UECE - CCT - Curso de Ciência da Computação

Disciplina: Teoria dos Grafos - 2016.2

Inserção Aleatória

Francisco Gleidson da Silva Ferreira: gleidson.fgs@gmail.com

Juvenal da Costa Lavres da Conceição: jconceicao09@gmail.com

Dhayvison Braga Mendonça: dhayvisonbraga@gmail.com

Introdução

A heurística construtiva de critério aleatório tem por objetivo a resolução do problema de forma simplificada, diminuindo o número de comparações entre os vértices não adicionados ao conjunto solução. Isto ocorre porque diferente de outros métodos, os vértices não adicionados não precisam ser avaliados por um critério, seja este melhor, mais próximo ou mais distante. Gerando, com inexatidão, uma solução aproximada para o problema.

1 Descrição do Problema

Dadas n cidades, o problema consiste em definir o menor ciclo que passa uma vez por cada uma delas, sendo que há um caminho direto entre cada duas cidades. A dificuldade deste problema consiste em encontrar a solução em tempo hábil. O método clássico consiste em definir todos as rotas, calcular os custos e assim definir qual o menor ciclo existente. Apesar de ser um método determinístico, a quantidade de operações necessárias para realizá-lo cresce na ordem de (n-1)!, que faz com que o tempo de de operação exploda mesmo para distâncias pequenas de n. Por exemplo, para n = 20, serão necessárias 121.645.100.408.852.000 operações.

1.1 Algoritmo

O algoritmo segue os seguintes passos:

- Gerar um ciclo com três vértices aleatórios e calcular seu custo inicial.
- Adicionar vértices aleatoriamente ao ciclo considerando a melhor posição de inserção até que todos os vértices tenham sido inseridos.

Segue descrição e implementação em Java:

A classe Grafo foi definida com as seguintes variáveis necessárias para resolução do problema:

```
double[][] D; // Matriz de Distancia
ArrayList<Integer> vnv; // Lista dos vertices nao visitados
int[] ciclo, cicloInicial; // vetor de ciclo
int tamanhoCiclo; // tamanho atual do ciclo
double custoCiclo; // custo atual do ciclo
int TAMANHO; // constante que define o número de vertices
private BufferedReader read; // Leitura e escrita em arquivos
```

O método construtor da classe Grafo recebe como parâmetro (n) a quantidade de vértices.

```
/**
 * Inicializa o array de vertices nao visitados (vnv) e demais variaveis.
 * @param n diz respeito ao tamanho do ciclo e serve para montagem da
 * matriz de distancia (D)
 */
public Grafo(int n) {
	vnv = new ArrayList<>();
	ciclo = new int[n];
	cicloInicial = new int[3];
	D = new double[n][n]; // matriz de distancia
	TAMANHO = n;

	for (int i = 1; i <= TAMANHO; i++) {
		vnv.add(i); // popular os vertices nao visitados
	}

	custoCiclo = 0.0;
	tamanhoCiclo = 0;
}
```

O primeiro passo para a solução é gerar a matriz de distâncias/custo entre cada um dos pontos. Para isso implementamos três metodologias, considerando o tipo do arquivo, devidamente formatado. A criação do arquivo .csv foi utilizada apenas para checagem dos dados.

```
/**
 * Gera a matriz a partir do arquivo tipo 1
 *
 * @param filename
 * @throws IOException
 */

public void gerarMatrizTipo1(String filename) throws IOException {

	String[] aux = new String[TAMANHO];
/**
 * Processo de leitura do arquivo, e sua inserção na matriz de distâncias
 */
	try {
		read = new BufferedReader(new FileReader(filename));

		String line = read.readLine();
		line = read.readLine();
		System.out.println(D.length);
		double d;

		int i = 0;
		do {
			aux = line.split(" ");

			for (int j = i + 1; j < TAMANHO;) {
				for (int j2 = 0; j2 < aux.length; j2++) {
					d = Double.parseDouble(aux[j2]);
					// simetrica
					D[i][j] = d;
					D[j][i] = d;
					j++;
				}

				if ((j - i) > aux.length) {
					break;
```

```
                        }

                    }

                    line = read.readLine();
                    i++;

                } while (line != null);

            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }

            /**
             * Escreve no arquivo matriz_distancia_nome_do_arquivo_.csv
             */
            try {
                CSVWriter writer = new CSVWriter(new FileWriter("matriz_dis-
tancia_" + filename + ".csv"));

                String[] entrada = new String[TAMANHO];
                for (int i = 0; i < TAMANHO; i++) {

                    for (int j = 0; j < TAMANHO; j++) {
                        entrada[j] = "";
                        entrada[j] += D[i][j];
                    }
                    writer.writeNext(entrada);
                }

                writer.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }

        /**
         * Gera a matriz a partir do arquivo tipo 2
         *
         * @param filename
         * @throws IOException
         */
        public void gerarMatrizTipo2(String filename) throws IOException {
            /**
             * A classe ponto contém os atributos X e Y, do tipo double.
             */
            Ponto[] pontos = new Ponto[TAMANHO];
        /**
         * Processo de leitura do arquivo, e sua inserção na matriz de distâncias
         */
            try {
                read = new BufferedReader(new FileReader(filename));

                String line = read.readLine();
                line = read.readLine();

                int i = 0;

                do {
                    String[] s = line.split(" ");
                    pontos[i] = new Ponto(Double.parseDouble(s[0]), Double.-
parseDouble(s[1]));
                    line = read.readLine();
```

```
                        i++;

                } while (line != null);

            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }

            double d, xi, xj, yi, yj;

            for (int i = 0; i < TAMANHO; i++) {
                xi = pontos[i].getX();
                yi = pontos[i].getY();
                for (int j = 0; j < TAMANHO; j++) {
                    xj = pontos[j].getX();
                    yj = pontos[j].getY();

                    d = ((xi - xj) * (xi - xj)) + ((yi - yj) * (yi - yj));
                    d = Math.sqrt(d);
                    D[i][j] = d;
                    D[j][i] = d;
                }
            }

            /**
             * Escreve no arquivo matriz_distancia_nome_do_arquivo_.csv
             */
            try {
                CSVWriter writer = new CSVWriter(new FileWriter("matriz_dis-
tancia" + filename + ".csv"));

                String[] entrada = new String[TAMANHO];
                for (int i = 0; i < TAMANHO; i++) {

                    for (int j = 0; j < TAMANHO; j++) {
                        entrada[j] = "";
                        entrada[j] += D[i][j];
                    }
                    writer.writeNext(entrada);
                }

                writer.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        /**
         * Gera a matriz a partir do arquivo tipo 3, nao simetrica
         */
        public void gerarMatrizTipo3(String filename) throws IOException {

            String[] aux = new String[TAMANHO];

        /**
         * Processo de leitura do arquivo, e sua inserção na matriz de distâncias
         */            try {
                read = new BufferedReader(new FileReader(filename));

                String line = read.readLine();
                line = read.readLine();
                System.out.println(D.length);
                double d;
```

```
                int i = 0;
                do {

                        aux = line.split(" ");

                        for (int j = 0; j < TAMANHO; j++) {
                                d = Double.parseDouble(aux[j]);
                                D[i][j] = d;
                        }

                        line = read.readLine();
                        i++;

                } while (line != null);

        } catch (FileNotFoundException e) {
                e.printStackTrace();
        }

        /**
         * Escreve no arquivo matriz_distancia_nome_do_arquivo_.csv
         */
        try {
                CSVWriter writer = new CSVWriter(new FileWriter("matriz_dis-
tancia_" + filename + ".csv"));

                String[] entrada = new String[TAMANHO];
                for (int i = 0; i < TAMANHO; i++) {

                        for (int j = 0; j < TAMANHO; j++) {
                                entrada[j] = ""; // limpa a posicao
                                entrada[j] += D[i][j];
                        }
                        writer.writeNext(entrada);
                }

                writer.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
    }
```

O método *printCiclo* imprime no console o ciclo atual, associando o tamanho ao custo do ciclo.

```
    /**
     * Impressao em console
     */
    public void printCiclo() {
            System.out.print("\n\n( ");
            for (int i = 0; i < tamanhoCiclo;) {
                    System.out.print(ciclo[i] + 1);
                    if (++i < tamanhoCiclo)
                            System.out.print(", ");
            }
            System.out.println(" )\n\n TAMANHO = " + tamanhoCiclo + "\t" + "CUS-
TO = " + custoCiclo);
    }
```

O método *atualizarCusto* percorre o ciclo atual e atualiza o custo do ciclo.

```
    /**
     * Usado no metodo de insercao do ciclo, atualiza o custo a cada novo ci-
clo
     */
```

```
public void atualizarCusto() {
	custoCiclo = 0;

	if (tamanhoCiclo > 2) {
		for (int i = 0; i < tamanhoCiclo - 1; i++) {
			int c1 = ciclo[i];
			int c2 = ciclo[i + 1];
			double dzao = D[c1][c2];
			custoCiclo += dzao;
		}
		custoCiclo += D[ciclo[tamanhoCiclo - 1]][ciclo[0]];
	}

	printCiclo();
}
```

O método *inserirCiclo* adiciona um vértice ao vetor de ciclo.

```
/**
 * Metodo de insercao no vetor de ciclo, recebendo a posicao e seu
 * respectivo valor
 */
public void inserirCiclo(int posicao, int elemento) {
	int i = tamanhoCiclo - 1;
	while (i >= posicao) {
		ciclo[i + 1] = ciclo[i];
		i--;
	}
	tamanhoCiclo++;
	ciclo[posicao] = elemento - 1;
	atualizarCusto();

}
```

O método *gerarCicloInicial* adiciona dos vértices não visitados três vértices aleatórios ao clico.

```
/**
 * Gera ciclo incial com 3 vertices aleatorios.
 */
public void gerarCicloInicial() {
	Random random = new Random();

	int numero;
	for (int i = 0; i < 3; i++) {
		numero = random.nextInt(vnv.size());
		inserirCiclo(i, vnv.get(numero));
		cicloInicial[i] = vnv.get(numero);
		vnv.remove(numero);
	}
}
```

gerarCiclo é o método principal no processo. Nele acontecem as inserções aleatórias necessárias para construção do ciclo mínimo.

```
public void gerarCiclo() {
	Random random = new Random();

	int numero;
	gerarCicloInicial();

	/**
	 * Gera o restante do ciclo
	 */
	int vToAdd;
	while (vnv.size() > 0) {
		numero = random.nextInt(vnv.size());
```

```
                    vToAdd = vnv.get(numero);
                    double minCiclo = 99999999, auxCiclo;
                    int minPosicao = 0;

                    for (int j = 0; j < tamanhoCiclo - 1; j++) {
                          auxCiclo = custoCiclo - D[ciclo[j]][ciclo[j + 1]] +
D[ciclo[j]][vToAdd - 1]
                                      + D[vToAdd - 1][ciclo[j + 1]];
                          if (auxCiclo < minCiclo) {
                                minCiclo = auxCiclo;
                                minPosicao = j;
                          }
                    }

                    // apos inserir no vetor de ciclo eu retiro o do array de ver-
tices
                    // nao visitados
                    inserirCiclo(minPosicao, vToAdd);
                    vnv.remove(numero);

              }

        }
```

O método *limparCiclo* auxilia na obtenção dos resultados, limpando o vetor de ciclo a cada nova iteração (*k*)

```
        /**
         * reinicializa o vetor ciclo
         */
        public void limparCiclo() {
              ciclo = new int[TAMANHO];
        }
```

A execução de todo o algoritmo se concentra no método principal, main. A primeira instância do objeto tipo Grafo servirá para armazenar todas as variáveis e realizar as operações listadas anteriormente. Variáveis auxiliares são definidas para obtenção dos dados da tabela de experimentos.

```
        public static void main(String[] args) throws IOException {

              Grafo grafo;  // Objeto base das operacoes
              double res = 999999, max = 0, med = 0;  // res armazenara o mínimo
dos custos finais, max o custo maximo e med o valor medio
              int[] init = new int[3];  // Armazenara o ciclo incial referente ao
resultado minimo
              int k = 200;  // numero de execucoes do experimento
              long tempo;  // tempo de execucao do experimento
```

Em sequencia iniciamos a leitura do arquivo txt atribuindo às variáveis do grafo os valores lidos, por exemplo, o tipo da matriz e a quantidade de vértices.

```
              try {
                    String filename = "tsp10t3.txt"; // Chamar o arquivo.
                    BufferedReader read = new BufferedReader(new FileReader(file-
name));
                    int n, tipo;

                    String line = read.readLine();
                    String[] temp = line.split("  ");
                    n = Integer.parseInt(temp[0].split("=")[1]);
                    tipo = Integer.parseInt(temp[1].split("=")[1]);
                    System.out.println("N: " + n + " - tipo: " + tipo);
                    grafo = new Grafo(n);
```

```
read.close();
```

Dependendo do tipo do arquivo (*t1*, *t2*, *t3*), eu direciono o mesmo para seu respectivo método de geração da matriz.

```
switch (tipo) {
case 1:
	grafo.gerarMatrizTipo1(filename);
	break;

case 2:
	grafo.gerarMatrizTipo2(filename);
	break;
case 3:
	grafo.gerarMatrizTipo3(filename);
	break;

default:
	break;
}
```

Com a matriz instanciada, se dá a geração do ciclo *k* vezes, a fim de obter os resultados para popular a tabela de experimentos. A cada iteração será atribuído às variáveis seus respectivos valores.

```
/**
 * Preenche as variaveis resultado, media, maximo e
 * o tempo de execucao
 */
tempo = System.currentTimeMillis(); // Tempo inicial
for (int i = 0; i < k; i++) {
	grafo.gerarCiclo();

	if (grafo.custoCiclo < res) {
		res = grafo.custoCiclo;
		init = grafo.cicloInicial;
	}
	if (grafo.custoCiclo > max) {
		max = grafo.custoCiclo;
	}

	med += grafo.custoCiclo;

	/**
	 * Operacoes para validar a proxima geracao de ciclo
	 */
	grafo.limparCiclo();
	grafo.vnv.clear();
	for (int j = 1; j <= grafo.TAMANHO; j++) {
		grafo.vnv.add(j);
	}
	grafo.custoCiclo = 0.0;
	grafo.tamanhoCiclo = 0;
}
med /= k; // Valor medio
tempo = System.currentTimeMillis() - tempo; // Tempo final da
execucao
```

Por fim, será impresso no console os resultados finais do algoritmo

```
System.out.println("\n\n Ciclo inicial: " + init[0] + '-' +
init[1] + '-' + init[2]);
System.out.println("Maximo: " + max);
System.out.println("Media: " + med);
System.out.println("Resultado: " + res);
System.out.println("Tempo: " + tempo);
```

```
            } catch (FileNotFoundException e) {
                  e.printStackTrace();
            }
      }
}
```

2 Resultados Obtidos

Nº	Instância	N	Início	Máxima	Média	Solução	Tempo (s)
1	tsp10t3	10	1 – 6 - 8	385,0	292,67	211,0	0,326
2	tsp12t2	12	11 – 12 – 4	81,08	66,96	49,91	0,55
3	tsp20t3	20	4 – 16 – 9	5597,0	3910.64	2531,0	0,592
4	tsp58t1	58	50 – 57 – 54	141.398,0	84.481,245	56.353,0	2,057
5	tsp73t3	73	26 – 39 – 54	112.293,0	82.637,64	65.780,0	3,995
6	tsp225t2	225	159 – 120 – 170	24.892,9	16.637,72	12.618,42	27,569
7	tsp16t2	16	11 – 13 – 10	174,83	125,6	97,44	0,416
8	tsp29t1	29	24 – 18 – 7	4.839,0	3.527,28	2.687,0	1,838
9	tsp280t2	280	93 – 152 – 248	18.518,85	12.417,93	9.580,09	42,343

2.1 Comentários sobre os resultados obtidos

Foi observado que o critério aleatório gera flutuações entre os resultados, por não ser um método guloso. No entanto o resultado aproximado foi satisfatório para um número de 200 iterações apresentando um bom custo/benefício entre o tempo de execução e a solução obtida.

3 Conclusão

O método da inserção aleatória associado ao Problema do Caixeiro Viajante obteve resultados satisfatórios, mas por não ser um método determinístico conclui-se que não há grandes ganhos em sua aplicação à situações que exigem exatidão dos resultados.

4 Referências Bibliográficas