De entradas y salidas.

Carlos Castaño del Castillo
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
UVUS: carcasdel1@alum.us.es y carloscastcast97@gmail.com

I. Introducción

En este proyecto el problema a solventar se trata de auto-reparar circuitos electrónicos adecuados para así determinar si nuestro hardware ha sufrido algún tipo de daño físico y, en ese caso, obtener una nueva configuración para obtener un comportamiento global adecuado.

Este tipo de circuitos se plantean visualmente como una matriz de mxn donde m resultan ser el número de capas y n la cantidad de puertas lógicas que serán incluidas en fila dentro de cada nivel.

Las puertas lógicas son dispositivos electrónicos con una función booleana o aritmética; así como sumar o restar. Son aplicadas en tecnologías eléctricas, mecánica, electrónica, hidráulica y neumática. [1] Encontramos una gran variedad de ejemplares para nuestro proyecto:

1) Puertas con lógica directa.

- Puerta SI/BUFFER: La ecuación característica que describe el comportamiento de la puerta SI es:
  - F =A

Entrada A	Salida A
0	0
1	1

- Puerta AND: La puerta lógica que representa la unión 'Y'. Realiza la función booleana de un producto lógico. Su símbolo es '·' aunque se suele omitir. Es el producto de dos variables. Su ecuación característica es:
  - F = A·B

Entrada A	Entrada B	Salida
0	0	0
0	1	0
1	0	0
1	1	1

- Puerta OR: La puerta de la exclusión 'O'. Indica que se suman los dos valores.
  - F = A+B

Entrada A	Entrada B	Salida
0	0	0
0	1	1
1	0	1
1	1	1

- Puerta XOR: La puerta lógica OR-exclusiva, realiza la función booleana A'B+AB', su signo es un + inscrito en un círculo.
  - F = A'·B+A·B'

Entrada A	Entrada B	Salida
0	0	0
0	1	1
1	0	1
1	1	0

2) Puertas con lógica negada.

- Puerta NOT: La puerta lógica NO, realiza la función booleana de inversión o negación de una variable lógica.
  - F = A'

Entrada A	Salida
0	1
1	0

- Puerta NAND: La puerta lógica NO-Y realiza la operación de producto lógico negado. En ocasiones es llamada también barra de Sheffer.
  - F = (A·B)' = A' + B'

Entrada A	Entrada B	Salida

0	0	1
0	1	1
1	0	1
1	1	0

- Puerta NOR: La puerta lógica NO-O realiza la operación de suma lógica negada. En ocasiones es llamada también barra de Pierce.
  - F = (A + B)' = A'· B'

Entrada A	Entrada B	Salida
0	0	1
0	1	0
1	0	0
1	1	0

- Puerta XNOR: La puerta NO-Exclusiva es el complemento de la puerta OR exclusiva.
  - F = A·B + A'· B'

Una definición bastante completa de un algoritmo genético es la propuesta por John Koza:

"Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo con el principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud". [2]

Los Algoritmos genéticos se compondrán de: los cromosomas (lista de genes que representan objetos o cualidades de éstos), genes (valor numérico dentro del cromosoma), las poblaciones (conjuntos de individuos empleados en el problema) y las generaciones (nuevos individuos que son resultado de un cruce entre ancestros o mutación de alguno de ellos). Consiguiendo una manera de representar finalmente la posible auto-reparación que pueda obtener el circuito.

II. Preliminares

En este proyecto he empleado el siguiente método:

- Algoritmos Genéticos: explicado anteriormente en qué consistía, se ha utilizado la librería deap, vista en una práctica en clase, en referencias adjuntaremos la api que hemos utilizado para entender mejor esta librería. [3]

  Una buena razón de la decisión de toma de estos métodos la describe el señor Yepes Piqueras:

  "Podemos resolver problemas difíciles simplemente emulando lo que hace la naturaleza en procesos físicos o biológicos". [4]

- También hemos empleado la librería importlib, distribuyendo así las funciones y métodos que haya en el trabajo por categorías: Relacionadas con el cromosoma, la interacción con el usuario y las operaciones de puertas lógicas.

III. Metodología

Para la resolución del trabajo he creado varias secciones importantes y una extra para los puntos adicionales:

*1) La primera sección contiene la¡os métodos que ejecutan las puertas lógicas. S*on llamados con el siguiente formato 'plX_nombrePuerta'. En estos métodos introducimos los bits correspondientes a las entradas de la puerta lógica. Obteniendo como resultado el bit de salida tras su operación aritmética y/o booleana. Su pseudocódigo sería:

plx_nombrePuerta(bits de entrada):
1. Si bits de entrada = X e Y
   a) Devolver valor M
2. Si no es así:
   a) Devolver valor N.

2) Nos encontramos en la sección de interacción con el usuario. Aquí mostramos información por pantalla al usuario para que introduzca los valores convenientes y así proseguir con la obtención de información sobre el circuito y sus auto-reparaciones.

a) SolicitarNumeroAlUsuario: Solicita al usuario un valor numérico entre un punto inicial y otro final, ambos inclusive. En el caso de que el usuario escriba por teclado un valor que no corresponda con lo solicitado, volverá a preguntar por pantalla.

solicitarNumeroAlUsuario(incio, final):
1. Imprimir por pantalla información
2. Solicita valor al usuario
   a) Mientras que sea nulo o fuera de rango:
      - solicitar valor al usuario.
3. Devolver valor.

b) Obtener/SeleccionarDatos: Piden al usuario información llamando dentro del método a solicitarNumeroAlUsario() con los valores correspondientes para cada circunstancia.

obtener/seleccionarDatos():
3. Imprimir por pantalla información
4. dato = solicitaNumeroAlUsuario(inicio, final)
4. Devolver dato

Solicitamos los siguientes datos:

- Número de puertas lógicas diferentes
- Número de capas del circuito.
- Número de columnas del circuito.
- Valores de entrada (bits)
- Valores de salida (bits)
- Población para el algoritmo genéticos
- Número de iteraciones en el algoritmo génetico.
- Probabilidad de cruce
- Probabilidad de mutación en el cromosomas
- Probabilidad de mutación en el genoma.

3) Esta sección está destinada para todos los métodos que tienen relación con el cromosoma respecto a algoritmos genéticos o por relación al circuito.

a) *CrearCromosomaInicial():* Esta función recorre un array en forma de matriz para ir rellenando las diferentes casillas del cromosoma (genes) indicando la numeración y, por consiguiente, de qué puerta lógica se trata.

crearCromosomaInicial(filas, columnas, puertasDiferentes):

1. Inicializar cromosoma vacío, I =1, j=1
2. Mientras que i<= filas:
    a) j = 1
    b) Mientras que j<=columnas:
        - Imprimitr por pantalla información.
        - Valor = solicitarNumeroAlUsuario()
        - cromosoma.append(valor)
        - j = j+1
    c) i = i+1
3. Devolver cromosoma

b) CrearEnlacesCromosoma(): Este método se basa en crear un array secundario donde indicamos en la casilla de íncide n que su salida está enlazada por cable con la entrada m. Es decir, si en la primera casilla del array = 3 indicamos que la puerta lógica nº0 usa su salida como entrada de la puerta lógica nº3. Utilizaremos un filtro para que no creemos enlaces de salida en la última fila, ya que son salida directa del sistema.

crearEnlacesCromosoma(filas, columnas, cromosoma):

1. Inicializar enlaces vacío.
2. Para índice=0 hasta longitid(cromosoma)
    a) Si indice < columnas*(filas-1)
        - valorEnlace = comprobarEnlaceValido(cromosoma, enlaces, indice, filas, columnas)
        - enlaces.append(valorEnlace)
3. Devolver enlaces

c) ComprobarEnlaceValido(): En esta función comprobamos y mostramos las posibles casillas con las que puede enlazar la puerta nºX independientemente de su posición, comprobando que la primera posición válida es la puerta inicial del nivel inferior. También que la última posible sea el final de dos niveles más abajo. En caso de estar en la penúltima fila, solo habría un nivel por debajo de la casilla actual donde elegir. La fórmula obtenida para las casillas fueron realizadas mediante módulos de matemática discreta. Finalmente vemos que depende de la puerta lógica que queramos asignar como entrada, puede tener una o dos posibles entradas, lo cual también se tiene en cuenta.

En una matriz de M capas y N puertas por cada capa:

1ª Posición válida = Índice – Indice mod(N) + M
*Uª Posición válida = 1ª PV + 2*N-1 ó 1ªPV + N-1*

comprobarEnlaceValido(cromosoma, enlaces, idx, filas, columnas):

1. 1PV y UPV = 0, 0
2. 1PV = idx – (idx%columnas) + columnas
3. UPV = 1PV +2*columnas-1
4. Si UPV > filas*columnas:
    a) UPV = 1PV + columnas-1
5. Mientras 1:
    a) valor = solicitarNumeroAlUsuario(1PV, UPV)
    b) Si tiene una entradaSimple & estaEntradaSimpleDisponible()
        - Devolver valor
    c) Si tiene entrada doble & estaEntradaDobleDisponible()
        - Devolver valor
    d) En caso contrario:
        - Informar al usuario que dicha entrada no es válida.

d) estaEntradaSimpleDisponible(): Comprueba que esta puerta no tenga su índice asignado dentro de la lista de enlaces. De esta manera sabemos que no hay ninguna puerta lógica conectada a ella.

estaEntradaSimpleDisponible(valor, enlaces):
1. Para indice en longitud(enlaces)
    a) si valor == enlaces[indice]:
    - Devolver Falso
2. Devolver Verdadero

e) estaEntradaDobleDisponible(): Comprueba que esta puerta no tenga su índice asignado dentro de la lista de enlaces más de una vez. De esta manera sabemos que hay al menos una entrada libre.

estaEntradaSimpleDisponible(valor, enlaces):
1. contador = 0
2. Para indice en longitud(enlaces)
    a) si valor == enlaces[indice]:
    - contador = contador +1
    b) Si contador == 2:
    - Devolver Falso
3. Devolver Verdadero.

f) CalcularSalidasDelCircuito(): Este método se basa en la obtención del resultado final del circuito. Para ello tenemos que crear un diccionario donde almacenamos los valores de salida de cada una de las puertas lógicas del circuito. Vamos filtrando ciertas acciones según en qué capa estemos. Mientras que no sea la última línea, tendremos una salida que conectar a otra puerta (lo cual he considerado obligatorio). Si está en la primera fila: el valor que obtiene es el bit posicional, etc.

calcularSalidasDelCircuito(valoresEntrada, circuito, enlaces):

1. creamos un diccionario y una lista para los valores finales de salida.
2. Para todas las puertas del circuito:
    a) puertaContectada = None
    b) Si no está en la última capa:
    - puertaConectada = enlaces[indice]
    c) Si está en la primera fila:
    - valorSalidaPuerta = calcularSalidaPuerta()
    d) Si no:
    - entrada1, entrada2 = obtenerEntradasPuerta()
    - valorSalidaPuerta = calcularSalidaPuerta()
        - Si es la última fila:
            - lista.append(valorSalidaPuerta)
    - diccionario = anadirNuevoValorSalienteComoEntrada()

3. Delvolver lista

g) calcularSalidaPuerta(): Según el tipo de puerta y los valores de entrada que recibe esta estructura, devolverá un valor u otro llamando a los métodos de las acciones referentes a puertas lógicas.

calcularSalidaPuerta(tipoPuerta, entrada1, entrada2):
1. Si tipoPuerta == 0
    a) return 0
2. Si entrada1 == None
    a) entrada1 = entrada2
3. Si entrada2 == None
    a) entrada2 = entrada1
4. Si tipoPuerta = NOT
    a) return pl3_not(entrada1)
5. Si tipoPuerta = YES
    a) return pl3_yes(entrada1)
6. Si tipoPuerta = OR
    a) return pl3_or(entrada1, entrada2)
7. Si tipoPuerta = AND
    a) return pl3_and(entrada1, entrada2)
8. Si tipoPuerta = NAND
    a) return pl3_nand(entrada1, entrada2)
9. Si tipoPuerta = XOR
    a) return pl3_xor(entrada1, entrada2)
10. Si tipoPuerta = XNOR
    a) return pl3_xnor(entrada1, entrada2)
11. Si tipoPuerta = NOR
    a) return pl3_nor(entrada1, entrada2)

h) ObtenerEntradasPuerta(): Busca en el diccionario de salidas de puertas los valores referentes a la clave 'e'. En caso de que no haya valores, devolvemos dos 0's. En caso de haber un único valor, devolvemos el valor a y b. Por último, si hubiera dos valores, devolvemos tanto a como b.

obtenerEntrasasPuerta(diccionario, e):
1. entrada1, entrada2 = 0, 0 ; clave = None
2. Para key en todo el diccionario:

    a) Si key == e
    - Si no hay valores
        - Salir
    - Si hay un valor:
        - entrada1 = valor de key
        - salir
    - Si no:
        - entrada1 = valor1
        - entrada2 = valor2
        - salir

3. Devolver entrada1, entrada2

i) AnadirNuevoValorSalientComoEntrada(): Este método comprueba si el diccionario contiene la clave 'puertaConectada'. En caso de no estar, se crea esta nueva clave. Después de esto (o de darse el caso de que ya existía), añadimos el valor asignado a dicha clave.

AnadirNuevoValorSalienteComoEntrada(diccionario, puertaConectada, valorSalidaPuerta):

1. Si no está puertaConectada en diccionario:
    a) diccionario[puertaConectada] = [valorSalidaPuerta]

2. diccionario[puertaConectada].append(valorSalidaPuerta)
3. Devolver diccionario

j) EvaluaCircuito(): Es la función fitness de nuestro algoritmo genético. Lo que hemos hecho es sumar el número de puertas diferentes que vamos a tener respecto al circuito original. Además, incluir una penalización de cien puntos por cada valor de salida que no sea correspondido.

evaluaCircuito(cOriginal, circuito, entrada, salidas buenas, enlaces):
1. puntuacion = 0
2. valoresSalida = calcularSalidasDelCircuito()

3. puntuacion = diferenciasEntre(valoresSalida, salidasBuenas)*100 + diferenciasEntre(circuito, cOriginal)

4. Devolver puntuacion

k) DiferenciaEntre(): Este método simplemente crea un acumulador que va aumentando a medida que hay diferencias de valores entre dos listas del mismo tamaño.

l) ImprimeFormatoCircuito(): Este método se emplea tras la finalización del algoritmo genético y tenemos ya nuestro cromosoma predilecto. Se crea un string donde mostramos cada uno de sus niveles y las diferentes puertas que éste tiene en él.

4) En esta última sección voy a comentar los puntos extras que he añadido para este proyecto.

a) Puertas lógicas: He añadido tres puertas lógicas extras además de las cinco obligatorias y la puerta rota. Son las siguientes: BUFFER, NOR y XNOR. De esta manera conseguimos que haya mayor variedad a la hora de modificar o crear los circuitos y así el número de cambios puede ser menor.

b) Visualización pseudográfica: Muestro a través de pantalla un boceto del circuito final que divide el circuito por sus respectivas capas mostrando las diferentes puertas lógicas que encontramos en cada una. Así al finalizar el proceso de algoritmos genéticos podemos tener una idea del resultado final y facilitando la tarea al usuario.

c) Versatilidad: A la hora de hacer el algoritmo genético podemos modificar casi la totalidad de valores y datos relacionados con el algoritmo genético y el circuito. Son los siguientes:
   a) Nº de capas del circuito.
   b) Nº de puertas por capa.
   c) Nº de puertas disponibles para la creación del circuito.
   d) La distribución de puertas lógicas en el circuito original.
   e) La distribución original de los enlaces entre puertas lógicas.
   f) Valor de población inicial de AG.
   g) Número de iteraciones de AG.
   h) Probabilidad de cruce de cromosomas.
   i) Probabilidad de mutación de cromosoma.
   j) Probabilidad de mutación de genoma.

d) Pruebas manuales: He creado un pequeño fichero que puede realizar compraraciones automáticas entre un circuito original que nosotros introduzcamos y una sucesión de circuitos que queramos nosotros también. Tras cada comparación, nos da la posibilidad de incluir un nuevo circuito con el que comparar o salir del programa. Si elegimos la segunda opción, entonces nos mostrará una lista de valores fitness obtenidos.

e) Pruebas montadas: He creado otro fichero donde incluyo un ejemplo de cada una de los tipos de pruebas que he ido realizando a lo largo del periodo de pruebas del proyecto. De esta manera se puede ejecutar y comprobar de manera automática los diferentes resultados.

IV. Resultados

Antes de comenzar propiamente con los resultados que se han obtenido a lo largo del proceso de pruebas, me gustaría

matizar ciertos detalles respecto a la configuración de los circuitos debido a las decisiones que he tomado.

- He considerado que las puertas que estén rotas deberán devolver el valor '0' independientemente de si se cambia dicha puerta por cualquier otra. Ya que consideramos que una puerta rota se refiere al hardware completo. Así que para ello creé un método auxiliar a la hora de evaluar los individuos para que colocase una entrada rota en las posiciones originales que estaban atribuidas, de esta manera conseguíamos un 'parche'. Debido a que el algoritmo genético modifica a su antojo los valores de los cromosomas.
- Los enlaces en la placa de circuitos son estáticos, es decir, no se pueden modificar los enlaces una vez de salidas y entradas por las puertas lógicas.
- Cada puerta lógica tiene únicamente una salida.
- Toda salida de puerta lógica debe ir conectada a otra a través de la entrada de la segunda. La única excepción al respecto es la última fila, cuya salida será directamente el valor de los bits mostrados al final del circuito.

Las pruebas iniciales las realicé en conjuntos pequeños, circuitos de 3x3 hasta 5x5, probando casos simples con puertas lógicas básicas y luego ir aumentando los tipos de ésta que había disponibles. Viendo si funcionaban correctamente y se correspondían con la tabla de verdad o algún tipo de comportamiento anómalo. Tras ciertas reparaciones, empecé a probar casos particulares, así como limitar el tipo de puertas a dos ó tres casos. De esta manera vería de una manera más ligera el comportamiento del algoritmo genético a través del fitness.

Comprobé que podríamos hacer una evaluación adecuada si priorizamos primeramente que los valores de salida concuerden, creando una penalización de 100 puntos unitarios en el valor de ésta. Además, por cada puerta lógica que modifiquemos le sumaremos otro punto más. De esta manera, lo que intentaremos será crear una acción para sacar el mayor rendimiento posible. De esta manera obtendríamos este método de puntuación para los distintos cromosomas:

$$Fitness = 100 \cdot N.^{o}\,salidas\ diistintas + N.^{o}\,puertas\ cambiadas$$

Al empezar con tamaños superiores y con mayor variedad de puertas lógicas pude percatarme de que ciertos valores variables tenían una gran repercusión sobre los distintos resultados a obtener. Por lo que fui probando datos y anotando sus susodichos resultados resultados:

- Las distintas probabilidades encontradas en los problemas deben tener un valor mediano, entre el 30 y el 80%. Estas variables son: probabilidad de cruce, de mutación de cromosoma y de genoma. Comparando con otros proyectos o investigaciones, vi que mis valores (obtenidos de manera puramente manuales) no estaban muy alejados de los que otros indican. En este caso, ha sido por el TFG de Rodrigo Barajas Fernández (páginas 17 y 18. Apartado 1.3.4). [5]
- También encontré coherente que el tamaño de una población inicial de un algoritmo genético es proporcional a la probabilidad de obtener un mejor resultado. De esta manera, conseguiríamos mayor diversidad gracias a los cruces y las respectivas mutaciones.

Tras las deducciones y aclaraciones fui probando diferentes circuitos con los que comprobé sus resultados de manera satifactoria:

- Comprobar que las puertas rotas no son cambiadas por el programa al finalizar y así obtener resultados coherentes. Obligamos a que las última capa sea todo puertas rotas, por lo que son inmovibles. Además, queremos hacer que la salida sea todo 1s, lo cual provocará que haya un valor de fitness de 100·Nº columnas (ya que las puertas rotas devuelven automáticamente 0).
  - Circuito 3x3
  - Todas las puertas disponibles

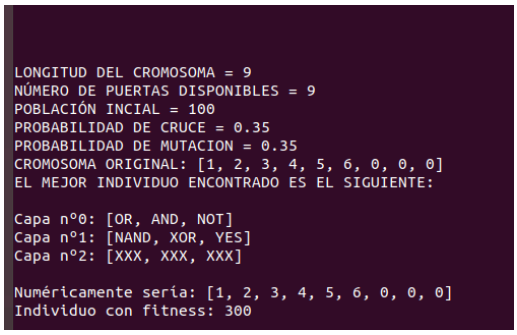

Figura 1: Prueba nº1 del proyecto de circuitos: Funcionamiento correcto de las puertas rotas

- Comprobar el funcionamiento del algoritmo genético usando únicamente dos puertas lógicas (OR y AND). Viendo si con poca capacidad de maniobra refiriéndome a la poca diversidad de puertas lógicas, podíamos converger a una solución.
  - Circuito 3x3
  - Dos puertas disponibles.

```
LONGITUD DEL CROMOSOMA = 9
NÚMERO DE PUERTAS DISPONIBLES = 2
POBLACIÓN INCIAL = 100
PROBABILIDAD DE CRUCE = 0.4
PROBABILIDAD DE MUTACION = 0.4
CROMOSOMA ORIGINAL: [1, 2, 1, 2, 1, 2, 1, 2, 1]
EL MEJOR INDIVIDUO ENCONTRADO ES EL SIGUIENTE:

Capa nº0: [OR, OR, OR]
Capa nº1: [OR, OR, OR]
Capa nº2: [OR, OR, OR]

Numéricamente sería: [1, 1, 1, 1, 1, 1, 1, 1, 1]
Individuo con fitness: 4


carlos@carlos-VirtualBox:~/Escritorio/circuitosIA$
```

Figura 2: Prueba Nº2: Funcionamiento del AG con un número pequeño de puertas.

- Comprobar el funcionamiento del algoritmo genético con todas las puertas lógicas obligatorias (AND, OR, NOT, NAND, XOR)
  - Circuito 5x5
  - 5 puertas disponibles.

```
LONGITUD DEL CROMOSOMA = 25
NÚMERO DE PUERTAS DISPONIBLES = 5
POBLACIÓN INCIAL = 100
PROBABILIDAD DE CRUCE = 0.4
PROBABILIDAD DE MUTACION = 0.4
CROMOSOMA ORIGINAL: [1, 2, 0, 3, 4, 5, 3, 2, 1, 2, 3, 4, 1, 0, 3, 1, 3, 5, 3, 0, 1, 0, 3, 5, 4]
EL MEJOR INDIVIDUO ENCONTRADO ES EL SIGUIENTE:

Capa nº0: [OR, AND, XXX, NOT, NAND]
Capa nº1: [AND, NOT, AND, OR, AND]
Capa nº2: [XXX, NAND, OR, XXX, XXX]
Capa nº3: [OR, NOT, AND, NOT, XXX]
Capa nº4: [NOT, XXX, NOT, NOT, NAND]

Numéricamente sería: [1, 2, 0, 3, 4, 2, 3, 2, 1, 2, 0, 4, 1, 0, 0, 1, 3, 2, 3, 0, 3, 0, 3, 3, 4]
Individuo con fitness: 106


VALORES DE ENTRADA = [1, 0, 1, 0, 1]
VALORES DE SALIDA = [1, 1, 1, 1, 1]
carlos@carlos-VirtualBox:~/Escritorio/circuitosIA$
```

Figura 3: Pruena Nº3: Correcto funcionamiento con las puertas lógicas obligatorias.

- Comprobar que el algoritmo genético funciona correctamente con todas las puertas lógicas disponibles(Las anteriores nombradas junto a BUFFER, NOR y XNOR).
  - Circuito 6x6
  - 8 puertas disponibles

```
LONGITUD DEL CROMOSOMA = 36
NÚMERO DE PUERTAS DISPONIBLES = 9
POBLACIÓN INCIAL = 100
PROBABILIDAD DE CRUCE = 0.4
PROBABILIDAD DE MUTACION = 0.4
CROMOSOMA ORIGINAL: [0, 1, 2, 3, 4, 6, 7, 8, 5, 2, 6, 9, 0, 1, 2, 3, 4, 5, 5, 6, 0, 1, 3, 2, 5, 4, 5, 6, 9, 8, 7, 4, 1, 0, 0, 0]
EL MEJOR INDIVIDUO ENCONTRADO ES EL SIGUIENTE:

Capa nº0: [XXX, OR, AND, NOT, NAND, YES]
Capa nº1: [AND, NOR, XNOR, AND, YES, XOR]
Capa nº2: [XXX, OR, AND, NOT, NAND, XOR]
Capa nº3: [NOR, YES, XXX, OR, NOT, OR]
Capa nº4: [XNOR, NAND, XOR, YES, NOT, NOR]
Capa nº5: [XNOR, NAND, OR, XXX, XXX, XXX]

Numéricamente sería: [0, 1, 2, 3, 4, 6, 2, 8, 7, 2, 6, 5, 0, 1, 2, 3, 4, 5, 8, 6, 0, 1, 3, 1, 7, 4, 5, 6, 3, 8, 7, 4, 1, 0, 0, 0]
Individuo con fitness: 7


VALORES DE ENTRADA = [1, 0, 0, 1, 1, 1]
VALORES DE SALIDA = [1, 1, 1, 0, 0, 0]
carlos@carlos-VirtualBox:~/Escritorio/circuitosIA$
```

Figura 4: Prueba Nº4: Resultado correcto con todas las puertas disponibles en un circuito 6x6

- Comprobamos ahora la influencia que tiene la población inicial en el mismo circuito que hemos usado en la prueba nº3. En este caso, usaremos un valor de población muy pequeño: 10. De esta manera podremos comprobar si con un valor mayor (como el de antes: 100) obtenemos o no mejores resultados.
  - Circuito 6x6
  - 4 puertas disponibles
  - Población inicial = 10

```
LONGITUD DEL CROMOSOMA = 25
NÚMERO DE PUERTAS DISPONIBLES = 9
POBLACIÓN INCIAL = 10
PROBABILIDAD DE CRUCE = 0.4
PROBABILIDAD DE MUTACION = 0.4
CROMOSOMA ORIGINAL: [1, 2, 0, 3, 4, 5, 3, 2, 1, 2, 3, 4, 1, 0, 3, 1, 3, 5, 3, 0, 1, 0, 3, 5, 4]
EL MEJOR INDIVIDUO ENCONTRADO ES EL SIGUIENTE:

Capa nº0: [OR, AND, XXX, AND, OR]
Capa nº1: [OR, NOT, YES, NOT, XXX]
Capa nº2: [NAND, NOR, OR, XXX, OR]
Capa nº3: [XNOR, XXX, OR, YES, XXX]
Capa nº4: [NOR, XXX, NAND, YES, NAND]

Numéricamente sería: [1, 2, 0, 2, 1, 1, 3, 6, 3, 0, 4, 8, 1, 0, 1, 7, 0, 1, 6, 0, 8, 0, 4, 6, 4]
Individuo con fitness: 116


VALORES DE ENTRADA = [1, 0, 1, 0, 1]
VALORES DE SALIDA = [1, 1, 1, 1, 1]
carlos@carlos-VirtualBox:~/Escritorio/circuitosIA$
```

Figura 5: Prueba Nº5: Empleo de una población inicial menor dando como resultado un cromosoma peor.

En la prueba anterior obtuvimos un fitness de valor 106 debido a que tenemos una puerta rota en el final del circuito y siempre devuelve 0 cunado en dicha casilla especialmente necesitamos un 1. Exceptuando ese apartado, se realizaron sólo 6 cambios en el cromosoma respecto al original. Mientras que en esta prueba el resultado ideal obtenido era un cromosoma de valor fitness = 116. En este caso podemos deducir que una mayor población nos daría mayor diversidad y, por tanto, mayor probabilidad de obtener un resultado favorable.

- En esta prueba comprobamos la diferencia que hay entre usar una probabilidad pequeña y otra mediana para el cruce entre cromosomas. Tomando como referencia el circuito de la prueba cuarta (obtuvimos un cromosoma de fitness = 7), modificaremos únicamente esta variable (siendo anteriormente 0’4) siendo su nuevo valor 0’1.
  - Circuito 6x6
  - 8 puertas disponibles.
  - Probabilidad de cruce = 0’1

```
LONGITUD DEL CROMOSOMA = 36
NÚMERO DE PUERTAS DISPONIBLES = 9
POBLACIÓN INCIAL = 100
PROBABILIDAD DE CRUCE = 0.1
PROBABILIDAD DE MUTACION = 0.4
CROMOSOMA ORIGINAL: [0, 1, 2, 3, 4, 6, 7, 8, 5, 2, 6, 9, 0, 1, 2, 3, 4, 5, 5, 6, 0, 1, 3, 2, 5, 4, 5, 6, 9, 8, 7, 4, 1, 0, 0, 0]
EL MEJOR INDIVIDUO ENCONTRADO ES EL SIGUIENTE:

Capa nº0: [XXX, NOT, AND, XNOR, NOR, YES]
Capa nº1: [XOR, NOR, XOR, OR, YES, XNOR]
Capa nº2: [XXX, OR, AND, XNOR, NAND, XOR]
Capa nº3: [NOT, YES, XXX, OR, NOT, NOR]
Capa nº4: [NAND, NAND, XOR, YES, NAND, NOR]
Capa nº5: [XNOR, AND, OR, XXX, XXX, XXX]

Numéricamente sería: [0, 3, 2, 7, 8, 6, 5, 8, 5, 1, 6, 7, 0, 1, 2, 7, 4, 5, 3, 6, 0, 1, 3, 8, 4, 4, 5, 6, 4, 8, 7, 2, 1, 0, 0, 0]
Individuo con fitness: 12


VALORES DE ENTRADA = [1, 0, 0, 1, 1, 1]
VALORES DE SALIDA = [1, 1, 1, 0, 0, 0]
carlos@carlos-VirtualBox:~/Escritorio/circuitosIA$
```

Figura 6: Prueba Nº6: La probabilidad de cruce se ha reducido a 0'1. El valor resultante ha sido peor que el anterior.

En la prueba anterior obtuvimos también un fitness adecuado, con valor de 7. En este, se ve diferenciado por tener un valor de 12. Siendo así una variación del 71’42% respecto al primer valor.

- En esta prueba comprobamos la diferencia que hay entre usar una probabilidad pequeña y otra mediana para la mutación de un cromosoma. Tomando como referencia el circuito de la prueba cuarta (obtuvimos un cromosoma de fitness = 7), modificaremos únicamente esta variable (siendo anteriormente 0’4) siendo su nuevo valor 0’1.
  - Circuito 6x6
  - 8 puertas disponibles
  - Probabilidad de mutación de cromosoma = 0’1

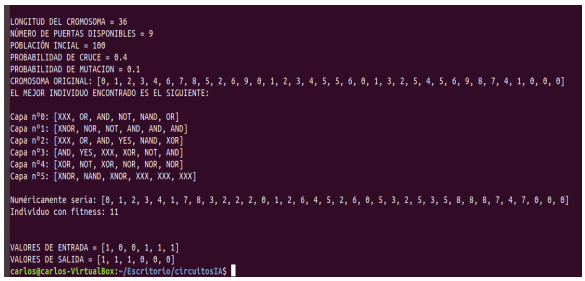

Figura 7: Prueba Nº7: La probabilidad de mutación en cromosoma se ha reducido a 0'1. El valor resultante ha sido peor que el anterior.

Hemos podido comprobar que el nuevo valor fitness es 11, también superior al obtenido anteriormente, pero esta vez una variación del 57’14%. Vemos otra vez que estos valores tan bajos en los porcentajes repercuten de manera negativa en la búsqueda de mínimos en la función fitness.

- En esta prueba comprobamos la diferencia que hay entre usar una probabilidad pequeña y otra mediana para la mutación de un genoma. Tomando como referencia el circuito de la prueba cuarta (obtuvimos un cromosoma de fitness = 7), modificaremos únicamente esta variable (siendo anteriormente 0’4) siendo su nuevo valor 0’1.
  - Circuito 6x6
  - 8 puertas disponibles
  - Probabilidad de los genomas = 0’1

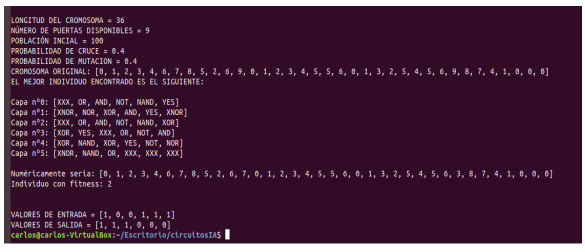

Figura 8: hgfd

Aquí tenemos un caso particular, al menos así lo deduje. Al disminuir la probabilidad del cromosoma esperaba tener un valor de fitness mayor del obtenido en la cuarta prueba. Sin embargo, aquí encontramos un valor mucho menor, fitness = 2. Aproximadamente un 28’57% del primer resultado. De esta manera, el circuito sólo necesitaría dos cambios en las puertas lógicas para el resultado deseado.

- En esta prueba comprobamos la importancia que tiene el número de iteraciones dentro del algoritmo genético. Al haber usado siempre hemos usado un número estándar (100), no hemos podido percibir qué ocurre con el fitness si éste varía. Así que cambiamos su valor actual por el de 10. Usando nuevamente el circuito 6x6 de las pruebas anteriores.
  - Circuito 6x6
  - 8 puertas disponibles.
  - Iteraciones de AG = 10

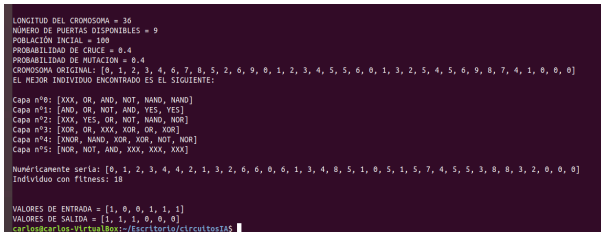

Figura 9: Prueba Nº9: Se ha reducido hasta 10 el número de iteraciones del algoritmo. Vemos que el resultado es peor del obtenido en la primera ronda.

Aquí encontramos de nuevo unos resultados esperados. El valor fitness de este cromosoma (18) resulta ser mayor que el de la prueba original (7). Superando por más del doble su valor, para ser exactos es un 257’14%. Así que deducimos que un número de iteraciones pequeño perjudica al algoritmo por no dar la suficiente carga a la creación mediante cruce o mutaciones de los nuevos cromosomas.

- Mi última prueba se basa en la potencia de nuestro programa. Hemos creado un circuito del máximo tamaño: 20x10. Además los números de población e iteración son también cuantiosos. De esta manera conseguiríamos un dato cercano a la realidad deseada.
  - Circuito 20x10
  - 8 puertas disponibles
  - Población inicial = 3000
  - Iteraciones AG = 1540
  - Probabilidad de cruce = 0’6
  - Probabilidad de mutación de cromosoma= 0’65
  - Probabilidad de mutación de genoma = 0’67

La valoración de fitness que tenemos es de 241 provocado por una entradas rotas en las salidas que no deja mostrar dos 1's que eran necesarios. Exceptuando eso, tenemos 141 cambios en las puertas del circuito de un total de 200 puertas lógicas (exceptuando que hay 39 puertas rotas). Así que vemos que no es tan efectivo, hasta un 87% de las puertas han sido cambiadas. También el proceso de cálculo fue muy costoso, aproximadamente finalizó en 30 minutos.

Todas estas pruebas serán incluidas en pruebasMostradas.py para el uso y comprobación de resultados por los tutores si ellos lo viesen necesario.

V. Conclusiones

Tras la finalización del proyecto he concluido que algoritmo genético funciona casi a la perfección para circuitos de tamaño pequeño y mediano.

Al ir realizando varias pruebas, he visto que el valor aleatorio de algoritmo genético y su decisión directa excluye numerosos cromosomas que inicialmente no son tan válidos como otros. Buscando de manera indiscriminada únicamente valores mínimos (o máximos en otros tipos de problemas) locales. Haciendo así la búsqueda más rápida.

Esto podríamos solventarlo si empleáramos el método de enfriamiento natural de los cristales en la naturaleza.[4] Esto es conocido en informática como 'enfriamiento simulado'. Consiguiendo gracias a sus pausas y bajadas de temperatura graduales obtener cromosomas que inicialmente no son tan buenos pero sí podrían alcanzar más tarde mínimos globales. De esta manera no excluimos opciones para problemas de un tamaño superior.

Referencias

[1] Definición de puertas lógicas. https://es.wikipedia.org/wiki/Puerta_l%C3%B3gica

[2] Definición utilizada de algoritmos genéticos junto a la cita de jhon koza: http://eddyalfaro.galeon.com/geneticos.html

[3] Api deap

[4] Profesor Víctor Yepes Viqueras. Universidad Politécnica de Valencia. En 'Optimización Heurística Mediante Cristalización Simulada'. https://www.youtube.com/watch?v=wtw_B_3lrjE

[5] TFG Rodrigo Barajas Fernández. http://oa.upm.es/45666/1/TFG_RODRIGO_BARAJAS_FERNANDEZ.pdf

[6] Algunos pequeños fragmentos de texto han sido reutilizados de mi anterior trabajo de Inteligencia Artificial 'Draw my graph'.