RESOLVIENDO SUDOKUS MEDIANTE ALGORITMOS GENÉTICOS

En este proyecto, resolverás un tipo de rompecabezas denominado Sudoku utilizando algoritmos genéticos. Un Sudoku es un rompecabezas de 81 celdas donde cada fila y cada columna del rompecabezas contiene los dígitos del 1 al 9, al igual que cada "super-cuadrado" de 3 x 3 celdas. Algunas celdas ya están llenas, como se ve aquí:

1	9			2		5		8
	6	7					4	
		4	6	8	3		9	
3			7			2		9
			1			6		5
			5	9	8			4
4		5	8			9		6
2		6		4			5	1
9		1			6		7	

Por lo tanto, la tarea es rellenar las celdas vacías con los números adecuados números adecuados de forma que cada fila, columna y cuadrado de 3x3 contenga exactamente una vez cada número del 1 al 9. Se pueden encontrar más ejemplos de Sudokus aquí: https://sudoku.com/

Puedes implementar el algoritmo genético tú mismo o usar el framework JGAP: http://jgap.sourceforge.net/
Si decides usar JGAP, debes descargar jgap_3.6.3_full.zip del campus virtual. Este archivo contiene una versión de JGAP. En este caso, puedes empezar echando un vistazo al ejemplo de la mochila (knapsack) que se puede encontrar dentro de jgap_3.6.3_full.zip file. Debes añadir JGAP (*jgap.jar*) al proyecto Eclipse para usarla en tu código.

Tanto si usas JGAP como si implementas el algoritmo genético, debes diseñar tu propia función de fitness para resolver Sudokus.

Puedes usar también la librería QQWing, que permite solucionar y generar Sudokus: https://qqwing.com/
Puedes descargar el fichero qqwing-1.3.4.jar del campus virtual, que contiene la versión de QQWing que puedes usar para generar un puzle de Sudoku junto con su solución.

Debes escribir una aplicación de Java que muestre sus resultados por consola (salida de texto) y haga lo siguiente:

1. Generar una matriz 9x9 que sea un puzle de Sudoku válido. Esto puede hacerse con la librería QQWing, que puedes descargar del campus virtual. El puzzle y su solución deben ser mostrados por consola.

2. Ejecutar un algoritmo genético que intente resolver el puzle de Sudoku previamente generado. Por cada generación del algoritmo genético, la aplicación debe imprimir el fitness medio de la población, y el fitness del mejor individuo. El algoritmo genético puede ser de implementación propia, o implementado usando JGAP.
3. Una vez el algoritmo genético haya terminado, se debe imprimir un mensaje indicando si se ha encontrado la solución al puzle, y el mejor individuo de la última generación debe imprimirse.

Se debe generar un fichero de texto denominado output.txt que contenga la salida completa del programa

Todos los materiales (presentación de diapositivas, código fuente y archivo output.txt) deben enviarse a la tarea de campus virtual asociada.

Tarea opcional:

Puede mostrar la progresión del algoritmo genético gráficamente, mostrando cómo cambian el fitness medio de la población y el del mejor individuo conforme transcurren las generaciones. También se debe analizar cómo cambia el rendimiento del algoritmo genético a medida que cambian los parámetros del algoritmo genético. Puedes realizar esta tarea con cualquier herramienta que te permita generar gráficas: Matlab, Python, Excel, etc.

SUGERENCIAS:

Es conveniente que cada individuo de la población (es decir, cada cromosoma) esté compuesto por una secuencia de números que represente una posible solución al problema. Por ejemplo, para el siguiente Sudoku sin resolver:

1	9			2		5		8
	6	7					4	
		4	6	8	3		9	
3			7			2		9
			1			6		5
			5	9	8			4
4		5	8			9		6
2		6		4			5	1
9		1			6		7	

Cada celda vacía es un número que debe formar parte de la solución. Nombrando cada celda vacía según la fila y columna en la que están situadas:

1	9	A1	A2	2	A3	5	A4	8
B1	6	7	B2	B3	B4	B5	4	B6
C1	C2	4	6	8	3	C3	9	C4
3	D1	D2	7	D3	D4	2	D5	9
E1	E2	E3	1	E4	E5	6	E6	5
F1	F2	F3	5	9	8	F4	F5	4
4	G1	5	8	G2	G3	9	G4	6
2	H1	6	H2	4	H3	H4	5	1
9	I1	1	I2	I3	6	I4	7	I5

Así, la estructura del cromosoma para este ejemplo en concreto es como sigue, con 38 valores numéricos en total:

A1	A2	A3	A4	B1	B2	B3	B4	B5	B6	C1	C2	C3	C4	D1	D2	D3	D4	D5	E1	E2	E3	E4	E5	E6	G1	G2	G3	G4	H1	H2	H3	H4	I1	I2	I3	I4	I5

Hay 27 restricciones que cualquier solución válida debe respetar: 9 restricciones de fila, 9 restricciones de columna, y 9 restricciones de bloques 3x3. Es aconsejable asegurar que todos los individuos satisfagan de forma automática al menos uno de estos 3 tipos de restricciones, de forma que hay 3 posibilidades:

1. Asegurar el cumplimiento de las restricciones de fila.
2. Asegurar el cumplimiento de las restricciones de columna.
3. Asegurar el cumplimiento de las restricciones de bloques 3x3.

De ahora en adelante, se mostrará cómo proceder con la primera opción (asegurar el cumplimiento de las restricciones de fila).

Para generar la población inicial (por ejemplo de 20 individuos), se debe generar un primer individuo que satisfaga las restricciones de las 9 filas. Para hacerlo, para cada fila, debe completar los genes asociados a la fila con los dígitos faltantes de esa fila. En nuestro ejemplo, para la primera fila, los genes A1, A2, A3 y A4 deben asignarse a los dígitos 3, 4, 6 y 7, que son los dígitos que faltan para esa fila. Después de generar el primer individuo, se debe generar la población inicial permutando aleatoriamente los dígitos de cada fila. En nuestro ejemplo, para la primera fila, debes hacer una permutación aleatoria de los valores de los genes A1, A2, A3 y A4. De esta forma, todos los individuos de la población inicial cumplen las restricciones de 9 filas.

El operador de cruce (crossover) debe asegurar que si los dos progenitores cumplen todas las restricciones de fila, entonces cualquier descendiente también las cumpla. Esto se puede conseguir restringiendo los puntos de cruce en los cromosomas a los límites entre filas adyacentes. Por ejemplo, si tenemos estos dos progenitores:

A1	A2	A3	A4	B1	B2	B3	B4	B5	B6	C1	C2	C3	C4	D1	D2	D3	D4	D5	E1	E2	E3	E4	E5	E6	G1	G2	G3	G4	H1	H2	H3	H4	I1	I2	I3	I4	I5
A1'	A2'	A3'	A4'	B1'	B2'	B3'	B4'	B5'	B6'	C1'	C2'	C3'	C4'	D1'	D2'	D3'	D4'	D5'	E1'	E2'	E3'	E4'	E5'	E6'	G1'	G2'	G3'	G4'	H1'	H2'	H3'	H4'	I1'	I2'	I3'	I4'	I5'

Entonces los únicos puntos de cruce permitidos deben ser A4-B1, B6-C1, C4-D1, D5-E1, E6-G1, G1-H1, y H4-I1. Si se escoge el punto de cruce C4-D1, entonces dos posibles descendientes serían:

A1	A2	A3	A4	B1	B2	B3	B4	B5	B6	C1	C2	C3	C4	D1'	D2'	D3'	D4'	D5'	E1'	E2'	E3'	E4'	E5'	E6'	G1'	G2'	G3'	G4'	H1'	H2'	H3'	H4'	I1'	I2'	I3'	I4'	I5'
A1'	A2'	A3'	A4'	B1'	B2'	B3'	B4'	B5'	B6'	C1'	C2'	C3'	C4'	D1	D2	D3	D4	D5	E1	E2	E3	E4	E5	E6	G1	G2	G3	G4	H1	H2	H3	H4	I1	I2	I3	I4	I5

El operador de mutación debe asegurar que si un individuo cumple las restricciones de fila, el resultado de mutarlo también las cumpla. Esto se puede conseguir definiendo la

mutación como el intercambio de los valores numéricos de dos celdas que pertenezcan a una misma fila. Por ejemplo, si tenemos este individuo:

Podríamos mutarlo, por ejemplo, intercambiando los valores numéricos de las celdas B2 y B5, pues ambas están en la segunda fila. Pero no podríamos intercambiar los valores de D4 y E3 pues no pertenecen a la misma fila: D4 está en la cuarta fila, mientras que E4 está en la quinta fila.

La función de fitness puede definirse como la suma de la cantidad de valores numéricos únicos en cada columna y en cada bloque de 3x3 celdas. En este caso concreto no hace falta considerar las filas para la función de fitness puesto que si aseguramos que siempre se cumplen las restricciones de fila, siempre habrá exactamente 9 números distintos en cada fila. Esto significa que la función de fitness es la suma de 18 valores (9 columnas y 9 bloques de 3x3 celdas), en la que cada valor es un número entre 1 y 9. Por lo tanto, la función de fitness tendrá como resultado un número natural entre 18*1=18 y 18*9=162. Un individuo será una solución al puzle si y sólo si su fitness es 162.

Téngase en cuenta que la efectividad de la búsqueda genética dependerá de los diversos parámetros: tasas de mutación y cruce, cantidad de mutaciones, tamaño de población, cantidad de generaciones… Un puzle muy simple (por ejemplo, con como mucho 3 ó 4 celdas vacías por fila) será fácilmente solucionable con casi cualquier configuración, pero puzles de complejidad creciente requerirán de ajustar los parámetros del algoritmo genético.

En caso de usar JGAP:

- El cromosoma debe ser una clase que implemente la interfaz Ichromosome de JGAP.
- El operador de cruce debe implementarse como una subclase de la clase CrossoverOperator de JGAP.
- El operador de mutación debe implementarse como una subclase de la clase MutationOperator de JGAP.

Téngase en cuenta igualmente que JGAP no contempla que los cromosomas puedan cumplir restricciones internas, por lo que se debe poner especial cuidado en la generación de la población inicial, y en la implementación de los operadores de cruce y mutación para que cumplan las restricciones de fila. Si se implementan sin estas consideraciones, la búsqueda genética será dramáticamente menos efectiva.