglés ya para que la terminología sea más fácil de buscar, debido a que los términos de los algoritmos genéticos también están en inglés.

Un ejemplo de declaración de un algoritmo genético junto a la política de intercambios sería:

```
BEGIN
AG
[
        CROSSOVER 10 | 90;
        MUTATION BITFLIP 20;
        REPLACEMENT WORST;
        SELECTION TOURNAMENT 5;
        POBLATION 1000;
        FITNESS (100-7x);
]
END

POLICY_BEGIN
[
        GENERATIONS 10;
        INTERCHANGE 20;
        SEND BEST;
        RECEIVE WORST;
]
POLICY_END
```

El lenguaje viene explicado más específicamente en el documento de manual de usuario.

2 Gramática

Para la construcción de la gramática hemos seleccionado las palabras clave necesarias para la configuración de un algoritmo genético y como deben interactuar entre ellos ("política" de intercambio). Además de algunos símbolos o identifi-

cadores, tales como paréntesis, punto y coma, números y "| ".

ID, EQUATION, BEGIN, END, POLICY_BEGIN, POLICY_END, POBLATION, FITNESS, SELECTION, CROSSOVER, MUTATION, REPLACEMENT, INTERCHANGE, SEND, RECEIVE, GENERATIONS, TOURNAMENT, ROULETTE, RANKING, BITFLIP, POLYNOMIAL, WORST, RANDOM, BEST, NUMBER, LPAREN, RPAREN, PTOCOMA, AND.

Una vez escogido los tokens necesarios implementamos las reglas de la gramática, en las cuales hemos utilizado la factorización para evitar ambigüedades.

2.1 Reglas

- S' − > total
- total − > BEGIN typebegin
- typebegin − > expr END POLICY_BEGIN LPAREN rules RPAREN POLICY_END
- typebegin − > END
- expr − > term
- expr − > expr term
- term − > ID LPAREN caracteristicas RPAREN
- rules − > polytypes rules − > rules polytypes caracteristicas − > sentence
- caracteristicas − > caracteristicas sentence
- polytypes − > RECEIVE typereceive PTOCOMA
- polytypes − > SEND typesend PTOCOMA
- polytypes − > INTERCHANGE NUMBER PTOCOMA
- polytypes − > GENERATIONS NUMBER PTOCOMA
- sentence − > FITNESS EQUATION PTOCOMA
- sentence − > SELECTION typeselection PTOCOMA
- sentence − > CROSSOVER NUMBER typecross
- sentence − > REPLACEMENT typereplace PTOCOMA
- sentence − > MUTATION typemutation PTOCOMA
- sentence − > POBLATION NUMBER PTOCOMA
- typemutation − > POLYNOMIAL NUMBER

- typemutation $->$ BITFLIP NUMBER
- typeselection $->$ RANKING
- typeselection $->$ ROULETTE
- typeselection $->$ TOURNAMENT NUMBER
- typereplace $->$ RANDOM
- typereplace $->$ WORST
- typecross $->$ AND NUMBER PTOCOMA
- typecross $->$ PTOCOMA
- typesend $->$ RANDOM
- typesend $->$ BEST
- typereceive $->$ RANDOM
- typereceive $->$ WORST

Para clarificar la gramática hemos hecho uso de términos con nombres intuitivos, como typemutation que esta ligado al tipo de mutación del algoritmo. Además estos términos son resultado de factorizar y evitan ambigüedades por ejemplo:

Reglas originales:
sentence $->$ MUTATION POLYNOMIAL NUMBER PTOCOMA
sentence $->$ MUTATION BITFLIP NUMBER PTOCOMA

Resultado de factorizar:
sentence $->$ MUTATION typemutation PTOCOMA
typemutation $->$ POLYNOMIAL NUMBER
typemutation $->$ BITFLIP NUMBER

Con todas las reglas definidas en el Parser, utilizamos un árbol binario para interpretar la entrada de los datos. Para ver su funcionamiento deberemos observar la parte correspondiente del código, que esta debidamente comentada.

3 Prueba del programa

Dada una entrada:

```
BEGIN
    First
    [
        POBLATION 100;
        CROSSOVER 50 | 70;
        MUTATION BITFLIP 40;
        REPLACEMENT RANDOM;
        SELECTION TOURNAMENT 7;
        FITNESS (7x+2);
    ]
    Second
    [
        REPLACEMENT WORST;
        CROSSOVER 50 | 70;
        POBLATION 1000;
        MUTATION BITFLIP 40;
        SELECTION TOURNAMENT 7;
        FITNESS (654894/5x*5+98);
    ]
    Third
    [
        CROSSOVER 10 | 90;
        MUTATION BITFLIP 20;
        REPLACEMENT WORST;
        SELECTION TOURNAMENT 5;
        POBLATION 1000;
        FITNESS (100-7x);
    ]
END

POLICY_BEGIN
[
    GENERATIONS 10;
    INTERCHANGE 20;
    SEND BEST;
    RECEIVE WORST;
]
POLICY_END
```

Figure 1: Contenido fichero "code.txt"

```
>Variable  First  con las siguientes caracteristicas:
  * Población con 100 elementos
  * Crossover con punto de corte en 50 y en 70
  * Mutacion bitflip con 40 elementos
  * Reemplazo (mejor)
  * Seleccion por torneo con 7 elementos
  * Funcion FITNESS: (7x+2)

>Variable  Second  con las siguientes caracteristicas:
  * Reemplazo (peor)
  * Crossover con punto de corte en 50 y en 70
  * Población con 1000 elementos
  * Mutacion bitflip con 40 elementos
  * Seleccion por torneo con 7 elementos
  * Funcion FITNESS: (654894/5x*5+98)

>Variable  Third  con las siguientes caracteristicas:
  * Crossover con punto de corte en 10 y en 90
  * Mutacion bitflip con 20 elementos
  * Reemplazo (peor)
  * Seleccion por torneo con 5 elementos
  * Población con 1000 elementos
  * Funcion FITNESS: (100-7x)


    *****   POLITICAS DE INTERCAMBIOS   *****

  * Intercambios cada 10 generaciones
  * Intercambios de 20 elementos
  * Enviando mejores individuos
  * Reeemplaza recibidos por sus peores individuos
```

Figure 2: Salida del programa

Ahora quitamos uno de los términos de la configuración de un algoritmo genético:

```
BEGIN
    First
    [
        POBLATION 100;
        CROSSOVER 50 | 70;
        MUTATION BITFLIP 40;
        REPLACEMENT RANDOM;
        SELECTION TOURNAMENT 7;
        FITNESS (7x+2);
    ]
    Second
    [
        REPLACEMENT WORST;
        CROSSOVER 50 | 70;
        POBLATION 1000;
        MUTATION BITFLIP 40;
        SELECTION TOURNAMENT 7;
        FITNESS (654894/5x*5+98);
    ]
    Third
    [
        CROSSOVER 10 | 90;
        MUTATION BITFLIP 20;
        REPLACEMENT WORST;
        SELECTION TOURNAMENT 5;
        POBLATION 1000;
        FITNESS (100-7x);
    ]
END

POLICY_BEGIN
[
    GENERATIONS 10;
    INTERCHANGE 20;
    SEND BEST;
    RECEIVE WORST;
]
POLICY_END
```

Figure 3: Misma entrada de datos pero sin la instrucción señalada

```
Traceback (most recent call last):
  File "c:\Users\Ale2\Desktop\Alejandro\TALF\Practica GLOBAL\ParserGENETIC.py", line 429, in <module>
    interpreter(tree)
  File "c:\Users\Ale2\Desktop\Alejandro\TALF\Practica GLOBAL\ParserGENETIC.py", line 246, in __init__
    resultado = self.avanzarArbol(tree)
  File "c:\Users\Ale2\Desktop\Alejandro\TALF\Practica GLOBAL\ParserGENETIC.py", line 275, in avanzarArbol
    self.avanzarArbol(node[1])
  File "c:\Users\Ale2\Desktop\Alejandro\TALF\Practica GLOBAL\ParserGENETIC.py", line 269, in avanzarArbol
    self.avanzarArbol(node[1])
  File "c:\Users\Ale2\Desktop\Alejandro\TALF\Practica GLOBAL\ParserGENETIC.py", line 270, in avanzarArbol
    self.avanzarArbol(node[2])
  File "c:\Users\Ale2\Desktop\Alejandro\TALF\Practica GLOBAL\ParserGENETIC.py", line 297, in avanzarArbol
    raise ValueError("ERROR: Falta caracteristica %s en variable %s "%(self.sentenceNames[findError[0]],node[1].fElement))
ValueError: ERROR: Falta caracteristica POBLATION en variable Second
```

Figure 4: Salida del programa de la entrada errónea