

Reparación automática de circuitos reconfigurables

Brayan Nicolás Pérez Cobos e

Universidad de Sevilla

Isabela Ferreira do Nascimento

brapercob@alum.us.es -
brayanpercob@gmail.com &

Dpto. Ciencias de la Computación e Inteligencia
Artificial

isaferdo@alum.us.es - isabela.faxinha@gmail.com

Profesor: Agustín Riscos Núñez

Resumen: El objetivo de este trabajo es implementar un método en Python y algoritmos genéticos donde el circuito pueda auto-repararse si encontrara algún daño en su hardware y así reconfigurarse para volver a comportarse igual que anteriormente.

Palabras claves: circuito, algoritmo genético, cruzamiento, mutación,

Abstract: The main goal of this work is the implementation using Python and genetics algorithms of a function where the circuit could be self-repair if it finds any damage in the hardware before coming back to its original state. *Key words:* circuit, genetic algorithm, crossover, mutation,

I. INTRODUCCIÓN:

Como sabemos, en la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos básicos, como agua, comida, refugio, etc. Destacándose así los mejores preparados o con mayores habilidades.

Este fenómeno se llama Selección Natural postulada por Charles Darwin en 1859 en su libro *El origen de las especies*.

Los Algoritmos Genéticos usan una analogía con el comportamiento natural. Trabajamos con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor o puntuación, relacionado con lo bueno que sea esa solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos.

Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma.

Este cruce producirá nuevos individuos (descendientes de los anteriores) los cuales comparten algunas de las características de sus padres.

Los Algoritmos Genéticos son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Siempre basándose en el proceso genético de los organismos vivos, la selección natural y la supervivencia de los más fuertes.

Los principios básicos de los Algoritmos Genéticos fueron establecidos por Holland (1975, Goldberg (1989), Davis (1991), Michalewicz (1992) y Reeves (1993) –

Los dos tipos de selección principales son mutación y cruce, dentro de los cuales hay más subtipos.

II. PRELIMINARES

Para hablar de Algoritmo Genético tenemos que saber representar cualquier problema planteado.

Para dicha representación necesitaremos definir una población, eligiendo correctamente su tamaño y quienes serían su conjunto de individuos.

Decidimos también como será cada individuo, su secuencia de genes, la longitud de dicho individuo y como se representaría la solución.

Establecemos que operador de variación usaremos, *cruzamiento*, obteniendo dos hijos a partir de dos padres o *mutación* resultando un nuevo individuo a partir de uno ya existente.

Definimos la función fitness evaluando la calidad de un fenotipo (solución candidata) y el objetivo general.

En el caso de nuestro trabajo la población sería el conjunto de todos los circuitos y cada circuito, $m \times n$, sería equivalente a un único individuo.

El individuo tiene una serie de números enteros que cada cual representaría un término necesario para la resolución del problema.

Por lo tanto, para nuestro dominio los genes vendrían representados por cada uno de los valores enteros, e interpretándolos de manera que sean tipos de puerta o bien entradas.

Y finalmente nuestro operador de variación será con un método de cruzamiento con dos puntos, se eligen aleatoriamente los puntos intermedios, las puertas en nuestro caso, y se intercambian los genes de ambos en ese punto.

Alternativamente tendremos la opción de usar cruzamiento con un único punto y a partir de ese

punto se cambiaría los genes del individuo o la opción de cruzamiento uniforme.

En la opción de cruzamiento uniforme todos los genes, las puertas, tienen la misma probabilidad de intercambio ya que se cambia cada par aleatoriamente.

Podríamos elegir, también dentro de las opciones que hemos hecho para poder completar nuestro algoritmo genético, estaría la selección elitista o por torneo.

En la selección elitista siempre se quedaría con los circuitos que tuviera menos defectos, eligiendo así la fitness más cercana a la óptima.

En la selección por torneo se hace tantos torneos como individuos haya para seleccionar.

III. METODOLOGÍA

En primer lugar para la representación de nuestro dominio comenzamos por plantearnos la definición de clases objeto, para trabajar más fácilmente con los atributos, pero finalmente nos decantamos por una matriz $m \times n$ de vectores donde cada uno de ellos representa una puerta lógica, siguiendo la estructura (N.º de puerta, tipo de puerta, capa de la puerta, [conexion1, conexion2]), siendo esto último el par de conexiones de la forma (puerta que envía señal, puerta que recibe señal).

Durante la creación de los circuitos se tienen en cuentas las pautas proporcionadas, delimitando las conexiones entre puertas a las 2 capas anteriores y evitando así incongruencias en los circuitos.

Con relación al algoritmo que utilizaremos para la auto reparación de circuitos, decidimos hacer uso del método de algoritmo genético simple. Representado en el siguiente fragmento de pseudocódigo.

inicializar y evaluar una población de tamaño *tam_pob*

repetir

seleccionar con
reemplazamiento
tam_pob individuos de
población

emparejar los individuos
seleccionados

prob_c, cruzar cada
pareja

prob_m, mutar cada
hijo

reemplazar población con los
tam_pob individuos
obtenidos

evaluar la nueva población

hasta que se alcance la generación
máx_gen

En general este algoritmo comienza por evaluar una población proporcionada y selecciona un conjunto y aplica operadores de cruzamiento/mutación reemplazando los nuevos individuos y reevaluando la población, repitiendo este proceso hasta un número de generaciones determinado.




Para realizarse el reemplazo de los individuos para cada población se hace estrategias de reemplazo tales como aleatorio, basados en la edad o en la función de aptitud.



En nuestro caso se usa la estrategia de reemplazo por la aptitud ya que siempre tendrán mayor probabilidad de ser reemplazados los individuos peores, en nuestro caso los circuitos, con más defecto tendrán mayor probabilidad de ser reemplazado.

Lo primero que hicimos fue la implementación de la estructura de datos ya que se nos exigía dicha implementación en el objetivo

específico número 1, para poder crear así el circuito rectangular de $m \times n$ puertas lógicas.

Hacemos la definición de cada tipo de puerta, siendo las opciones para el trabajo *OR*, *AND*, *NOT*, *NAND* y *XOR*

NOT	AND	NAND																																				
\overline{A}	AB	\overline{AB}																																				
																																						
<table><tr><th>A</th><th>X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0
A	X																																					
0	1																																					
1	0																																					
B	A	X																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				
B	A	X																																				
0	0	1																																				
0	1	1																																				
1	0	1																																				
1	1	0																																				



OR		XOR																													
$A + B$		$A \oplus B$																													
																															
<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0
B	A	X																													
0	0	0																													
0	1	1																													
1	0	1																													
1	1	1																													
B	A	X																													
0	0	0																													
0	1	1																													
1	0	1																													
1	1	0																													

Creamos un método para satisfacer el objetivo número 2 donde se nos pedía que pudiéramos a partir de una entrada de n bits devolver el vector de salida con alguna marca de que se tratara de un defectuoso en dicho caso, creamos la opción de *puertas_defectuosas* y *conexiones_defectuosas*.

Creamos un método, *auto_diagnostico*, para el objetivo específico número 3 donde comparamos si las salidas esperadas coinciden con las salidas obtenidas al realizarse una nueva combinación de individuos.

Para el objetivo específico número 4 el algoritmo genético debe, sin conocer las puertas, trabajar buscando buenos circuitos. Utilizamos la función *creaGenes* para crear nuevos circuitos.

Finalmente decidimos hacer el apartado de mejoras añadiendo 2 puertas lógicas más, *NOR* y *XNOR*, como el ejemplo (b) que sugería el profesor.

NOR	XNOR																														
$\overline{A + B}$	$\overline{A \oplus B}$																														
																															
<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
B	A	X																													
0	0	1																													
0	1	0																													
1	0	0																													
1	1	0																													
B	A	X																													
0	0	1																													
0	1	0																													
1	0	0																													
1	1	1																													

IV. RESULTADOS

gen	nevals	mínimo	media	máx.
0	20	8	12.55	16
1	12	8	10.9	16
2	17	6	8.2	12
3	12	6	7.8	10
4	13	4	7.1	12
5	16	4	6.3	10
6	14	4	6.5	12
7	10	4	4.2	8
8	12	4	4.2	8
9	15	4	5.1	16
10	12	4	5.7	12
11	11	4	4.3	8
12	16	4	4.5	12
13	17	2	5	12
14	14	2	3.9	8
15	10	2	3.8	12
16	13	2	3.3	12
17	16	2	2.4	6
18	11	2	2.9	12
19	14	2	2.6	8
20	12	2	3.8	12

figura 1

En los resultados estadísticos obtenidos para una prueba concreta, en la figura 1 vemos representado en la columna gen las poblaciones desde la primera hasta la generación 20, en la columna nevals obtenemos cuantos individuos nuevos que calcula nuestra función fitness. En la columna mínimo nos dice los mínimos individuos malos, nuestras puertas con errores y en máximo la cantidad máxima de errores que nos podría salir, y la media, su media correspondiente.

La mejor solución encontrada ha sido:

```
-----
Individuo: [(1, 'XOR', 1, [(0, 1), (0, 1)]),
(2, 'AND', 1, [(0, 2), (0, 2)]), (3, 'NOT', 1,
[(0, 3), (0, 3)]), (4, 'AND', 2, [(1, 4), (1, 4)]),
(5, 'AND', 2, [(3, 5), (3, 5)]), (6, 'AND', 2,
[(2, 6), (3, 6)]), (7, 'XOR', 3, [(6, 7), (0, 7)]),
(8, 'OR', 3, [(2, 8), (4, 8)]), (9, 'XOR', 3,
[(4, 9), (5, 9)]]; Fitness: 2.0
```

figura 2

Como se ven en la figura 2 nos devuelve el mejor resultado con la función fitness, para un circuito 3x3, donde nos muestra como se hizo esa combinación, en el caso señalado, como se puede contemplar en la figura 3, el primer número, 8, corresponde a la puerta que estamos viendo, esta puerta sería una puerta lógica de tipo OR y se encontraría en la capa 3. Dentro de los corchetes, estamos viendo que en la entrada 1 de la puerta 8 entra la salida de la puerta 2, en la entrada 2 de la puerta 8 entraría el resultado de la puerta 4.

La mejor solución encontrada ha sido:

```
-----
Individuo: [(1, 'XOR', 1, [(0, 1), (0, 1)]),
(2, 'AND', 1, [(0, 2), (0, 2)]), (3, 'NOT', 1,
[(0, 3), (0, 3)]), (4, 'AND', 2, [(1, 4), (1, 4)]),
(5, 'AND', 2, [(3, 5), (3, 5)]), (6, 'AND', 2,
[(2, 6), (3, 6)]), (7, 'XOR', 3, [(6, 7), (0, 7)]),
(8, 'OR', 3, [(2, 8), (4, 8)]), (9, 'XOR', 3,
[(4, 9), (5, 9)]]; Fitness: 2.0
```

figura 3

Para ver los errores miramos en nuestro circuito solución y comparamos cuantos resultados distan del correcto en la figura 4.

Este es nuestro circuito solución:

```
[(1, 'NOT', 1, [(0, 1), (0, 1)]), (2, 'NOT',
1, [(0, 2), (0, 2)]), (3, 'NAND', 1, [(0, 3),
(0, 3)]), (4, 'AND', 2, [(3, 4), (0, 4)]),
(5, 'OR', 2, [(3, 5), (3, 5)]), (6, 'AND',
2, [(0, 6), (0, 6)]), (7, 'XOR', 3, [(3, 7),
(0, 7)]), (8, 'NOT', 3, [(0, 8),
(2, 8)]), (9, 'OR', 3, [(6, 9), (5, 9)])]
```

figura 4

Estas son las tuplas de salida de los circuitos.

```
1 0, 1] 0 0, 1]
[0, 0, 0] [0, 0, 0]
[1, 1, 1] [1, 1, 1]
[0, 1, 0] [0, 1, 0]
1 0, 1] 0 0, 1]
[0, 0, 0] [0, 0, 0]
[1, 1, 1] [1, 1, 1]
[0, 1, 0] [0, 1, 0]
```

Hemos obtenido un circuito alternativo con 2.0 errores respecto al circuito solución

Comparamos que solo se ven dos errores o dos puertas que no coinciden con el resultado esperado.

Por lo tanto, el rendimiento de nuestro algoritmo es del 91.7%.

Por último, realizamos un estudio comparativo entre la ejecución del algoritmo para diferentes circuitos, operadores de cruzamiento, estrategias de selección, número de individuos que conforman una población y generaciones de aplicación del algoritmo.

Realizamos pruebas sobre circuitos generados por nuestro propio método creaCircuitos(), de un tamaño 3*3, 4*3 y 5*4.

Presentamos ahora tablas con los resultados obtenidos:

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
3*3	2 Puntos	Torneo	20	20	0
3*3	1 Punto	Torneo	20	20	0
3*3	Uniforme	Torneo	20	20	0
3*3	2 Puntos	Elitista	20	20	0
3*3	1 Punto	Elitista	20	20	0
3*3	Uniforme	Elitista	20	20	4

En este primer caso para un circuito 3*3 aplicando los diferentes operadores y estrategias, además de distintos valores para población y generaciones alternando entre 20 y 40.

En esta primera tabla vemos que el algoritmo se comporta de manera muy efectiva generando alternativas con una función fitness óptima, a excepción del cruzamiento uniforme selección elitista que como veremos más adelante no suele arrojar muy buenos resultados con la selección elitista. En general la fitness se reducía de manera

constante a medida que pasaban las generaciones.

Salvo en el caso de la selección elitista donde la evolución se producía de manera poco constante apareciendo solo un circuito donde la función fitness resultara 0.

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
3*3	2 Puntos	Torneo	40	20	0
3*3	1 Punto	Torneo	40	20	0
3*3	Uniforme	Torneo	40	20	0
3*3	2 Puntos	Elitista	40	20	0
3*3	1 Punto	Elitista	40	20	0
3*3	Uniforme	Elitista	40	20	0

En este se experimento se aumenta el número de individuos por población, obteniendo resultados mejores que en el caso anterior, sobre todo en cruzamiento uniforme. Aunque de nuevo se produce la poca constancia de los resultados en la selección elitista.

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
3*3	2 Puntos	Torneo	20	40	0
3*3	1 Punto	Torneo	20	40	0
3*3	Uniforme	Torneo	20	40	0
3*3	2 Puntos	Elitista	20	40	0
3*3	1 Punto	Elitista	20	40	0
3*3	Uniforme	Elitista	20	40	4

Por último, se vuelve a realizar el experimento con un límite de generación establecido en 40, de nuevo se repiten los resultados, en el caso de la selección por torneo el individuo con fitness se encuentra rápidamente a las pocas generaciones, por otro lado, la selección elitista vuelve a ser poco constante. Llegando a no encontrarse el circuito alternativo completo en el cruzamiento uniforme.

Pasamos ahora con el siguiente circuito a examinar, esta vez de tamaño 4*3, incluimos las tablas de información:

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
4*3	2 Puntos	Torneo	20	20	2
4*3	1 Punto	Torneo	20	20	4
4*3	Uniforme	Torneo	20	20	0
4*3	2 Puntos	Elitista	20	20	4
4*3	1 Punto	Elitista	20	20	4
4*3	Uniforme	Elitista	20	20	4

En una primera tanda de pruebas, se hace presente que el aumento de tamaño afecta negativamente en el resultado del algoritmo, solo el cruzamiento uniforme consigue encontrar un circuito bueno, como en el circuito anterior se repite la poca constancia en la selección elitista

contrastando con la rapidez de la selección por torneo a la hora de reducir la función fitness al ir sucediéndose las generaciones.

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
4*3	2 Puntos	Torneo	40	20	0
4*3	1 Punto	Torneo	40	20	2
4*3	Uniforme	Torneo	40	20	2
4*3	2 Puntos	Elitista	40	20	2
4*3	1 Punto	Elitista	40	20	4
4*3	Uniforme	Elitista	40	20	2

Aumentado el número de individuos por población obtenemos resultados mejores en la mayoría de los casos excepto, sorprendentemente en el tercer caso que pasa de encontrar un circuito fitness 0 a uno con 2 errores en la salida respecto a la colección de pares referencia.

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
4*3	2 Puntos	Torneo	20	40	2
4*3	1 Punto	Torneo	20	40	2
4*3	Uniforme	Torneo	20	40	0
4*3	2 Puntos	Elitista	20	40	4
4*3	1 Punto	Elitista	20	40	4
4*3	Uniforme	Elitista	20	40	4

Finalmente, aumentado las generaciones conseguimos un resultado parecido al primer caso, a excepción de el cruzamiento en 1 punto con selección por torneo que consigue mejorar 2 dos errores.

Como conclusión se hace evidente que al aumentar número de variables que influyen en la salida el resultado se ve afectado negativamente, vamos a comprobar esto con el último experimento, utilizando un circuito con tamaño 5*4:

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
5*4	2 Puntos	Torneo	20	20	16
5*4	1 Punto	Torneo	20	20	12
5*4	Uniforme	Torneo	20	20	8
5*4	2 Puntos	Elitista	20	20	8
5*4	1 Punto	Elitista	20	20	16
5*4	Uniforme	Elitista	20	20	8

Como podemos observar el resultado que arroja ahora el algoritmo ha empeorado significativamente, en ambas estrategias de selección el número de errores ha aumentad en más del doble respecto al ejemplo anterior pasando de 12 puertas a 20.

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
5*4	2 Puntos	Torneo	40	20	2
5*4	1 Punto	Torneo	40	20	0
5*4	Uniforme	Torneo	40	20	0
5*4	2 Puntos	Elitista	40	20	4
5*4	1 Punto	Elitista	40	20	4
5*4	Uniforme	Elitista	40	20	4

Aumentado el número de individuos por población los resultados mejoran en gran medida, llegando incluso a obtenerse circuitos alternativos perfectamente funcionales. Aunque estos resultados pueden no ser muy representativos ya que la constancia en los resultados de cada generación no es muy alta, obteniendo ese resultado en únicamente una de ellas.

Por otro lado, en la selección elitista, aunque se aprecian mejoras, estos son muy poco constantes variando los resultados obtenidos en las generaciones entre los 4 y 16 errores como mínimo.

Circuito	Cruzamiento	Selección	Población(Indiv)	Generación	Fitness (Mejor)
5*4	2 Puntos	Torneo	20	40	8
5*4	1 Punto	Torneo	20	40	4
5*4	Uniforme	Torneo	20	40	0
5*4	2 Puntos	Elitista	20	40	8
5*4	1 Punto	Elitista	20	40	8
5*4	Uniforme	Elitista	20	40	8

Para finalizar la fase de experimentación aumentamos el valor de las generaciones y obtenemos unos resultados que, aunque mejoran lo obtenido en el primer caso no son mejores que al aumentar la población de circuito por generación.

V. CONCLUSIONES

Teniendo en cuenta la fase de experimentación por la que hemos pasado y viendo los resultados obtenidos podemos concluir que el tamaño de los circuitos que se intentan reconstruir afecta en gran medida a la efectividad del algoritmo genético simple que se ha implantado, respecto a las estrategias de selección que se han introducido, podemos decir que, de forma general la selección por torneo de 2 participantes ha obtenido mejores, o al menos iguales, resultados que la selección elitista.

Por otro lado, sobre los operadores de cruzamiento que se han introducido en el proyecto podemos decir que el cruzamiento por recombinación uniforme es el que mejores resultados ha obtenido en los casos en los que se

ha utilizado con selección por torneo, ya que con elitista es de los que peores resultados ha obtenido

Entre recombinación en un punto y dos puntos no hay un claro operador mejor ya que se han obtenido resultados dispares, en algunos casos se han obtenido mejores resultados con dos puntos y viceversa.

Para concluir, aunque se haya intentado realizar una etapa de experimentación bastante representativa, podemos ver que el factor aleatorio juega un gran papel en los resultados de los algoritmos genéticos.

VI. REFERENCIAS

<http://sabia.tic.udc.es/mgestal/cv/AAGGtutoria/node4.html>

<https://agorafilia2012.wordpress.com/2017/01/22/puertas-logicas/>

<http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf>

<https://www.cs.us.es/cursos/iais-2017/temas/Geneticos.pdf>

https://www.researchgate.net/publication/255599933_Logic_Circuits_Synthesis_Through_Genetic_Algorithms - Estructura de datos y definición de genes para el dominio de los circuitos lógicos.