Tarea Larga. Metaheurísticas Poblacionales

Implementación.

La implementación del algoritmo evolutivo se realizó por medio de dos clases, siguiendo el esquema del algoritmo para el 2DPP.

- Clase SudokuProblem: Es la clase que contiene el tablero del Sodoku sin resolver.
- Clase Sudoku: Se deriva de la clase Individual, contiene un miembro estático de la clase SudokuProblem, un valor para almacenar el fitness del individuo, y una matriz donde guardamos las penalizaciones que cada elemento (casilla) aporta al fitness del individuo. Incluye también los métodos generateRandom(), calculateFitness(), dependentMutation(), dependetCrossover() y localSearch(), además de modificaciones a los métodos derivados de la clase Individual.

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

Implementación por bloques

El constructor de la clase SudokuProblem recibe como parámetro el nombre del archivo donde esta guardada la instancia a resolver, y genera un arreglo bidimensional con cada uno de los valores leídos (en representación completa).

A continuación se describen las funciones mas importantes implementadas para la clase Sudoku:

- init(): recibe los parámetros de entrada del algoritmo para crear un objeto de la clase SudokuProblem, también establece en numero de objetivos (1) y el numero de variables (N*N=81).
- generateRandom(): Genera un individuo aleatorio, copia el contenido del tablero desde la clase SudokuProblem al individuo creado, y llena los espacios en ceros con una permutación de números aleatorios del 1 al 9, utilizando el método generateRandomVector().
- calculateFitness(): calcula el fitness de un individuo, se suma +1 a la variable objective si hay dos elementos en una misma fila o columna que se encuentren en conflicto, y +100 adicionales si uno de esos elementos es fijo, además actualiza la matriz con las penalizaciones que cada elemento en conflicto aporta al fitness.
- dependentMutation(): intercambia dos elementos en un mismo bloque con probabilidad pm, esto se hace para cada bloque del individuo.
- dependentCrossover(): realiza una cruza uniforme entre todos los bloques de dos individuos, la probabilidad de intercambiar un mismo bloque entre dos individuos es de 0.5.
- localSearch(): busca en la matriz de penalizaciones cual es el valor mas alto, y cuantas veces aparece ese valor, luego selecciona aleatoriamente uno de ellos e intercambia el elemento asociado a esa posición con otro del mismo bloque seleccionado al azar, si el fitness disminuye, el cambio se queda, si no, se regresa a como se encontraba anteriormente, esto se ejecuta 100 veces.

```
Algorithm1: LocalSearch
Input: k←0, kmax;
repeat:
        (pmax, ||pmax||) ←Penalty (Individual I);
        [ i, j ]←randomSelection (||pmax||);
        [ i, r ]←randomSelection (block i);
        swap ( j, r );
        calculateFitness( );
        if (improved == false);
                swap( r , j );
        k++;
until: k=kmax
```

Las demás funciones de la clase Sudoku se implementaron igual que en el algoritmo 2DPP.

En esta práctica se trabajo en la implementación del algoritmo MULTI_DYN descrito en el artículo "The Importance of Proper Diversity Management in Evolutionary Algorithms for Combinatorial Optimization", en el framework MiniMETCO proporcionado en la clase:

Algorithm 2 MULTI_DYN survivor selection scheme

1: CurrentMembers = Population ∪ Offspring
2: Best = Individual with best $f(x)$ in CurrentMembers
3: NewPop = { Best }
4: CurrentMembers = CurrentMembers - { Best }
5: **while** (|NewPop| < N) **do**
6: Calculate DCN of CurrentMembers, taking NewPop as reference
7: $D = D_I - D_I * \frac{T_{Elapsed}}{T_{End}}$
8: Penalize(CurrentMembers, D)
9: ND = Non-dominated individuals of CurrentMembers (without repetitions)
10: Selected = Randomly select an individual from ND
11: NewPop = NewPop ∪ Selected
12: CurrentMembers = CurrentMembers - {Selected}
13: **end while**
14: Population = NewPop

De acuerdo al artículo mencionado, el algoritmo esta basado en un paradigma memético Lamarkiano:

Algorithm 3 Lamarckian Memetic Algorithm

1: **Initialization**: Generate an initial population P_0 with N individuals. Assign $t = 0$.
2: **Local Search**: Perform a local search for every individual in the population.
3: **while** (not stopping criterion) **do**
4: **Evaluation**: Evaluate all individuals in the population.
5: **Mating selection**: Perform binary tournament selection on P_t in order to fill the mating pool.
6: **Variation**: Apply genetic operators to the mating pool to create a child population CP.
7: **Local Search**: Perform a local search for every individual in the offspring.
8: **Survivor selection**: Apply the replacement scheme to create P_{t+1}.
9: $t = t + 1$
10: **end while**

Resultados.

A continuación se muestra una tabla con los resultados obtenidos por el algoritmo para las diferentes instancias de Sudoku utilizadas, como se aprecia, el ratio de éxito fue del 100% en todos los casos, los parámetros utilizados fueron los mismos que se recomendaron en el artículo, se anexa una carpeta con las archivos de resultado obtenidos en cada ejecución (solo contienen el ultimo resultado):

Instancias	Ejecuciones	Ratio	distanceInit	pm	pc	Población
Easy1	30	100	20	0.01	0.5	100
Easy2	30	100	20	0.01	0.5	100
Easy3	30	100	20	0.01	0.5	100
Medium1	30	100	20	0.01	0.5	100
Medium2	30	100	20	0.01	0.5	100
Medium3	30	100	20	0.01	0.5	100
Hard1	30	100	20	0.01	0.5	100
Hard2	30	100	20	0.01	0.5	100
Hard3	30	100	20	0.01	0.5	100
Evil1	30	100	20	0.01	0.5	100
Evil2	30	100	20	0.01	0.5	100
Evil3	30	100	20	0.01	0.5	100
SD1	30	100	20	0.01	0.5	100
SD2	30	100	20	0.01	0.5	100
SD3	30	100	20	0.01	0.5	100
SS1	30	100	20	0.01	0.5	100
SS2	30	100	20	0.01	0.5	100
SS3	30	100	20	0.01	0.5	100
SS4	30	100	20	0.01	0.5	100
Film1	30	100	20	0.01	0.5	100
Film2	30	100	20	0.01	0.5	100
Inkala	30	100	20	0.01	0.5	100

Compilación/Ejecución.

Los ficheros Sudoku.h y Sudoku.cpp contienen la implementación de las clases y métodos descritos anteriormente, a continuación se muestra un ejemplo de la linea de ejecución (con directivas relativas) que utilizamos en esta practica:

```
./metcoSeq ./../../../Results/ ./ PlainText output.dat MonoGA2 Sudoku TIME
300 1 0 0 pm = 1 pc = 0.85 LocalSearchType = Always PopType = Fixed N = 50
SurvivalSelection = CentralPartOfFrontParentBinaryTournament MutationType =
Normal ! ../../plugins/problems/Sudoku/instances/SD3.txt $ NoOp +
MultiObjNearestIndDistThresholdPopPercentReference  pThresholdInit = 0
pThresholdEnd = 0 DUpdateType = Linear distanceInit = 0 distanceEnd = 0
```

Se anexa un script para ejecutar el programa en paralelo, el cual se utilizó para ejecutar el programa en El Insurgente, así como un taskfile de muestra.