FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Inteligencia Artificial 2014/2015

Relatório : TP1 – 2ª Fase
Algoritmo Genérico

Docente: Susana Nascimento de Almeida

Turno: 3

Relatório : TP1 – 2ª Fase

Grupo 2:

André Martins Lopes nº 41883

David de Oliveira Pinto Gago nº 41710

João Pedro Guita de Almeida nº 42009

Índice

Introdução

Este trabalho consiste na implementação, em java, de um algoritmo de procura local, para resolver o problema do caixeiro viajante. Para isso recorremos a um algoritmo genético, para obter uma solução aproximada ou mesmo a solução optima. Para usar este algoritmo temos de ter em atenção, a população, métodos de seleção, operadores de mutação, probabilidade de mutação, função de mérito de cada indivíduo, bem como definir características da população. Para o utilizador introduzir os parâmetros, fizemos uma interface, onde o utilizador pode definir a população inicial, a elite, o crossover, probabilidade de mutação e o número de gerações. Os resultados depois são mostrados num gráfico.

O que é o Algoritmo Genético

Um algoritmo genético é um algoritmo heurístico baseado no princípio da seleção natural e da evolução biológica teorizado em 1859 por Charles Darwin. O adjectivo "genético" deriva do facto de que as explicações modelo de Darwin e pelo facto de algoritmos genéticos implementarem mecanismos conceptualmente semelhantes às dos processos bioquímicos descobertos pela ciência. Podemos dizer que os algoritmos genéticos consistem em algoritmos que permitem a avaliação das soluções de partida, a recombinação de elementos que introduzem novos individuos, na tentativa de convergir para soluções ótimas. Essas técnicas são normalmente utilizadas para tateou para resolver problemas de otimização para os quais não há outros algoritmos. Apesar deste uso, dada a natureza intrínseca de um algoritmo genético, não há nenhuma maneira de saber com antecedência se ele vai realmente ser capaz de encontrar uma solução ideal para o problema considerado.

Operadores

Caso o utilizador não introduza valores, assumimos os seguintes valores por defeito:

```
GEN_CAP = 2000; // População inicial
DEFAULT_P_CROSSOVER = 0.6f; // Probabilidade de Crossover
DEFAULT_P_MUTATE = 0.01f; // Probabilidade de Mutação
ELITE = 50; // Número de individuos da Elite
```

Elite

A elite é fundamental no algoritmo genético pois é na elite que se encontram os melhores indivíduos da população. Ou seja, sempre que se gera uma nova geração os melhores indivíduos são sempre preservados. Assim, com a elite temos sempre a vantagem de que temos os melhores elementos de cada geração, melhorando a performance do algoritmo genético.

Crossover

O crossover serve para gerar uma nova geração de indivíduos, descendentes dos pais, através da recombinação dos genes. Seguindo o mesmo princípio de que, a cada geração que passa, a geração seguinte poderá ser melhor.

A probabilidade de cruzamento estabelece a frequência com que o crossover acontece.

Mutação

Após o crossover, cada indivíduo tem uma percentagem de mutação associada, podendo ou não mutar. A mutação altera a ordem dos genes, para que se consiga chegar próximo de uma solução ideal que de outra forma podia não ser possível.

Também a mutação é ajustada através da probabilidade, e tal como na probabilidade de Crossover, ajuda a diversificar os genes da população.

Analise e resultados

5 Pontos

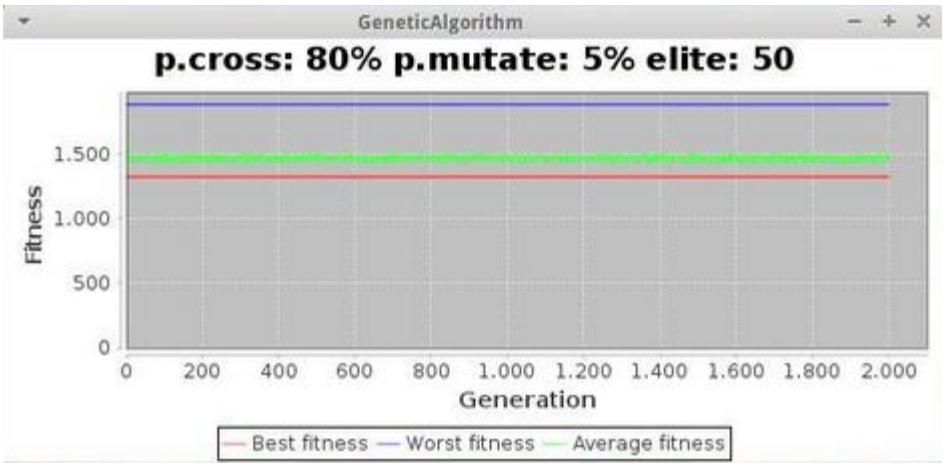


Gráfico I evoluçao do problema cinco linha

Anel de 50 pontos

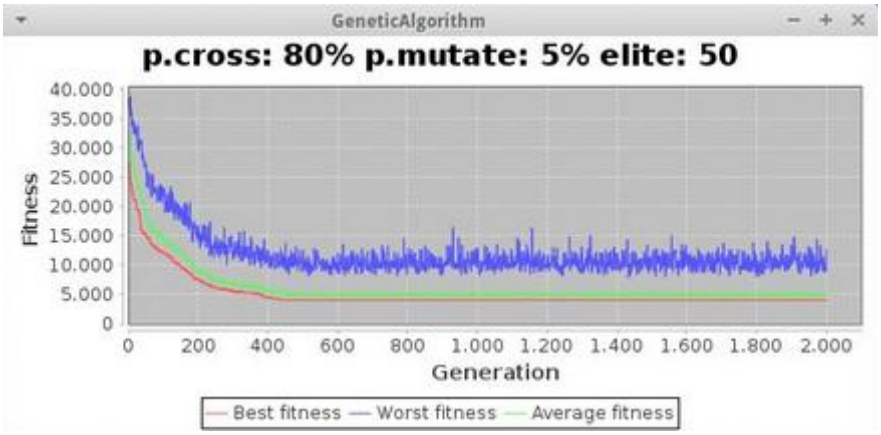


Gráfico II evoluçao do problema Anel de 50Pontos

30 Real

Durante a fase de desenvolvimento do trabalho, encontramos alguns valores de referência para a população, mutação, crossover, elite. Estes valores foram a referência para os testes que fizemos de seguida. Isto é, testámos sobre cada factor valores abaixo e valores acima da referencia que tinhamos. E combinamos todas as hipóteses possíveis para que os resultados fossem claros. Depois da tabela, encontram-se os gráficos de cada teste.

Para os testes usámos:

População: 200.
Gerações: 2000.
Operador: OX1

Teste nº	Elite	Crossover	Mutation	HillClimbing (S/N)	Best individual found:	Best individual fitness:
1	20	30%	1%	Sim	[4 16 3 17 29 28 11 6 1 9 8 18 24 27 26 20 22 0 25 7 23 10 5 19 14 2 21 13 12 15]	1.416.952
2	20	30%	1%	Não	[5 25 17 27 11 6 20 22 0 29 28 7 16 23 13 3 15 24 18 12 4 8 1 9 26 10 14 2 21 19]	1.482.757
3	20	30%	5%	Sim	[19 22 20 27 9 18 24 12 15 1 11 6 26 8 3 16 4 7 21 14 2 23 0 10 17 29 28 25 5 13]	1.493.678
4	20	30%	5%	Não	[3 18 24 12 4 19 25 23 13 17 27 11 6 26 22 14 2 0 29 28 7 9 8 1 20 10 5 21 15 16]	1.560.196
5	20	30%	25%	Sim	[5 22 0 25 20 27 9 18 24 15 12 1 17 10 29 21 2 7 28 26 11 6 8 4 16 3 19 14 23 13]	1.584.504
6	20	30%	25%	Não	[9 26 17 29 28 7 23 13 19 5 21 10 27 6 20 11 18 15 3 16 25 14 2 0 22 1 12 4 24 8]	1.567.619

7	20	60%	1%	Sim	[23 10 20 22 11 6 1 9 15 24 8 18 12 4 3 16 13 2 19 14 5 0 25 27 26 17 29 28 7 21]	1.463.023
8	20	60%	1%	Não	[3 4 27 9 8 1 17 29 28 0 23 14 21 19 10 22 7 26 11 6 20 2 5 25 13 24 18 12 15 16]	1.560.196
9	20	60%	5%	Sim	[23 2 25 26 11 6 27 9 18 12 15 24 8 1 20 22 14 19 5 0 10 29 28 17 4 3 16 7 21 13]	1.568.625
10	20	60%	5%	Não	[3 4 7 21 14 0 29 28 10 25 19 23 13 2 5 22 17 27 11 6 20 26 1 9 8 18 24 12 15 16]	1.382.646
11	20	60%	25%	Sim	[5 25 23 10 0 22 7 28 17 29 9 27 11 6 20 26 1 8 18 15 24 12 4 16 3 14 2 21 13 19]	1.417.191
12	20	60%	25%	Não	[5 21 13 23 19 14 2 7 28 17 10 25 16 24 15 12 8 1 26 29 0 22 20 11 6 27 9 18 3 4]	1.485.813
13	20	90%	1%	Sim	[0 22 7 28 27 11 6 20 26 29 17 10 25 15 12 24 18 8 1 9 4 3 16 14 2 5 21 13 19 23]	1.475.699
14	20	90%	1%	Não	[3 18 8 27 11 6 20 26 17 22 29 28 7 23 10 25 19 0 14 2 5 21 13 24 1 9 4 12 15 16]	1.382.646
15	20	90%	5%	Sim	[12 4 7 25 0 29 28 26 17 22 14 2 23 13 19 5 21 10 20 11 6 27 18 3 16 24 9 8 1 15]	1.402.231

16	20	90%	5%	Não	[23 19 14 2 0 29 16 24 8 1 9 11 20 27 6 26 18 3 15 12 4 7 28 22 17 10 25 5 21 13]	1.479.850
17	20	90%	25%	Sim	[21 10 20 22 17 27 11 6 26 29 28 7 9 18 15 24 8 1 12 4 16 3 19 14 2 5 0 25 23 13]	1.394.657
18	20	90%	25%	Não	[24 9 27 11 6 20 26 8 1 17 10 22 7 29 28 25 0 2 5 21 14 19 23 13 3 18 15 12 4 16]	1.488.576
19	50	30%	1%	Sim	[3 4 7 28 27 21 2 13 22 17 29 5 10 0 25 23 19 14 20 11 6 26 18 9 8 1 24 12 15 16]	1.560.196
20	50	30%	1%	Não	[17 27 26 11 6 20 22 25 19 21 10 5 14 2 23 13 24 4 16 7 0 29 3 15 12 18 9 8 1 28]	1.513.905
21	50	30%	5%	Sim	[23 14 2 5 0 10 17 27 9 8 1 22 25 19 21 13 3 18 24 15 12 4 7 28 11 6 20 26 29 16]	1.447.314
22	50	30%	5%	Não	[3 16 14 19 5 0 10 20 11 6 8 9 26 29 17 1 15 12 4 13 25 23 21 2 7 28 22 27 18 24]	1.620.659
23	50	30%	25%	Sim	[24 12 4 7 23 10 5 22 0 25 15 18 9 28 29 19 14 2 21 13 17 27 11 6 20 26 8 1 16 3]	1.402.231
24	50	30%	25%	Não	[11 6 20 22 5 19 14 2 25 17 27 9 8 1 3 18 16 12 24 15 4 7 23 13 21 10 0 29 28 26]	1.435.692

25	50	60%	1%	Sim	[23 14 21 10 25 18 24 12 15 16 3 9 8 27 26 17 29 28 22 20 11 6 1 4 7 5 19 0 2 13]	1.479.850
26	50	60%	1%	Não	[25 19 21 10 22 17 1 26 11 6 20 27 9 8 24 18 12 15 3 4 16 23 13 5 14 2 0 29 28 7]	1.495.234
27	50	60%	5%	Sim	[17 29 20 11 6 1 24 3 15 12 4 13 19 5 21 14 2 23 10 0 22 7 28 25 27 26 18 9 8 16]	1.519.806
28	50	60%	5%	Não	[23 19 10 20 26 29 28 17 22 0 25 5 21 14 2 7 8 27 11 6 1 9 18 24 16 3 15 12 4 13]	1.391.075
29	50	60%	25%	Sim	[23 10 17 29 20 27 9 18 24 12 15 4 7 28 11 6 26 8 1 3 16 22 25 0 19 14 2 5 21 13]	1.391.075
30	50	60%	25%	Não	[3 4 24 27 9 1 11 6 20 10 5 0 22 25 19 21 13 23 14 2 7 28 26 17 29 8 18 12 15 16]	1.382.646
31	50	90%	1%	Sim	[23 14 2 5 21 10 0 25 19 20 22 7 28 17 29 9 8 1 11 6 27 26 18 3 16 24 15 12 4 13]	1.391.075
32	50	90%	1%	Não	[17 22 29 28 1 9 26 20 11 6 27 8 18 24 3 4 12 15 16 23 14 2 21 13 19 5 0 10 25 7]	1.393.016
33	50	90%	5%	Sim	[17 29 28 11 6 26 18 24 12 15 3 4 13 23 19 14 2 21 10 25 5 0 22 20 27 9 8 1 16 7]	1.393.016

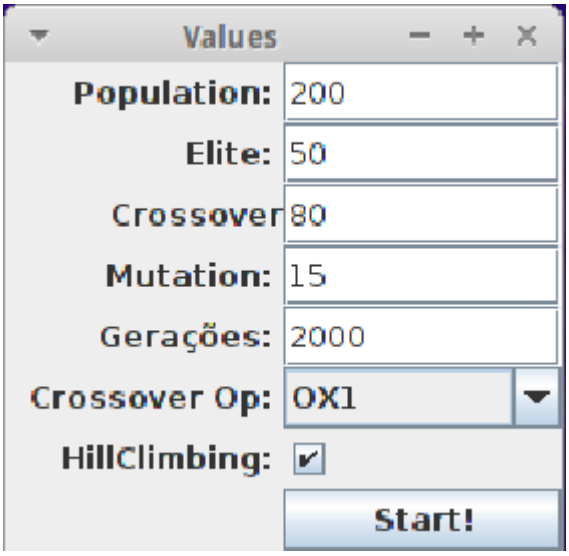
34	50	90%	5%	Não	[16 3 9 8 27 26 11 6 20 22 14 2 23 13 19 5 21 10 0 25 15 4 12 18 24 1 17 29 28 7]	1.408.592
35	50	90%	25%	Sim	[3 4 12 1 9 8 27 26 11 6 20 10 22 17 29 28 7 19 5 0 25 23 14 2 21 13 24 18 15 16]	1.382.646
36	50	90%	25%	Não	[12 4 16 17 10 0 25 21 13 19 23 14 2 5 22 20 11 6 26 27 9 8 1 29 28 7 3 18 24 15]	1.402.231
37	100	30%	1%	Sim	[0 25 23 10 27 18 12 4 21 2 22 7 20 26 11 6 19 14 5 13 29 28 8 1 15 16 3 24 9 17]	1.759.478
38	100	30%	1%	Não	[21 20 6 26 22 14 5 25 29 28 7 15 16 9 8 1 24 3 4 12 18 19 0 10 17 27 11 23 13 2]	1.796.615
39	100	30%	5%	Sim	[3 15 24 12 4 13 23 19 14 2 22 0 25 5 21 10 20 11 6 27 9 8 1 17 29 7 28 26 18 16]	1.394.814
40	100	30%	5%	Não	[7 19 29 28 26 18 9 8 1 24 3 4 12 15 16 17 27 11 6 20 22 0 14 2 23 10 25 5 21 13]	1.412.723
41	100	30%	25%	Sim	[23 19 14 2 0 29 28 7 8 1 12 4 15 16 3 17 10 13 18 24 9 27 11 6 26 20 22 25 5 21]	1.469.567
42	100	30%	25%	Não	[23 19 14 7 28 17 29 3 4 21 2 10 25 5 0 22 11 6 26 20 27 8 18 16 1 9 15 12 24 13]	1.568.625

43	100	60%	1%	Sim	[3 15 8 27 9 18 1 16 24 12 4 13 22 11 6 20 26 17 29 28 7 21 19 10 0 14 2 5 25 23]	1.507.120
44	100	60%	1%	Não	[17 29 10 21 2 5 0 22 7 23 19 14 25 18 16 12 24 15 3 4 13 20 11 6 26 27 8 1 9 28]	1.602.680
45	100	60%	5%	Sim	[23 0 10 5 14 2 20 27 26 8 1 17 22 29 28 11 6 9 24 4 3 15 12 18 16 25 7 19 21 13]	1.479.850
46	100	60%	5%	Não	[23 13 15 8 1 16 3 0 22 25 29 4 12 24 18 9 27 11 6 20 26 28 17 10 5 19 14 2 7 21]	1.551.798
47	100	60%	25%	Sim	[17 27 8 18 12 15 24 3 4 7 23 0 10 5 14 19 13 2 21 16 9 26 29 28 25 1 11 6 20 22]	1.527.522
48	100	60%	25%	Não	[17 27 18 9 6 20 11 22 5 10 0 25 15 16 3 12 4 24 8 1 26 23 19 14 2 21 13 7 28 29]	1.570.137
49	100	90%	1%	Sim	[12 24 18 3 4 16 21 14 2 5 13 25 23 19 10 0 22 17 29 7 28 26 20 11 6 27 8 1 9 15]	1.407.589
50	100	90%	1%	Não	[21 19 10 0 14 29 28 17 22 20 11 6 26 8 1 9 27 18 24 16 3 15 12 4 7 25 23 13 2 5]	1.452.013
51	100	90%	5%	Sim	[5 21 14 2 23 13 19 29 7 28 17 22 0 10 20 11 6 26 27 18 3 8 1 9 24 15 12 4 16 25]	1.420.649

52	100	90%	5%	Não	[17 29 28 26 11 6 20 2 13 23 14 25 19 5 21 10 0 22 1 9 8 27 18 3 4 12 15 24 16 7]	1.481.791
53	100	90%	25%	Sim	[17 22 10 0 14 23 13 2 5 21 3 4 24 12 15 27 8 1 9 18 16 25 19 20 11 6 26 29 28 7]	1.481.791
54	100	90%	25%	Não	[0 22 26 11 6 20 27 29 19 14 2 5 21 13 23 18 15 24 16 12 4 3 1 9 8 7 28 17 10 25]	1.579.975

Interface

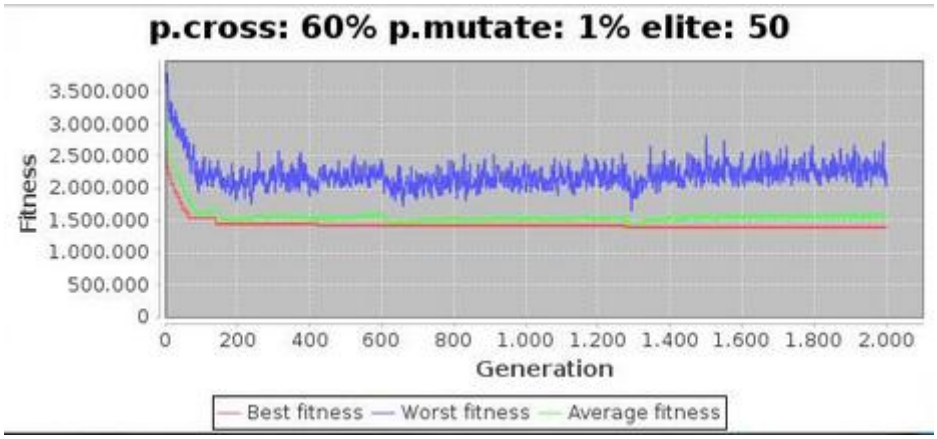
Interface gráfica com os valores default.



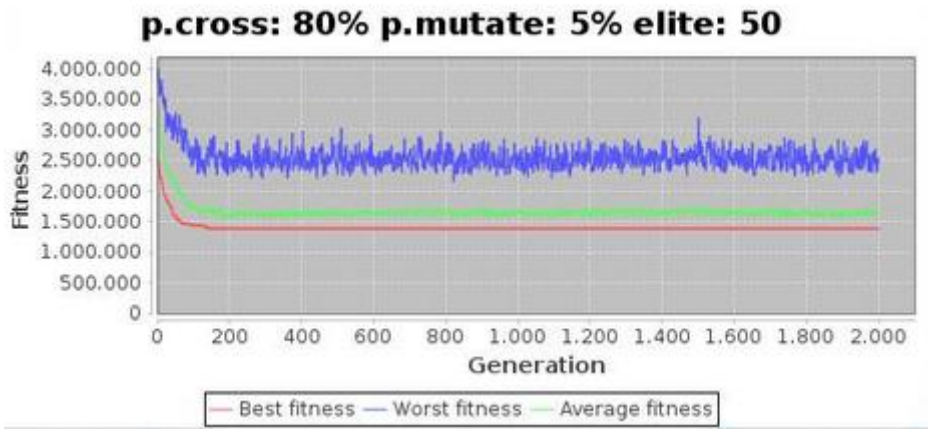
1- Interface para introdução de valores. População, Elite Crossover, Mutation, Gerações, OX1 e OX2, HillClimbing(on/off)

Gráficos 30 Real

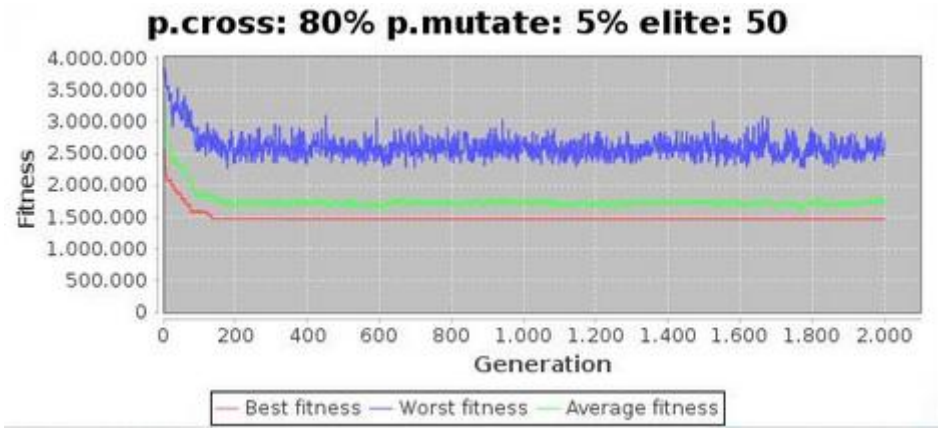

Values
Population: 200
Elite: 50
Crossover 60
Mutation: 1
Gerações: 2000
Crossover Op: OX1
HillClimbing:
Start!


p.cross: 60% p.mutate: 1% elite: 50
Fitness
3.500.000
3.000.000
2.500.000
2.000.000
1.500.000
1.000.000
500.000
0
0 200 400 600 800 1.000 1.200 1.400 1.600 1.800 2.000
Generation
Best fitness
Worst fitness
Average fitness


Values
Population: 300
Elite: 50
Crossover 80
Mutation: 5
Gerações: 2000
Crossover Op: OX1
HillClimbing:
Start!


p.cross: 80% p.mutate: 5% elite: 50
Fitness
4.000.000
3.500.000
3.000.000
2.500.000
2.000.000
1.500.000
1.000.000
500.000
0
0 200 400 600 800 1.000 1.200 1.400 1.600 1.800 2.000
Generation
Best fitness
Worst fitness
Average fitness


Values
Population: 300
Elite: 50
Crossover: 80
Mutation: 5
Gerações: 2000
Crossover Op: OX1
HillClimbing:
Start!


p.cross: 80% p.mutate: 5% elite: 50
4.000.000
3.500.000
3.000.000
2.500.000
2.000.000
1.500.000
1.000.000
500.000
0
Fitness
0 200 400 600 800 1.000 1.200 1.400 1.600 1.800 2.000
Generation
Best fitness
Worst fitness
Average fitness

Conclusão/Discussão de resultados

Através dos testes que fizemos podemos supor de que forma os parâmetros influenciam os resultados. Entre os operadores OX1 e OX2, o OX1 foi claramente o que nos deu melhores resultados. O Hill Climbing influencia bastante o desenho do gráfico, tornando a curva mais acentuada do que quando não está activo. Ou seja, activando o Hill Climbing chegamos aos melhores valores, mais rapidamente. No caso do crossover, percebemos que se aumentarmos a probablilidade de ocorrer, obtemos resultados melhores, mais rapidamente. A probabilidade de mutação, é também importante, pois evita que se estagne, antes de chegar a melhores resultados, isto em casos em que a população é mais homogénea. A elite, com valores entre 20 a 25% da população, também melhora os resultados obtidos. Percebemos que aumentando as gerações não melhora significativamente os resultados pois a população está a ficar homogénea por volta da geração nº 200, no entanto, aumentando a população obtém-se melhorias nos resultados. No problema do Cinco Linha, não há evolução do gráfico, estagnando desde o início, pois ao gerar os indivíduos da 1ª geração encontram-se logo os indivíduos com o melhor fitness, a não ser que a amostra da população seja demasiado pequena.

Anexo

Neste anexo está o código que fonte do trabalho.

CircuitTest

```
package circuit;

import java.io.*;

import static java.lang.Math.*;

public class CircuitTest {

	public static final int DEFAULT_POP = 200;

	private int pop_size;
	/**
	 * @param args
	 * @throws IOException, ClassNotFoundException
	 */
	//public static void main(String[] args) throws IOException, ClassNotFoundException {
	public CircuitTest(int pop_size,float pcrossover, float pmutate, int gen_cap, int elite,
int crossoverOP, boolean hillc) throws IOException, ClassNotFoundException{
		if (pop_size > 0)
			this.pop_size = pop_size;
		else
			this.pop_size = DEFAULT_POP;

		String datastr = "";
		BufferedReader reader = new BufferedReader( new
FileReader("/home/doping/Downloads/CinquentaAnel.txt"));

		String line = reader.readLine();
		while( line != null) {
			datastr += line + "\n";
			line = reader.readLine();
		}

		reader.close();


		ObservationData r = new ObservationData(datastr);

		int maxdist=0;
		int mindist=Integer.MAX_VALUE;

		for(int i=0; i < r.getSize(); i++ )
			for(int j=0; j < r.getSize(); j++ ) {
				maxdist = max(maxdist,r.getCost(i, j));
				if( i != j ) mindist = min(mindist,r.getCost(i, j));
			}

		System.out.println(mindist+" "+maxdist);

		/*
		Random gen = new Random();
		int size = 20;

		ObservationData.ObservationInterval interval = r.new
ObservationInterval(0,ObservationData.SOL-1);
		SortedSet<ObservationData.ObservationInterval> is = new
TreeSet<ObservationData.ObservationInterval>();
		is.add(interval);

		ObservationData.ObservationSpot[] spot = new
ObservationData.ObservationSpot[size];
		int[][] distances = new int[size][size];
		for( int s =0 ; s < size ; s++) {
			spot[s] = r.new
ObservationSpot("P"+s,gen.nextInt(ObservationData.SOL/2),is);
```

```
                }
                for( int i=0; i < size; i++)
                        for( int j=0; j < size; j++)
                                if( i != j )
                                        distances[i][j] = gen.nextInt(6*ObservationData.SOL);

                r = new ObservationData(size,spot,distances);
                */
                /*RoverCircuit p1 = new RoverCircuit(r);
                RoverCircuit p2 = new RoverCircuit(r);

                System.out.println(p1);
                System.out.println(p1.fitness());
                System.out.println(p2);
                System.out.println(p2.fitness());*/

                /*Individual[] offspring = p1.crossover(p2);
                System.out.println(offspring[0]);
                System.out.println(offspring[0].fitness());
                System.out.println(offspring[1]);
                System.out.println(offspring[1].fitness());*/

                Population p = new Population();
                for(int i=0; i < this.pop_size; i++) {
                        Individual ind = new RoverCircuit(r);
                        //System.out.println(ind);
                        //System.out.println(ind.fitness());
                        p.addIndividual(ind);
                }
                System.out.println(p.getBestFitness() + " "+ p.getAvgFitness() + " "
+p.getWorstFitness());
                System.out.println(p.getBestIndividual());

                //System.out.println(p.selectIndividual());
                //System.out.println(p.selectIndividual());

                GeneticAlgorithm ga = new GeneticAlgorithm(p,pcrossover,pmutate,gen_cap,elite,
crossoverOP,hillc);

                System.out.println("--------------------");
                Individual bestind = ga.search();
                System.out.println("Best individual found: "+ bestind);
                System.out.println("Best individual fitness: "+ bestind.fitness());


        }

}
```

Population

```
package circuit;

import java.util.ArrayList;
import java.util.Random;
import java.util.Collections;
/**
 *      Classe usada para a representaï¿½ï¿½o de uma populaï¿½ï¿½o.
 */
public class Population {

        private final static int CAP = 100;
        private static Random gen = new Random();
        private double sumOfFitness;
        private int size;
        private boolean currupt;
        private Individual bestInd, worstInd;
        private double bestFit, worstFit;
        private ArrayList<Individual> pop;
        private ArrayList<Double> acum;

        /**
         *      Construtor relativo ï¿½ classe Population
```

```
         */
        public Population(){
                this.size = 0;
                this.pop = new ArrayList <Individual>(CAP);
                this.acum = new ArrayList <Double>(CAP);
                this.sumOfFitness = 0.0;
                this.currupt = true;
                this.bestInd = null;
                this.worstInd = null;
                this.bestFit = Double.POSITIVE_INFINITY;
                this.worstFit = Double.NEGATIVE_INFINITY;
                gen.setSeed(System.nanoTime());
        }

        /**
         *      Construtor onde se especifica a popolaï¿½ï¿½o
         * @param p um array de indivï¿½duos
         */
        public Population(Individual[] pop){
                this();
                for(Individual i:pop)
                        addIndividual(i);
        }

        /**
         * Selecciona e devolve um indivï¿½duo da populaï¿½ï¿½o, tendo em conta a sua fitness
         * @return um array de indivï¿½duos
         */
        public Individual selectIndividual() {
                //estÃ¡ a implementar o mÃ©todo da roleta
                // Verifica se necessita de calcular os valores de probabilidade de selecï¿½ï¿½o
de cada indivï¿½duo
                if( currupt ) {
                        acum.clear();
                        double total=0.0;
                        for(int i=0; i < pop.size(); i++) {
                                total += 1/pop.get(i).fitness();
                                acum.add(total/sumOfFitness);
                        }
                        currupt = false;
                }

                double r = gen.nextDouble();
                int pos = Collections.binarySearch(acum, r);

                if( pos >= 0)
                        return pop.get(pos%size);
                else
                        return pop.get(-(pos+1)%size);

        }
        /**
         * Adiciona um indivï¿½duo ï¿½ populaï¿½ï¿½o
         * @param ind, um indivï¿½duo
         */
        public void addIndividual(Individual ind) {
                size++;
                pop.add(ind);
                double f = ind.fitness();
                sumOfFitness += 1/f;
                if( f > worstFit ) {
                        worstFit = f;
                        worstInd = ind;
                }
                if( f < bestFit ) {
                        bestFit = f;
                        bestInd = ind;
                }
        }

        public Population getElite(int n){
                Population newpop = new Population();
                Collections.sort(pop);
                if (n>size)
```

```
                n=size;
        for (int i=0;i<n;i++)
                newpop.addIndividual((Individual)pop.get(i).clone());
        return newpop;
    }

    public double getAvgFitness() {
        double avg = 0.0;
        for (Individual i:pop)
                avg += i.fitness();

        return avg/size;
    }

    public double getWorstFitness() {
        return worstFit;
    }

    public double getBestFitness() {
        return bestFit;
    }

    public Individual getBestIndividual() {
        return bestInd;
    }

    public int getSize(){
        return size;
    }

    public void hillClimbing(int n){
        Collections.sort(pop);
        boolean changed;
        Individual aux;
        for (int i = 0; i<n;i++){
                changed = true;
                while (changed){
                        changed = false;
                        aux = ((Individual)pop.get(i).clone());
                        aux.mutate();
                        if (aux.fitness()<=pop.get(i).fitness()){
                                currupt = true;
                                pop.set(i, aux);
                                if (aux.fitness()<bestFit){
                                        changed = true;
                                        bestInd = aux;
                                        bestFit = aux.fitness();
                                }
                        }
                }
        }




        /*Individual []children;
        while(changed){
                children = bestInd.crossover(this.selectIndividual());
                changed = false;
                for (int i=0;i<children.length;i++)
                        if (children[i].fitness()<bestInd.fitness()){
                                bestInd = children[i];
                                pop.set(0, bestInd);
                                changed = true;
                                currupt = true;
                        }
        }*/
    }

}
```

RoverCircuit

```
package circuit;

import java.util.*;

/**
 *      Classe que instancia a classe abstracta Individual
 */
public class RoverCircuit extends Individual {

        private static Random gen = new Random();
        private static Individual []children = new Individual[2];
        private int size;
        private int []circuit;
        private ObservationData data;
        private Double fit = 0.0;


        //2 construtores
        public RoverCircuit(ObservationData data){
                this.data = data;
                size = data.getSize();
                circuit = new int[size];
                List<Integer> c = new ArrayList<Integer>();
                for (int i = 0; i<data.getSize(); i++)
                        c.add(i);

                gen.setSeed(System.nanoTime());
                Collections.shuffle(c,gen);
                for(int i = 0;i<c.size();i++){
                        circuit[i]=c.get(i);

                }
                //gerar uma lista de inteiros cegamente (0..n-1)
                //aplicar shuffle
                //copiar para circuit
        }

        public RoverCircuit(ObservationData data, int[] circuit){
                this.data = data;
                this.circuit = new int[circuit.length];
                for (int i:circuit)
                        this.circuit[i] = circuit[i];
                size = circuit.length;
        }

        @Override
        public double fitness() {
                int time = data.getSpot(circuit[0]).firstTime();

                for(int i = 0; i<size;i++){
                        time += data.getSpot(circuit[i]).durationObservation(time);
                        if (i<size-1)
                                time += data.getCost(circuit[i], circuit[i+1]);
                        else
                                time += data.getCost(circuit[i], circuit[0]);
                }
                fit = (double)time;
                return fit;
        }

        @Override
        public Individual[] crossover(Individual other, GeneticAlgorithm.crossoverOP op) {
                switch (op){
                case OX1: return OX1(other);
                case OX2: return OX2(other);
                }
                return null;
        }
```

```
	private Individual[] OX2(Individual other){
		int[] ca = new int[size];
		int[] cb = new int[size];
		boolean[] check1 = new boolean[size];
		boolean[] check2 = new boolean[size];

		int cut1,cut2;
		int r1 = gen.nextInt(size-1);
		int r2 = gen.nextInt(size-2);
		if (r2 >= r1){
			cut1=r1+1;
			cut2=r2+2;
		}
		else{
			cut1=r2+1;
			cut2=r1+2;
		}

		for (int i=cut1;i<cut2;i++){
			ca[i] = this.getCircuit()[i];
			check1[ca[i]]=true;
			cb[i] = ((RoverCircuit)other).getCircuit()[i];
			check2[cb[i]]=true;
		}
		int n1 = 0;
		int n2 = 0;
		for(int i=0;i<size;i++){
			if (!check1[((RoverCircuit)other).getCircuit()[i]])
				ca[n1++]=((RoverCircuit)other).getCircuit()[i];
			if (!check2[this.getCircuit()[i]])
				cb[n2++]=this.getCircuit()[i];
			if (n1 == cut1)
				n1 = cut2;
			if (n2 == cut1)
				n2 = cut2;
		}

		children[0] = new RoverCircuit(data,ca);
		children[1] = new RoverCircuit(data,cb);

		return children;
	}

	private Individual[] OX1(Individual other){
		int[] ca = new int[size];
		int[] cb = new int[size];
		boolean[] check1 = new boolean[size];
		boolean[] check2 = new boolean[size];

		int cut1,cut2;
		int r1 = gen.nextInt(size-1);
		int r2 = gen.nextInt(size-2);
		if (r2 >= r1){
			cut1=r1+1;
			cut2=r2+2;
		}
		else{
			cut1=r2+1;
			cut2=r1+2;
		}

		for (int i=cut1;i<cut2;i++){
			ca[i] = this.getCircuit()[i];
			check1[ca[i]]=true;
			cb[i] = ((RoverCircuit)other).getCircuit()[i];
			check2[cb[i]]=true;
		}
		int n1 = cut2%size;
		int n2 = cut2%size;
		for(int i=cut2;i<size;i++){
			if (!check1[((RoverCircuit)other).getCircuit()[i]])
				ca[n1++]=((RoverCircuit)other).getCircuit()[i];
			if (!check2[this.getCircuit()[i]])
				cb[n2++]=this.getCircuit()[i];
```

```
                        if (n1 == size)
                                n1 = 0;
                        if (n2 == size)
                                n2 = 0;
                }

                for(int i=0;i<cut2;i++){
                        if (!check1[((RoverCircuit)other).getCircuit()[i]])
                                ca[n1++]=((RoverCircuit)other).getCircuit()[i];
                        if (!check2[this.getCircuit()[i]])
                                cb[n2++]=this.getCircuit()[i];
                        if (n1 == size)
                                n1 = 0;
                        if (n2 == size)
                                n2 = 0;
                }

                children[0] = new RoverCircuit(data,ca);
                children[1] = new RoverCircuit(data,cb);

                return children;
        }

        @Override
        public void mutate() {
                // operador de troca
                int n1 = gen.nextInt(size);
                int n2 = gen.nextInt(size);
                if (n1 == n2)
                        n1++;
                n1=n1%size;
                int aux = circuit[n1];
                circuit[n1]=circuit[n2];
                circuit[n2]=aux;
        }

        @Override
        public Object clone() {
                return new RoverCircuit(data, circuit);
        }

        public String toString(){
                String s = "[";
                for (int i = 0;i<circuit.length-1;i++)
                        s += circuit[i] + " ";
                s+=circuit[circuit.length-1]+"]";
                return s;
        }

        public int[] getCircuit(){
                return circuit;
        }

        @Override
        public int compareTo(Individual other) {
                double i1 = this.fitness();
                double i2 = other.fitness();
                if( i1 >  i2)
                        return 1;
                else if ( i1 < i2 )
                        return -1;
                else
                        return 0;
        }

}
```

GeneticAlgorithm

```
package circuit;

import java.util.Random;
```

```
import javax.swing.JFrame;

import org.jfree.chart.*;
import org.jfree.data.xy.*;
/**
 * Classe que "implementa" o algoritmo genï¿½tico
 */
public class GeneticAlgorithm {

	public final static int GEN_CAP = 2000;
	public static final float DEFAULT_P_CROSSOVER = 0.6f;
	public static final float DEFAULT_P_MUTATE = 0.01f;
	public static final int ELITE = 50;

	private static Random gen = new Random();
	private float pcrossover;
	private float pmutate;
	private int elite;
	private int gen_cap;
	private Population pop;
	private crossoverOP crossoverop;
	private boolean hillClimbing;

	public enum crossoverOP{
		OX1,OX2
	};

	/**
	 * Construtor
	 * @param pop uma populaï¿½ï¿½o
	 */
	GeneticAlgorithm(Population pop) {
		gen.setSeed(System.nanoTime());
		pcrossover = DEFAULT_P_CROSSOVER;
		pmutate = DEFAULT_P_MUTATE;
		elite = ELITE;
		gen_cap = GEN_CAP;
		this.pop = pop;
	}
	/**
	 * Construtor
	 * @param pop uma populaï¿½ï¿½o
	 * @param pcrossover a probabilidade de crossover
	 * @param pmutate a probabilidade de mutaï¿½ï¿½o
	 */
	GeneticAlgorithm(Population pop, float pcrossover, float pmutate, int gen_cap, int
elite, int crossoveroperator, boolean hillc) {
		gen.setSeed(System.nanoTime());
		this.pop = pop;
		this.pcrossover = pcrossover;
		this.pmutate = pmutate;
		this.elite = elite;
		this.gen_cap = gen_cap;
		if (crossoveroperator == 0)
			crossoverop = crossoverOP.OX1;
		else
			crossoverop = crossoverOP.OX2;
		this.hillClimbing = hillc;
	}

	/**
	 *	Mï¿½todo que pesquisa e devolve o melhor indivï¿½duo encontrado
	 * @return pop.getBestIndividual(), o melhor indivï¿½duo
	 */
	public Individual search() {
		Population newpop;
		Individual x,y;
		Individual[] children = new Individual[2];
		JFreeChart chart;
		XYSeriesCollection data = new XYSeriesCollection();
		XYSeries best = new XYSeries("Best fitness");
		XYSeries worst = new XYSeries("Worst fitness");
		XYSeries average = new XYSeries("Average fitness");
```

```
                        for(int i = 0;i<gen_cap;i++){
                                best.add(i,pop.getBestFitness());
                                worst.add(i,pop.getWorstFitness());
                                average.add(i,pop.getAvgFitness());
                                newpop = pop.getElite(elite);
                                while(newpop.getSize()<pop.getSize()){
                                        x=pop.selectIndividual();
                                        y=pop.selectIndividual();
                                        if (gen.nextFloat()<=pcrossover)
                                                children = x.crossover(y,crossoverop);
                                        else{
                                                children[0] = x;
                                                children[1] = y;
                                        }
                                        if(gen.nextFloat()<=pmutate)
                                                children[0].mutate();
                                        if(gen.nextFloat()<=pmutate)
                                                children[1].mutate();
                                        newpop.addIndividual(children[0]);
                                        newpop.addIndividual(children[1]);
                                }
                                pop=newpop;
                                if (hillClimbing)
                                        pop.hillClimbing(elite);
                        }
                        best.add(gen_cap,pop.getBestFitness());
                        worst.add(gen_cap,pop.getWorstFitness());
                        average.add(gen_cap,pop.getAvgFitness());
                        data.addSeries(best);
                        data.addSeries(worst);
                        data.addSeries(average);
                        chart = ChartFactory.createXYLineChart("p.cross:
"+Math.round(pcrossover*100)+"% p.mutate: "+Math.round(pmutate*100)+"% elite: "+elite,
"Generation", "Fitness", data);
                        ChartPanel chartPanel = new ChartPanel(chart);
                chartPanel.setPreferredSize(new java.awt.Dimension(600, 270));
                chartPanel.setVisible(true);
                JFrame frame = new JFrame("GeneticAlgorithm");
                frame.add(chartPanel);
                frame.pack();
                        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                frame.setVisible(true);
                return pop.getBestIndividual();
        }
}
```

Main

```
package circuit;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.swing.*;

public class Main {
        public static void main(String[] args) throws ClassNotFoundException, IOException {
                setWindow();
            }

        private static void setWindow() {

            final String[] labels = {"Population: ", "Elite: ", "Crossover", "Mutation: ",
"GeraÃ§Ãµes: ","Crossover Op: ", "HillClimbing: "};
            final JTextField []textField = new JTextField[labels.length-2];
            int labelsLength = labels.length;

            //Create and populate the panel.
            JPanel p = new JPanel(new SpringLayout());
            for (int i = 0; i < labelsLength-2; i++) {
                JLabel l = new JLabel(labels[i], JLabel.TRAILING);
```

```
                p.add(l);
                textField[i] = new JTextField(10);
                l.setLabelFor(textField[i]);
                p.add(textField[i]);
            }
            JLabel l = new JLabel(labels[labels.length-2], JLabel.TRAILING);
            p.add(l);
            final JComboBox c = new JComboBox();
            c.addItem("OX1");
            c.addItem("OX2");
            p.add(c);
            l = new JLabel(labels[labels.length-1], JLabel.TRAILING);
            p.add(l);
            final JCheckBox cc = new JCheckBox();
            cc.setSelected(true);
            p.add(cc);
            JButton button = new JButton("Start!");
            p.add(new JLabel());
            p.add(button);
            textField[0].setText(""+CircuitTest.DEFAULT_POP);
            textField[1].setText(""+GeneticAlgorithm.ELITE);
            textField[2].setText(""+Math.round(100*GeneticAlgorithm.DEFAULT_P_CROSSOVER));
            textField[3].setText(""+Math.round(100*GeneticAlgorithm.DEFAULT_P_MUTATE));
            textField[4].setText(""+GeneticAlgorithm.GEN_CAP);

            //Lay out the panel.
            p.setLayout(new GridLayout(8, 2));

            button.addActionListener(new ActionListener() {

                public void actionPerformed(ActionEvent e)
                {
                try {
                        CircuitTest circuit = new
CircuitTest(Integer.parseInt(textField[0].getText()),

        Float.parseFloat(textField[2].getText())/100,
Float.parseFloat(textField[3].getText())/100,

        Integer.parseInt(textField[4].getText()), Integer.parseInt(textField[1].getText()),
                                                        c.getSelectedIndex(), cc.isSelected());
                } catch (ClassNotFoundException | IOException e1) {
                                    e1.printStackTrace();
                                }
                    //Execute when button is pressed
                    //System.out.println("Test");
                }
            });

            JFrame frame = new JFrame("Values");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            p.setOpaque(true);
            frame.setContentPane(p);

            frame.setLocation(600, 100);
            frame.pack();
            frame.setVisible(true);
        }

}
```