Proyecto Análisis de Algoritmos
Algoritmo Solucionador de Nonogramas

Juan Camilo Chafloque Mesia
Laura Mariana Jiménez Jiménez
Julio Andres Mejía Vera

4 de diciembre de 2020

Resumen

En este documento se realizará el análisis, el diseño y el desarrollo del algoritmo **Nonogram Solver** el cual - a partir de un archivo de texto - pretende resolver el nonograma propuesto haciendo uso de optimización, búsqueda y aprendizaje inspirados en los procesos de evolución, proceso comúnmente conocido como ≪Algoritmo genético≫. Esto con el fin de encontrar la configuración del tablero más apropiada según las reglas propuestas.

1. Análisis y diseño del problema

1.1. Análisis

Un Nonogram es uno de los juegos de lógica más populares en Japón y Holanda. Consiste en una cuadrícula rectangular inicialmente en blanco con pistas a la izquierda de cada fila y arriba de cada columna. El objetivo es pintar celdas en cada fila y columna para que su longitud y secuencia correspondan a las pistas, además, hay al menos un cuadrado vacío entre un conjunto de celdas adyacentes previamente pintados. Normalmente el resultado de celdas llenas forma una imagen.

El problema expresado de manera informal, se busca desarrollar un algoritmo que actué como jugador y resuelva un nonograma teniendo en cuenta las reglas inciales del juego. Formalmente se puede decir que el objetivo principal es diseñar y construir un algoritmo que resuelva un nonograma haciendo uso de un algoritmo genético. Para esto, se cuenta con dos secuencias iniciales que serán llamadas reglas, las cuales ayudaran a decidir si la solución generada es válida o se debe seguir creando generaciones nuevas de cromosomas dada la población actual, estas secuencias son:

$$R = \langle r_1, r_2, ..., r_m \rangle$$
$$C = \langle c_1, c_2, ..., c_n \rangle$$

Cada elemento de la secuencia define la cantidad de celdas por bloque que deben ser pintadas en las filas (R) o las columnas (C) dependiendo de cuál sea la regla evaluada, en caso de haber más de un bloque, cada posición contendrá una secuencia con la cantidad de elementos a pintar en los bloques adicionales (entiéndase bloque como una secuencia contigua de celdas de manera vertical u horizontal), entonces las reglas quedan definidas como:

$$rows = \langle [r_1^1], [r_1^2], [r_1^3, r_2^3], ..., r_m^p] | r \in \mathbb{N} \rangle$$
$$cols = \langle [c_1^1], [c_1^2], [c_1^3, c_2^3], ..., c_n^q] | c \in \mathbb{N} \rangle$$

Teniendo en cuenta lo anterior, se debe buscar la configuración del tablero solución más acertado, representado como la matriz M de tamaño mxn con las celdas pintadas según la validación realizada a partir de las reglas previamente definidas, por lo que:

- $n \in \mathbb{N} > 0$ es la cantidad de filas.
- $m \in \mathbb{N} > 0$ es la cantidad de columnas.
- $c_i^p \in \mathbb{N}$ es la cantidad de celdas pintadas en la i-ésima columna.
- $r_j^q \in \mathbb{N}$ es la cantidad de celdas pintadas en la j-ésima fila

Inicialmente el proceso empieza con una población de N cromosomas. El tamaño N es definido por el usuario al momento de iniciar el programa. Dicha población estará compuesta de cromosomas.Un cromosoma es en donde se codifica la información de cada solución, en donde cada solución será una lista de tamaño mxn que definen una posible solución propuesta al problema que el algoritmo genético esta buscando resolver. Inicialmente gen (índice en la lista) de cada solución se generará de manera aleatoria para obtener una configuración inicial en la primera generación de cromosomas. Estos genes son configurados con los valores booleanos de $true$ y $false$ y representan en la celda si se encuentra pintada o no de la siguiente manera:

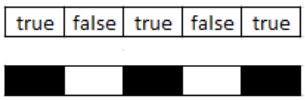

Figura 1: Configuración de cromosomas.

Luego, estas soluciones iníciales irán mutando y generando otras nuevas que serán validadas a partir del cálculo del fitness o función objetivo, el cual se vale de las reglas de entrada para realizarlo. El proceso se repetirá hasta generar un cromosoma con una configuración correcta, sin embargo, puede existir el caso en el que las reglas no definan una configuración válida de juego, en ese caso se genera una solución cercana después de cierto número de iteraciones, pues retorna el cromosoma con la función objetivo más cercana a cero, el cual sería el cromosoma con la solución más cercana a la correcta.

1.2. Diseño

Con las precisiones realizadas en el análisis anterior, podemos definir el diseño de un algoritmo que solucione el problema mencionado teniendo en cuenta las reglas que definen las celdas pintadas o los espcios y los elementos iniciales necesarios para el desarrollo de un algoritmo genético.

- **Entradas:**
  1. Una secuencia $rows = \langle [r_1^1], [r_1^2], [r_1^3, r_2^3], ..., r_m^p] \rangle$ donde $r_i^p \leq m \wedge r_i^p \in \mathbb{N} > 0$
  2. Una secuencia $cols = \langle [c_1^1], [c_1^2], [c_1^3, c_2^3], ..., c_n^q] | c \in \mathbb{N} \rangle$ donde $c_j^q \leq n \wedge c_j^q \in \mathbb{N} > 0$
  3. Valor n que representa la población inicial del algoritmo donde $n \in \mathbb{N}$

4. Valor p que representa la probabilidad de mutación de las generaciones siguientes donde $n \in \mathbb{N}$. La probabilidad de mutación hace referencia a si la solución de un cromosoma se va a alterar o afectar entre generaciones.
5. Valor e que representa el porcentaje de elitismo donde $n \in \mathbb{N}$, El elitismo hace referencia al porcentaje de cromosomas de la generación pasada, que tienen una probabilidad de 1 de permanecer en la siguiente generación.

- **Salidas:**

1. El mejor cromosoma de la última generación, el cual tendrá una matriz solución $M : \mathbb{T}^{mxn}$, con la cual se generará el archivo PGM.

2. Algoritmos

Como se mencionó anteriormente, se utilizará un algoritmo genetico [1][2] para la solución de un nonograma. Para dicha solución se siguió el estudio realizado por [3], en el cual los autores proponen un algoritmo genetico para la solución de nonogramas. Además, se adapto y se tomó como guía el siguiente trabajo: https://github.com/BryanCruz/nonogen

El algoritmo propuesta empieza con la inicialización de la población principal, en el cúal dado un valor de entrada N, se crearan N cromosomas cada uno con una solución inicial aleatoria. Los parámetros de entrada son R, las reglas del nonograma a resolver, r, la cantidad de columnas que tiene el nonograma, c, la cantidad de columnas que tiene el nonograma, y por último, N, el tamaño de la población inicial del algoritmo genetico.

Algorithm 1 Inicialización de los cromosomas para el algoritmo genetico.

```
1:  procedure INITSOLUTIONS(R, r, c, N)
2:      S ← []
3:      for i ← 0 to N do
4:          Genes ← []
5:          for j ← 0 to r * c do
6:              v ← RANDOM(1)
7:              if v ≤ 0,5 then
8:                  Genes ← Genes + True
9:              end if
10:             if v > 0,5 then
11:                 Genes ← Genes + False
12:             end if
13:         end for
14:         C = Chromosome(Gene, R, r, c)
15:         S ← S + C
16:     end for
17:     return S
18: end procedure
```

Una vez se tiene la población inicial y los cromosomas aleatorios establecidos, se procede a empezar con el algoritmo genetico. Los parametros a la función son los siguientes:S, la lista de cromosomas con la configuración inicial, R, las reglas del nonograma a resolver, r, la cantidad de columnas que tiene el nonograma,

c, la cantidad de columnas que tiene el nonograma, y por último, N, el tamaño de la población inicial del algoritmo genetico, m, la probabilidad de mutación y e, el porcentaje de elitismo.

Algorithm 2 Algoritmo genetico

```
1: procedure GENETICALGORITHM(S, R, r, c, N, m, e)
2:     iterations ← 1
3:     while converge(S) = False & iterations ≤ 2000 do
4:         C ← crossover(S, N, r * c, r, c)
5:         M ← mutation(C, m, R, r, c)
6:         S ← elitism(S, M, e, N, R, r, c)
7:         iterations ← iterations + 1
8:     end while
9:     return S[0].solution
10: end procedure
```

Algorithm 3 Procedimiento auxiliar para verificar si la mejor solución es correcta

```
1: procedure CONVERGE(S)
2:     if S[0].fitness = 0 then
3:         return True
4:     end if
5:     return False
6: end procedure
```

Para poder solucionar el algoritmo genetico y poder encontrar la mejor solución al nonograma, es necesario realizar unos pasos definidos.

El primer paso es realizar los cruces entre los cromosomas. En este paso, se opera un cromosoma para generar dos descendientes, en donde se combinan carácteristicas del cromosoma padre.

Algorithm 4 Procedimiento en donde se realiza el cruce entre los cromosomas para generar los descendientes de la próxima generación

```
1: procedure CROSSOVER(S, N, P, r, c)
2:     C ← []
3:     SORT(P)
4:     for i ← 0 to N/2 do
5:         C_1 ← []
6:         C_2 ← []
7:         F_1 ← RANDOM(S)
8:         F_2 ← RANDOM(S)
9:         for i ← 0 to |F_1| do
10:            v ← RANDOM(1)
11:            if v ≤ 0,5 then
12:                C_1 ← C_1 + F_1.solution[i]
13:                C_2 ← C_2 + F_2.solution[i]
14:            end if
15:            if v > 0,5 then
16:                C_1 ← C_1 + F_2.solution[i]
17:                C_2 ← C_2 + F_1.solution[i]
18:            end if
19:        end for
20:        C ← C + Chromosome(C_1, R, r, c)
21:        C ← C + Chromosome(C_2, R, r, c)
22:    end for
23:    return C
24: end procedure
```

Luego de obtener los cruces, se realiza el proceso de mutación. Esto significa que se modifica al azar una parte de cada cromosoma.

Algorithm 5 Procedimiento en donde se realiza la mutación de los cromosomas

```
1: procedure MUTATION(C, m, R, r, c)
2:     M ← []
3:     for i ← 0 to |C| do
4:         if RANDOM(1) < m then
5:             p ← RANDOM(0, |C[i].solution|)
6:             if C[i].solution[p] = True then
7:                 C[i].solution[p] = False
8:             end if
9:             if C[i].solution[p] = False then
10:                C[i].solution[p] = True
11:            end if
12:        end if
13:        M ← M + C[i]
14:    end for
15:    return M
16: end procedure
```

El último paso es el elitismo, esto es, escoger y seleccionar los mejores individuos o cromosomas para conformar la población de la siguiente generación.

Algorithm 6 Procedimiento en donde se realiza la selección de cromosomas para la siguiente generación

```
1: procedure ELITISM(S, M, e, N, R, r, c)
2:     U ← S + M
3:     G ← []
4:     B ← []
5:     W ← []
6:     SORT(U)
7:     p ← e * N
8:     for i ← 0 to p do
9:         B ← B + U[i]
10:    end for
11:    for i ← p + 1 to |U| do
12:        W ← W + U[i]
13:    end for
14:    G ← B + W
15:    return G
16: end procedure
```

Toca tener en cuenta que el principal valor de comparación para ver si la solución es correcta o más acertada que otra, es el valor de la función objetivo o fitness de cada cromosoma. Como lo que se quiere en un nonograma es que la cantidad de celdas diferentes entre una solución y la solución real sea de cero, los mejores cromosomas en cada generación, serán los cromosomas que tengan un fitness cercano o igual a cero.

2.0.1. Complejidad

Los algoritmos genéticos, al tener mucha variabilidad y soluciones iniciales aleatorias, pueden considerarse estocásticos y la complejidad del algoritmo va a depender de los parámetros de entrada como lo son el tamaño de la población, la cantidad de generaciones que se van a necesitar para solucionar el algoritmo y el tamaño del nonograma. Dado estos parámetros de entrada, la complejidad del algoritmo implementado es igual a: $\Theta(g * n * m)$.

Donde g es el número de generaciones, n es el tamaño de la población inicial de cromosomas y m es el tamaño de la solución de cada cromosoma, que en este caso $m = r * c$, siendo r el número de filas que tiene el nonograma y c, el número de columnas que tiene el nonograma.

2.0.2. Invariante

En la primera posición de la lista con la generación actual de cromosomas se encontrará siempre el cromosoma que contiene el mejor fitness o la mejor solución. Esto está fuertemente relacionado con la función objetivo o fitness del cromosoma, pues la mejor solución actual es aquella que más se acerca a 0 y cada cromosoma tendrá el fitness asociado a su solución actual.

2.0.3. Especificaciones de la implementación

El algoritmo fue desarrollado en Python y está conformado por dos archivos principales nonogram.py y utils.py. Para compilarlo y ejecutarlo se necesita Python3 a través del siguiente comando:

python3 nonogram.py nPopulation probMutation elitsmPercentage input_name.txt output_name.pgm

En el archivo nonogram.py se define la clase Game donde se encuentra el tablero de juego y la cantidad de filas y columnas del mismo, la clase Rule que contiene las listas que guardan los valores con los que se realizan las validaciones y la clase Chromosome la cual guarda el vector solución especifico. Se encuentran también las funciones previamente mostradas que manejan la inicialización de la población y todo el proceso genético que direcciona el comportamiento del algoritmo como la mutación, el elitismo parcial y el cruce.

En el archivo utils.py está conformado por la función que calcula el fitness de cada cromosoma, la lectura del archivo de reglas inicial, la impresión del tablero para las pruebas y la escritura del archivo en formato PGM que se espera como salida del algoritmo.

Además, se encuentran un total de diez archivos adicionales que contienen diferentes casos de prueba donde el tamaño del tablero del juego va variando, esto con el fin de probar el algoritmo con diferentes escenarios en los cuales su solución será visualizada a través de la imagen en formato PGM.

3. Casos de Prueba y Resultados

Para poder probar la efectividad y el comportamiento del algoritmo con diferentes casos se diseñaron y probaron 10 juegos de diferentes tamaños. Para cada una de las pruebas se midió el tiempo en el que el algoritmo generaba solución, la cantidad de generaciones necesarias y si género o no el tablero esperado.

Input	Solución	Celdas	Intentos	Tiempo	Generaciones
Input_3x3	Correcta	9	1	0.034 s	5
Input_4x4	Correcta	16	1	0.135 s	20
Input_4x7	Correcta	28	2	0.331 s	48
Input_5x5	Correcta	25	1	0.121 s	17
Input_6x6	Correcta	36	1	0.285 s	38
Input_6x8	Correcta	48	2	0.754 s	89
Input_7x7	Correcta	49	1	0.542 s	64
Input_8x8	Correcta	81	1	0.893 s	97
Input_9x8	Correcta	72	1	0.880 s	91
Input_10x10	Correcta	100	3	0.992 s	313

Tabla 1. Tabla de resultados con diferentes pruebas

Como se pudo ver en la tabla de resultados, al tener un incremento en las dimensiones del nonograma, que a su vez incrementa el número de celdas a evaluar, el tiempo, y el número de generaciones necesarias incrementan en la mayoría de los casos. Se pudo ver también, que el algoritmo propuesto solucionó los 10 casos de prueba que se tenían, que eran principalmente nonogramas de pequeñas dimensiones.

Debido a que el algoritmo genético genera unas soluciones iniciales de forma aleatoria, puede llegar el caso, que al hacer las iteraciones y generando las nuevas generaciones, los cromosomas entren en un óptimo local, lo cual significa que la solución converge entre iteraciones sin llegar a la solución óptima (0), por lo que

el algoritmo queda en un ciclo infinito si no se tiene otra condición de salida. Dado esto, se dieron los casos en los que se entró en un óptimo local al hacer los casos de prueba y se tuvieron que realizar varios intentos antes de poder encontrar una solución correcta.

Referencias

[1] Arranz de la Peña, J. y Parra Truyol, A., *Algoritmos genéticos*, Universidad Carlos III., 2007.

[2] Pose, M. G., «Introducción a los algoritmos genéticos», *Departamento de Tecnologías de la Información y las Comunicaciones Universidad de Coruña.*, 2000.

[3] Yu, C. H., Lee, H. L. y Chen, L. H., *An efficient algorithm for solving nonograms*, Applied Intelligence, 35(1), 18-31., 2011.