REPARACIÓN AUTOMÁTICA DE CIRCUITOS MEDIANTE ALGORITMO GENÉTICO

Angel Mármol Fernández
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
angmarfer3@alum.us.es
angelmariamarmolfernandez@gmail.com

Luis Rus Pegalajar
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
luiruspeg@alum.us.es
lrp_ruspeg@hotmail.com

***Resumen*— El objetivo de este trabajo es explicar cómo se ha llevado a cabo la implementación de un algoritmo genético capaz de hacer un diagnóstico de diferentes circuitos y que éstos se reconfiguren automáticamente en el caso de existir un fallo. Este algoritmo permite la auto reparación de circuitos dada una colección de vectores de entrada, calculando los vectores de salida ideales y comparándolos con los obtenidos en un determinado momento.**

Con la intención de comparar las diferentes posibilidades en cuanto a entradas, tamaño del circuito, número de generaciones y tamaño de la población, el trabajo se complementará con un análisis en el que se expondrán los resultados obtenidos.

A partir de estos resultados se concluye que la probabilidad de cruzamiento y mutación tiene más impacto a la hora de encontrar soluciones óptimas, siendo más eficiente cuanto menor sean.

Palabras Clave*— *Algoritmo Genético, Circuitos, Automáticos, Reconfigurables, Diagnosis, Reparación

I. INTRODUCCIÓN

La computación reconfigurable tiene sus inicios en 1960 cuando Gerald Estrin propuso un concepto en el que el procesador de un ordenador controlaba el funcionamiento de un hardware reconfigurable. El procesador podía cambiar como se comportaba el hardware para poder realizar diferentes tipos de tareas diferentes sin la necesidad de cambiar dicho hardware, incluso llegaba a ser tan rápido ejecutando estas tareas como el hardware específico para ellas [1].

Actualmente existen diferentes tipos de dispositivos en lo que llamamos la Lógica programada; un diseño implementando chips que permite la reconfiguración de los circuitos mediante el software que incorporan, al contrario que la lógica cableada. La lógica programada se basa en dispositivos lógicos programables (PLD) que no tienen una función establecida y que deben ser programados previamente para su uso. Los más usados son los CPLDs (Complex Programmable Logic Device) y las FPGAs (Field Programmable Gate Arrays) [2].

Si los circuitos electrónicos pudieran reconfigurarse por si solos para cumplir otras funciones, o en el caso de estar dañados para poder seguir funcionando correctamente, se podría hablar de robots u otras aplicaciones autosuficientes en el sentido de que podrían adaptarse a cualquier situación sin la necesidad de una persona vigilando su funcionamiento.

Por ello, las investigaciones acerca de este tema están adquiriendo cada vez más importancia, ya que existen numerosas ocasiones en las que es necesario que el propio sistema sea capaz de auto reconfigurarse al encontrarse en una situación peligrosa o de difícil acceso para el ser humano como pueden ser las operaciones llevadas a cabo en el espacio exterior.

Este trabajo tratará de ofrecer una visión sobre estos circuitos, en concreto serán circuitos que contarán con m capas y n puertas en cada capa. Estas puertas se pueden configurar de manera que pueden ser de diferentes tipos – i.e., AND y NOR. Las conexiones entre las puertas están bajo una serie de restricciones:

- Las entradas de una determinada puerta i que se encuentra en una capa j solo podrán estar conectadas a puertas de las capas j-1 y j-2
- Las puertas o conexiones defectuosas devolverán un 0
- Existen conexiones entre todas las puertas, mientras que cumplan las restricciones, pero sólo se usaran aquellas que el algoritmo vea oportunas.

El proyecto está enfocado en la auto reparación y diagnóstico de circuitos mediante su reconfiguración, ya sea cambiando el tipo de puerta o las conexiones entre ellas. Se propone una solución a como funcionarían estos circuitos por si solos en el caso de estar dañados y como de eficientes serían los resultados.

Nuestra intención es la de proveer un trabajo de investigación que ayude en la implementación y desarrollo de aplicaciones que hagan uso de estos circuitos auto reconfigurables, haciendo que el profesional pueda centrarse en otros aspectos como el de predecir cuando el circuito va a fallar, pudiendo mejorar aun más la eficiencia de estos.

La estructura del documento será la siguiente:

1) Preliminares

   En esta sección se describirán las técnicas empleadas y los trabajos relacionados con el nuestro.

2) Metodología

   En esta parte se explicará como se ha llevado a cabo el desarrollo del proyecto

3) Resultados

   En esta sección se expondrán los resultados de los experimentos

4) Conclusiones

   Por último, se indicarán las conclusiones obtenidas y como se podría mejorar el trabajo.

II. Preliminares.

A. *Métodos empleados*

Para llevar a cabo el trabajo se han utilizado una serie de librerías de Python: la implementación del algoritmo genético se ha hecho mediante las librerías DEAP y Numpy [3][4]. Para la representación de los circuitos como grafo, se han usado las librerías NetworkX y Matplotlib [5][6]. En cuanto a la interfaz gráfica, se ha realizado con la librería PyQt5 [7].

El algoritmo genético hará evolucionar a la población mediante mutaciones y cruces. Para la mutación, se reemplazarán los genes por un número aleatorio de 0 a 2*n-1 con probabilidad variable.

Un gen será un número aleatorio entre 0 y 2*n-1, siendo n el número de puertas en cada capa del circuito. Un individuo estará formado por 3*m*n genes y la decodificación de este se hará cogiendo los genes de 3 en 3, siendo los genes del 0 al 2 la puerta 0, del 3 al 5 la puerta 1 y así hasta la última puerta, la m*n-1.

Tanto el número de individuos de la población inicial como el número de generaciones son introducidos por el usuario, por lo que se hará una comparación de los resultados obtenidos dependiendo de estos valores.

El problema planteado será de minimización, ya que el algoritmo tratará de encontrar aquellos circuitos cuyas salidas difieran lo menos posible con las salidas ideales.

La representación de los circuitos mediante grafos hará más fácil su comprensión, pero siempre irán acompañadas de la representación en texto en el caso de existir ambigüedades.

B. *Trabajo Relacionado*

Como se comentó anteriormente, las investigaciones de Gerald Estrin y su equipo fueron muy importantes para el diseño y análisis de sistemas reconfigurables [8].

A veces el reconfigurar un circuito y buscar soluciones óptimas para su funcionamiento, puede tomar mucho tiempo, por ello hay artículos como el de Lukáš Sekanina donde se introducen métodos para acelerar este proceso [9].

III. Metodología

La solución descrita en este documento hace uso del lenguaje de programación Python, con librerías externas para el uso de una interfaz gráfica, la representación gráfica de circuitos y la búsqueda de distintas soluciones para reparar el circuito mediante algoritmo genético.

Las principales librerías usadas son las siguientes:

1) Interfaz gráfica: PyQt5
2) Representación del circuito: NetworkX
3) Algoritmo genético: DEAP

A continuación, detallamos en alto nivel la implementación de los distintos algoritmos usados, así como de la representación del circuito y del simulador de éste, partiendo de una entrada y devolviendo una salida.

Primero explicaremos los conceptos básicos de la estructura de nuestra solución:

1) La base de nuestro código es la clase Puerta, que tiene dos conexiones de entradas y una salida, así como el tipo de puerta lógica que representa. Hemos implementado 7 puertas lógicas diferentes: AND, OR, XOR, XNOR, NAND, NOR, NOT. Todas reciben señal de ambas conexiones entrantes de la puerta menos la NOT, para la cual sólo se tiene en cuenta la primera conexión entrante.

2) El circuito es formado por un conjunto indefinido de puertas, de tamaño m (capas) * n (puertas).

3) Aunque en un principio habíamos interpretado que tanto el circuito inicial como los circuitos solución recibían una entrada, finalmente hemos permitido que se reciban 10 entradas aleatorias. Así, un circuito se considera dañado si no consigue una salida igual a la ideal para cada entrada. El rendimiento de la solución es el número de pares entrada-salida para los cuales el circuito solución consigue simular al circuito ideal.

4) Para conseguir más opciones, el algoritmo genético devolverá las tres soluciones con mejor rendimiento de toda la población.

5) Un circuito tiene una representación gráfica y una representación textual.

   a) En la representación textual se indica explícitamente el índice de la puerta, el tipo de puerta que es, sus conexiones de entrada y su salida. Para diferenciar las

conexiones de una entrada respecto a la conexión de una puerta, se ha usado la siguiente representación:

a.1) Si se trata de una conexión de una entrada, se escribirá el índice de la entrada aumentado en una unidad y en negativo. Por ejemplo, una conexión con las entradas 0 y 3 se representará como una conexión con -1 y -4.

b.1) Si se trata de una conexión con una puerta, simplemente se escribirá su índice.

b) En la representación gráfica se pintan distintos nodos representando las puertas y entradas. Como indica la leyenda, si el nodo está en color azul significa que esa puerta o entrada tiene valor 1. Si está en rojo, significa que la puerta es defectuosa. Si existe una conexión entre dos nodos, significa que la puerta del nodo final toma como entrada la conexión del nodo inicial. Si dicha conexión está en color rosa, significa que es defectuosa.

6) La generación del circuito tiene un modo prueba que generará un circuito predefinido de tamaño 3x3, pero cabe la posibilidad de crear un circuito aleatorio del tamaño deseado. El circuito en modo prueba será el siguiente.

```
-----------------------------------------------------PUERTA NUMERO: 0
TIPO PUERTA: OR
CONEXIONES: [-1, -3]
SALIDA: 0

-----------------------------------------------------PUERTA NUMERO: 1
TIPO PUERTA: AND
CONEXIONES: [-2, -1]
SALIDA: 0

-----------------------------------------------------PUERTA NUMERO: 2
TIPO PUERTA: NOT
CONEXIONES: [-3, -2]
SALIDA: True

-----------------------------------------------------PUERTA NUMERO: 3
TIPO PUERTA: NAND
CONEXIONES: [-1, 1]
SALIDA: 0
```

Fig. 1. Circuito ideal por defecto de prueba (en modo textual).

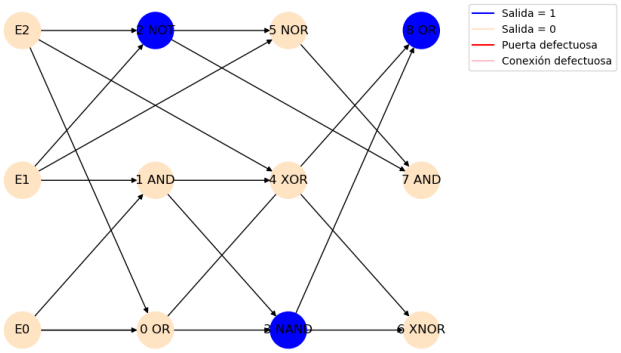

Fig. 2. Circuito ideal por defecto de prueba (en modo gráfico).

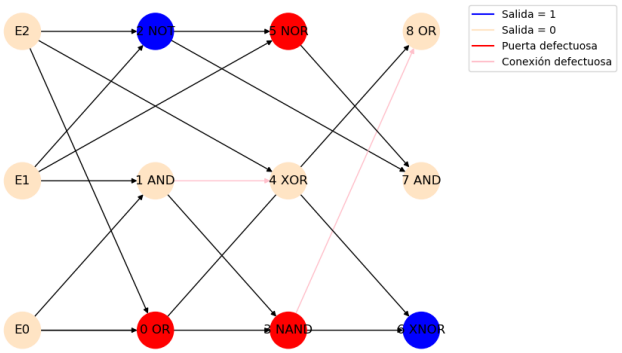

Fig. 3. Circuito defectuoso por defecto de prueba (en modo gráfico).

7) Tanto el tamaño de la población como el número de generaciones debe ser determinado por el usuario que ejecute el programa. Más adelante adjuntaremos las mediciones realizadas para diferentes valores de número de generaciones y tamaño de la población.

8) La interfaz gráfica tiene varias opciones que deben ser ejecutadas en orden para el correcto funcionamiento del programa. Todos los archivos y todas las imágenes producidas se incluirán en la carpeta “docs” del directorio raíz.

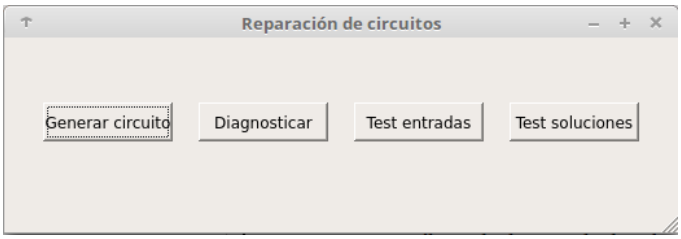

Fig. 4. Menú de inicio de la interfaz.

Fig. 5. Al pulsar el botón “Test entradas”, podremos ver en detalles qué salidas devuelve el circuito generado para las diferentes entradas.

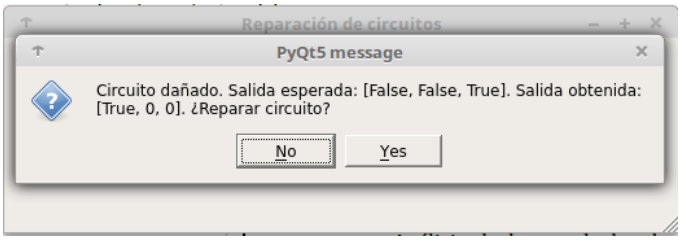

Fig. 6. Si se pulsa sobre "Diagnosticar", se detectarán los posibles errores que haya en el circuito generado.

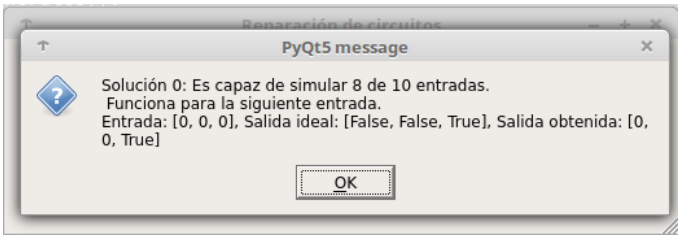

Fig. 7. Una vez ejecutado el algoritmo genético, se puede medir el rendimiento del circuito.

A continuación, entramos en detalle sobre los algoritmos usados a nivel de pseudocódigo.

Para simular el comportamiento de un circuito usamos, a partir de las puertas y las conexiones defectuosas, el siguiente algoritmo:

Procedimiento Simulador de circuito
Entrada:
- Vector de entradas
- Vector de puertas (circuito)
- Conexiones defectuosas

Salidas:
- Vector de salidas

Algoritmo:
1. entrada1,entrada2 = 0
2. Mientras I < tamaño(puertas)
   1. conexion1,conexion2 = Conexiones de la puerta de índice I
   2. Si I < n (primera capa)
      1. entrada1 = entradas[-1*(conexion1+1)]
      2. entrada2 = entradas[-1*(conexion2+1)]
   3. Si I > =n y I <n*2 (segunda capa)
      1. Si conexion1 < 0, entrada1 = entradas[-1*(conexion1+1)
      2. Si conexion1 >= 0, entrada1 = salida puerta de índice conexion1
      3. Repetir para conexion2
   4. Si no (resto de capas)
      1. entrada1,entrada2 = salida puerta indice conexion1, salida puerta de indice conexion2
   5. Si la conexion de conexion1 a puerta I es defectuosa, entrada1 = 0
   6. Si la conexion2 a puerta I es defectuosa, entrada2 = 0
   7. Asignar salida a puerta I desde entrada1 y entrada2
   8. i++
3. Devolver las salidas de las últimas n puertas

Para poder generar un circuito, ya sea aleatorio o por defecto, utilizamos el siguiente algoritmo:

Procedimiento Generador de circuitos
Entrada:
- Modo por defecto
- Vector de puertas vacío

Salidas:
- Circuito ideal (vector de puertas)
- Circuito defectuoso (vector de puertas)
- Representación gráfica y textual de ambos circuitos

Algoritmo:
1. Si Modo por defecto == 0
   a. N = Pedir número de puertas
   b. M = Pedir número de capas
   c. Entrada = Genera entradas(n)
   d. Pedir puertas defectuosas
   e. Pedir conexiones defectuosas
   f. i=0
   g. Mientras i != N*M
      i. Tipo=Aleatorio(0,6)
      ii. Si i<N (primera capa)
         1. conexion1,conexion2=Aleatorio(0,n)
         2. entrada1 = Entrada[conexion1]
         3. entrada2 = Entrada[conexion2]
         4. Convierte conexion1 a negativo
         5. Convierte conexion2 a negativo
      iii. Si i>n y i<n*2 (segunda capa)
         1. selector1,selecto2 = Aleatorio (0,1)
         2. conexion1,conexion2=Aleatorio(0,n)
         3. Si selector1 == 0, entrada1=Entrada[conexion1]
         4. Si no, entrada1 = Salida de puertas[conexion1]
         5. Pasos 3 y 4 para selector2, conexion2 y entrada2
      iv. Si no (resto de capas)
         1. minimo = Puerta mínima seleccionable
         2. maximo = Puerta máxima seleccionable
         3. conexion1,conexion2 = Aleatorio(minimo, maximo)
         4. entrada1, entrada2 = Salida de puertas[conexion1], salida de puertas[conexion2]
      v. Crear puerta de tipo Tipo, conexiones conexion1,conexion2, salida de Tipo(entrada1,entrada2), no defectuosa
      vi. Añadir puerta a vector de puertas
      vii. i++
   h. Representar circuito
   i. Repetir de i a vii para puertas y conexiones defectuosas
   j. Devolver circuito ideal y circuito defectuoso
2. Sino
   a. Crear 9 puertas por defecto
   b. Generar 10 entradas por defecto
   c. Indicar puertas defectuosas y conexiones defectuosas por defecto
   d. Crear circuito ideal
   e. Representar circuito ideal
   f. Crear circuito defectuoso
   g. Representar circuito defectuoso
   h. Devolver circuitos

Mediante el algoritmo anterior, el programa es capaz de diagnosticarse y comprobar que haya errores a reparar. En el caso de haberlos, se llama al algoritmo genético.

Para su implementación, hemos usado un cromosoma de tamaño 3*m*n, con genes formados por números que pueden tomar valores de 0 a 2*n, siendo n el número de puertas y m el número de capas.

Cada 3 genes del cromosoma representan a una puerta. El primer número nos indica el tipo de puerta que será, y el segundo y el tercero nos indican cuáles son sus conexiones entrantes (siguiendo el esquema de números positivos para las puertas y negativos para las entradas).

Si una de las conexiones está entre 0 y n, representa a la puerta de a+i de la capa m-2, donde m es la capa de la puerta que se está representando, a es el índice de la primera puerta de la capa m-2 e i es el número del gen.

Si la conexión está entre n y 2*n, representa a la puerta b + j de la capa m-1, donde b es el índice de la primera puerta de la capa m-1, y j es el número del gen.

Como sólo hay 7 tipos de puerta, al primer número del gen se le aplica mod 6, obteniendo así un número entre 0 y 6 que representará al tipo lógico de la puerta.

Un individuo será mejor o peor en función del número de salidas que obtenga iguales a la del circuito ideal al recibir las entradas generadas anteriormente. Mientras menor sea la diferencia, mejor será el individuo. Si en el cromosoma se genera una puerta con dos entradas repetidas, se le asignará un valor de diferencia de 1000, para que sea automáticamente descartado.

La evaluación de un individuo y su decodificación, como ya hemos explicado, vienen dadas por las siguientes funciones:

Procedimiento Evaluar Individuo
Entrada:
- Individuo
- Entradas (generadas con el circuito base)
- Circuito ideal
- Conexiones defectuosas

Salidas:
- Diferencia del individuo

Algoritmo:
1. Salidas, puertas, diferencia = Fenotipo(Individuo)
2. Si diferencia == 0
    1. Por cada entrada
        1. Salida ideal para entrada = Simula_Circuito(entrada, puertas ideal, [])
        2. Salida real para entrada = Simula_Circuito(entrada, puertas, conexiones defectuosas)
        3. Si salida ideal!= salida real, diferencia = diferencia+1
3. Devolver diferencia del individuo

Procedimiento Fenotipo
Entrada:
- Individuo

Salidas:
- Vector de salidas
- Vector de puertas (circuito)
- Valor de diferencia

Algoritmo:
1. entrada1,entrada2 = 0
2. n = Número de puertas
3. m = Número de capas
4. Puertas = []
5. Mientras I < tamaño(puertas)
    1. j= I//3
    2. tipo, conexion1,conexion2 = individuo[i], individuo[i+1], individuo[i+2]
    3. tipo = tipo mod(6)
    4. Si conexion1 == conexion2, diferencia = 1000 y acaba el bucle
    5. Si j < n (primera capa)
        1. Si la conexion1-n == conexion2 o conexion1==conexion2-n, diferencia=1000 y acaba el bucle
        2. Si conexion1<n, entrada1= entradas[conexion1]; representar conexion1 como entrada
        3. Si no, entrada1= entradas[conexion1-n]; representar conexion1 como entrada
        4. Repetir para entrada 2
    6. Si j> =n y j<n*2 (segunda capa)
        1. Si conexion1<n, entrada1= entradas[conexion1]; representar conexion1 como entrada
        2. Si no, entrada1= salida de puertas[conexion1-n];
        3. Repetir para conexion2
    7. Si no (resto de capas)
        1. Si conexion1<n, entrada1 = salida de puertas[capa actual-2*n]+conexion1
        2. Si no, entrada1 = salida de puertas[capa actual-1*n]+conexion1-n
        3. Repetir para conexion2
    8. Si la conexion de conexion1 a puerta I es defectuosa, entrada1 = 0
    9. Si la conexion2 a puerta I es defectuosa, entrada2 = 0
    10. Se crea la puerta con las conexiones y tipo calculadas, indicando si es defectuosa y asignando su salida
    11. Se añade al vector de puertas
6. Devolver las salidas de las últimas n puertas, el vector puerta y la diferencia

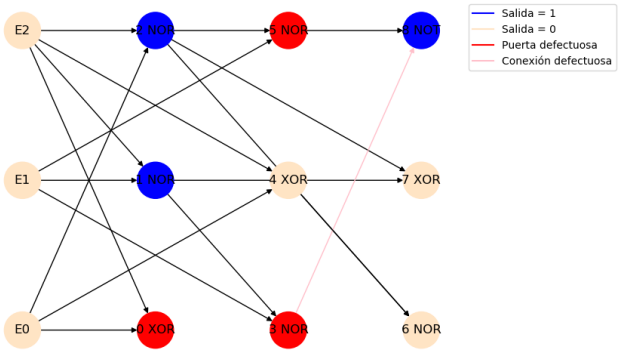

Fig. 8. Un ejemplo de las soluciones conseguidas a la hora de ejecutar el algoritmo de reparación.

Aunque se calcule las salidas en el algoritmo genético, esto es simplemente para representar el circuito gráficamente de forma más clara. A efectos prácticos, no es necesario calcular la salida pues ya se hace en la función de evaluación.

IV. RESULTADOS

A continuación, se detallarán los resultados obtenidos modificando los distintos parámetros de nuestro programa. Mediante el uso de la librería DEAP, podemos obtener diferentes estadísticas acerca de las iteraciones del algoritmo genético.

Para mantener unos valores que se puedan medir de manera estable, usaremos el circuito de prueba expuesto en el apartado anterior, cambiando los valores del tamaño de la población y del número de generaciones, así como del factor de mutación y la probabilidad de mutación y de cruce.

Los valores de las columnas de mínimo, media y máximo indican el mínimo, la media y el máximo valor de la primera generación y de la última, separados por un guión.

Tabla 1. Primer experimento. Probabilidad de cruzamiento 0.5 y probabilidad de mutación 0.3

#Test	*Tam.*	*N.º Gener.*	*Mínimo*	*Media*	*Máx.*
1	500	100	4-0	891-183	1000-1000
2	1000	500	2-0	911-173	1000-1000
3	500	1000	4-0	918-176	1000-1000
4	2000	250	2-0	908-188	1000-1000
5	250	3000	2-0	897-192	1000-1000

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[0, 2, 3, 2, 5, 1, 0, 2, 3, 5, 1, 0, 2, 3, 0, 3, 4, 1, 5, 1, 5, 1, 1, 5, 0, 1, 4]
Individuo con fitness: 0.0
[2, 4, 0, 3, 3, 1, 4, 1, 5, 2, 2, 5, 5, 2, 3, 1, 5, 2, 5, 0, 4, 0, 0, 5, 3, 1, 0]
Individuo con fitness: 0.0
[0, 2, 0, 2, 3, 1, 4, 1, 5, 2, 2, 5, 5, 2, 3, 1, 5, 2, 5, 0, 4, 0, 0, 5, 3, 1, 0]
gen     nevals  mínimo  media   máximo
0       250     2       896,884 1.000
1       179     2       758,064 1.000
2       168     2       583,552 1.000
3       146     0       373,072 1.000
4       169     0       249,648 1.000
5       160     0       253,156 1.000
6       148     0       208,768 1.000
7       166     0       168,292 1.000
8       154     0       167,304 1.000
9       165     0       190,492 1.000
10      163     0       154,164 1.000
```

Fig. 9. Output de la prueba 5

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[5, 1, 2, 0, 2, 1, 3, 1, 5, 0, 2, 5, 5, 2, 3, 4, 1, 3, 5, 4, 5, 5, 2, 1, 2, 5, 1]
Individuo con fitness: 0.0
[0, 0, 1, 2, 5, 4, 2, 3, 1, 1, 5, 3, 4, 0, 5, 4, 4, 1, 2, 1, 4, 4, 3, 5, 3, 3, 5]
Individuo con fitness: 0.0
[3, 1, 2, 2, 2, 0, 4, 2, 4, 0, 5, 4, 4, 0, 5, 4, 3, 5, 2, 1, 5, 1, 5, 3, 2, 5, 1]
gen nevals  mínimo  media   máximo
0   2000    2       908,303 1.000
1   1301    0       797,753 1.000
2   1255    2       599,417 1.000
3   1315    0       373,622 1.000
4   1247    0       218,732 1.000
5   1322    0       195,51  1.000
6   1255    0       192,309 1.000
7   1336    0       199,274 1.000
8   1270    0       201,345 1.000
9   1305    0       214,582 1.000
10  1271    0       187,883 1.000
```

Fig. 10. Output de la prueba 4

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[5, 5, 0, 2, 2, 3, 1, 0, 4, 4, 1, 3, 3, 5, 3, 2, 0, 4, 5, 3, 1, 5, 4, 5, 5, 0, 3]
Individuo con fitness: 0.0
[5, 1, 5, 0, 4, 2, 4, 3, 5, 5, 1, 5, 0, 2, 3, 2, 1, 5, 4, 4, 0, 1, 0, 1, 2, 0, 2]
Individuo con fitness: 0.0
[3, 4, 2, 0, 4, 2, 4, 3, 5, 5, 1, 5, 0, 2, 3, 2, 1, 5, 4, 4, 0, 1, 0, 1, 2, 0, 2]
gen nevals  mínimo  media   máximo
0   1000    2       911,768 1.000
1   619     0       825,447 1.000
2   657     0       674,661 1.000
3   678     0       470,271 1.000
4   648     0       283,402 1.000
5   650     0       198,223 1.000
6   642     0       197,477 1.000
7   656     0       214,857 1.000
8   634     0       198,599 1.000
9   644     0       196,208 1.000
10  665     0       184,035 1.000
```

Fig. 11. Output de la prueba 3

Como podemos ver, con la probabilidad de cruzamiento 0.5 y de mutación 0.3 y las variaciones de tamaño y número de generaciones nos dan aproximadamente los mismos resultados. Al final, se consigue una eficiencia del 100% (0 de diferencia) con las 3 mejores soluciones.

A partir de la 4 generación, la media de fitness (valor de diferencia del individuo) comienza a estabilizarse alrededor de 200.

Probemos ahora nuevos valores más altos para la probabilidad de cruzamiento y de mutación:

Tabla 2. Segundo experimento. Probabilidad de cruzamiento 0.7 y probabilidad de mutación 0.5

#Test	*Tam.*	*N.º Gener.*	*Mínimo*	*Media*	*Máx.*
1	500	100	4-0	898-309	1000-1000
2	1000	500	4-0	892-278	1000-1000
3	500	1000	2-0	909-341	1000-1000
4	2000	250	3-0	914-327	1000-1000
5	250	3000	4-0	905-321	1000-1000

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[3, 5, 1, 3, 1, 5, 3, 4, 2, 1, 3, 2, 2, 2, 5, 1, 5, 4, 2, 4, 2, 1, 0, 4, 3, 5, 4]
Individuo con fitness: 0.0
[2, 1, 0, 5, 3, 2, 3, 4, 3, 0, 4, 1, 5, 3, 2, 1, 0, 4, 5, 4, 1, 1, 2, 5, 5, 5, 3]
Individuo con fitness: 0.0
[0, 4, 0, 2, 0, 1, 5, 1, 3, 1, 3, 2, 2, 2, 5, 1, 5, 4, 2, 4, 2, 1, 0, 4, 3, 5, 4]
gen     nevals  mínimo  media   máximo
0       250     4       904,792 1.000
1       205     4       805,68  1.000
2       207     2       654,9   1.000
3       204     2       531,836 1.000
4       211     3       408,612 1.000
5       215     2       356,712 1.000
6       218     2       352,484 1.000
7       215     2       320,18  1.000
8       209     0       300,024 1.000
9       212     0       331,668 1.000
10      216     0       378,852 1.000
```

Fig. 12. Output de la prueba 5

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[4, 3, 1, 1, 3, 2, 1, 5, 0, 1, 4, 0, 4, 2, 4, 2, 4, 1, 0, 1, 4, 0, 0, 3, 2, 5, 0]
Individuo con fitness: 0.0
[3, 3, 4, 3, 3, 2, 0, 0, 2, 3, 0, 4, 2, 2, 0, 4, 0, 2, 5, 4, 5, 1, 4, 5, 5, 3, 5]
Individuo con fitness: 0.0
[3, 4, 0, 0, 0, 1, 2, 5, 1, 1, 5, 1, 5, 3, 4, 4, 1, 3, 2, 2, 4, 1, 5, 0, 2, 3, 4]
gen nevals  mínimo  media   máximo
0   2000    3       914,767 1.000
1   1687    2       829,001 1.000
2   1736    2       694,674 1.000
3   1683    2       560,277 1.000
4   1725    2       439,164 1.000
5   1684    0       367,5   1.000
6   1693    0       334,926 1.000
7   1711    0       331,645 1.000
8   1694    0       317,868 1.000
9   1679    0       312,1   1.000
10  1698    0       327,784 1.000
```

Fig. 13. Output de la prueba 4

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[1, 1, 3, 2, 4, 3, 3, 5, 4, 5, 0, 4, 2, 2, 4, 1, 4, 3, 5, 0, 4, 1, 5, 4, 2, 0, 5]
Individuo con fitness: 0.0
[4, 5, 1, 4, 0, 5, 2, 5, 4, 5, 0, 4, 2, 2, 4, 1, 4, 3, 5, 0, 4, 1, 5, 4, 2, 0, 5]
Individuo con fitness: 0.0
[5, 0, 2, 2, 4, 3, 3, 5, 4, 5, 0, 4, 2, 2, 4, 1, 4, 3, 5, 0, 4, 1, 5, 4, 2, 0, 5]
gen     nevals  mínimo  media   máximo
0       500     2       908,774 1.000
1       432     3       825,538 1.000
2       433     3       708,552 1.000
3       416     2       533,99  1.000
4       409     0       393,062 1.000
5       430     0       353,05  1.000
6       436     0       309,126 1.000
7       434     0       304,732 1.000
8       422     0       346,158 1.000
9       422     0       316,104 1.000
10      433     0       313,954 1.000
```

Fig. 14. Output de la prueba 3

Como podemos ver, con la probabilidad de cruzamiento 0.7 y de mutación 0.5 y las variaciones de tamaño y número de generaciones nos dan aproximadamente los mismos resultados. Al final, se consigue una eficiencia del 100% (0 de diferencia) con las 3 mejores soluciones.

A partir de la 5 generación, la media de fitness (valor de diferencia del individuo) comienza a estabilizarse alrededor de 300.

Aunque se lleguen al mismo número de soluciones válidas que en el experimento anterior, podemos comprobar que la media de fitness aumenta en 100 aproximadamente. Con esto, podemos concluir que usando unos valores más altos para determinar la probabilidad no implica un mejor valor de diferencia. Además, hace que el algoritmo genético tarde más generaciones en normalizar su valor.

Probemos ahora nuevos valores para la probabilidad de cruzamiento y de mutación, esta vez menores que en el primer experimento:

Tabla 3. Tercer experimento. Probabilidad de cruzamiento 0.3 y probabilidad de mutación 0.2

#Test	*Tam.*	*N.º Gener.*	*Mínimo*	*Media*	*Máx.*
1	500	100	4-0	907-110	1000-1000
2	1000	500	4-0	896-127	1000-1000
3	500	1000	4-0	921-106	1000-1000
4	2000	250	4-0	907-127	1000-1000
5	250	3000	4-0	877-121	1000-1000

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[5, 3, 1, 2, 5, 1, 5, 5, 1, 1, 2, 3, 1, 2, 1, 5, 4, 1, 2, 1, 5, 1, 3, 0, 2, 5, 0]
Individuo con fitness: 0.0
[5, 3, 1, 2, 5, 1, 5, 5, 1, 1, 2, 3, 1, 2, 1, 5, 4, 1, 2, 1, 5, 1, 2, 0, 2, 5, 0]
Individuo con fitness: 0.0
[5, 4, 5, 5, 2, 4, 1, 1, 2, 1, 2, 1, 5, 2, 5, 3, 0, 4, 5, 1, 4, 1, 2, 4, 2, 3, 0]
gen     nevals  mínimo  media   máximo
0       250     4       877,076 1.000
1       91      6       710,604 1.000
2       109     5       488,62  1.000
3       100     5       262,308 1.000
4       106     4       110,908 1.000
5       129     2       145,784 1.000
6       125     2       149,424 1.000
7       120     2       136,94  1.000
8       79      2       108,4   1.000
9       113     0       123,692 1.000
10      101     0       122,944 1.000
```

Fig. 15. Output de la prueba 5

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[4, 3, 2, 0, 5, 0, 2, 2, 4, 0, 1, 4, 1, 2, 1, 1, 3, 4, 2, 2, 3, 4, 3, 5, 3, 0, 4]
Individuo con fitness: 0.0
[3, 4, 5, 0, 0, 1, 5, 4, 3, 2, 2, 5, 0, 3, 2, 1, 0, 1, 0, 4, 3, 1, 2, 0, 3, 3, 1]
Individuo con fitness: 0.0
[1, 4, 5, 1, 5, 0, 4, 1, 3, 0, 5, 4, 0, 3, 2, 4, 1, 3, 0, 4, 3, 1, 2, 0, 3, 1, 0]
gen nevals  mínimo  media   máximo
0   2000    4       906,854 1.000
1   925     2       789,894 1.000
2   852     2       560,875 1.000
3   816     2       289,959 1.000
4   920     2       160,684 1.000
5   920     0       144,306 1.000
6   844     0       130,497 1.000
7   869     0       127,859 1.000
8   870     0       113,507 1.000
9   868     0       123,969 1.000
10  881     0       120,612 1.000
```

Fig. 16. Output de la prueba 4

```
Las tres mejores soluciones encontradas han sido:
Individuo con fitness: 0.0
[5, 5, 4, 2, 0, 1, 2, 2, 0, 4, 5, 2, 3, 2, 4, 3, 1, 4, 2, 2, 3, 1, 3, 2, 0, 3, 4]
Individuo con fitness: 0.0
[2, 0, 5, 3, 3, 5, 2, 2, 0, 0, 1, 2, 4, 0, 3, 1, 5, 2, 2, 2, 5, 1, 3, 1, 3, 3, 2]
Individuo con fitness: 0.0
[5, 0, 5, 3, 3, 5, 2, 2, 0, 0, 1, 2, 4, 0, 3, 1, 5, 2, 2, 2, 5, 1, 3, 1, 3, 3, 2]
gen     nevals  mínimo  media   máximo
0       500     4       920,706 1.000
1       214     2       783,926 1.000
2       232     2       528,13  1.000
3       201     2       266,04  1.000
4       243     1       130,316 1.000
5       219     1       113,354 1.000
6       248     2       154,246 1.000
7       212     2       109,698 1.000
8       228     2       146,938 1.000
9       213     2       96,586  1.000
10      221     2       118,178 1.000
```

Fig. 17. Output de la prueba 3

Como podemos ver, con la probabilidad de cruzamiento 0.3 y de mutación 0.2 y las variaciones de tamaño y número de generaciones nos dan aproximadamente los mismos resultados. Al final, se consigue una eficiencia del 100% (0 de diferencia) con las 3 mejores soluciones.

A partir de la 4 generación, la media de fitness (valor de diferencia del individuo) comienza a estabilizarse alrededor de 100.

Además, podemos ver que con estos valores de probabilidad de cruzamiento y mutación conseguimos una media incluso de 2 cifras, lo que significa que estos valores son los más óptimos para los experimentos realizados.

Sin embargo, hay que especificar que, debido al componente aleatorio de los algoritmos genéticos, esto no tiene por qué cumplirse siempre. Sin embargo, tras 5 pruebas con los diferentes valores podemos asegurar con cierta seguridad que la mayoría de las veces, los resultados entrarán dentro del rango esperado.

V. CONCLUSIONES

El autodiagnóstico y la autorreparación de circuitos es algo de vital importancia en situaciones en las que el circuito es inaccesible para los técnicos y reparadores. Por tanto, la creación de un algoritmo que pueda buscar soluciones es de igual importancia. Al tratarse de un problema que no tiene una solución directa, es necesario probar diferentes caminos.

Por tanto, el uso de un algoritmo genético es una de las posibles formas de resolverlo. Aunque para circuitos más grandes y complejos se necesitaría un hardware más avanzado, para la tecnología disponible hemos usado un circuito más sencillo de tamaño 3x3. Así, las soluciones a las que hemos llegado han sido equivalentes: En todos los casos se ha llegado a reconstruir un circuito equivalente con un 100% de rendimiento.

En los experimentos hemos variado el número de generaciones y tamaño de la población. Sin embargo, mientras ambos tengan valores lo suficientemente altos, no influyen tanto como la probabilidad de cruzamiento y de mutación. Para valores bajos de éstas dos últimas medidas, es cuando el algoritmo ha demostrado un rendimiento óptimo.

El algoritmo podría obtener resultados más eficientes si en lugar de devolver 1000 como diferencia en caso de una conexión repetida, se usase una conexión diferente.

Además, debido a la aleatoriedad del algoritmo genético, muchas veces no consigue llegar a una solución óptima. En estos casos, sería interesante poder comparar los resultados obtenidos con otros algoritmos como redes neuronales.

Por último, se podría mejorar la interfaz gráfica: por un lado, mostrar las representaciones del circuito en el mismo menú, y por otro lado cuidar las ambigüedades que se puedan dar en éstas.

REFERENCIAS

[1] Wikipedia: Computación reconfigurable. https://en.wikipedia.org/wiki/Reconfigurable_computing

[2] Wikipedia: Lógica programada. https://es.wikipedia.org/wiki/L%C3%B3gica_programada

[3] Librería DEAP utilizada para implementar el algoritmo genético. https://github.com/deap/deap

[4] Librería Numpy utilizada para implementar el algoritmo genético. http://www.numpy.org/

[5] Librería NetworkX utilizada para la representación en grafo. https://networkx.github.io/

[6] Librería Matplotlib utilizado para la representación en grafo. https://matplotlib.org/

[7] Librería PyQt5 utilizada para la interfaz gráfica. https://www.riverbankcomputing.com/software/pyqt/download5

[8] Estrin, G (2002). "Reconfigurable computer origins: the UCLA fixed-plus-variable (F+V) structure computer". *IEEE Ann. Hist. Comput*. 24 (4): 3–9. doi:10.1109/MAHC.2002.1114865.

[9] Sekanina L. (2003) Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. In: Tyrrell A.M., Haddow P.C., Torresen J. (eds) Evolvable Systems: From Biology to Hardware. ICES 2003. Lecture Notes in Computer Science, vol 2606. Springer, Berlin, Heidelberg. doi:10.1007/3-540-36553-2_17