Christian Takashi Nakata R.A: 90558
Kevin Levrone Rodrigues Machado Silva R.A: 89275

Modelagem e Otimização Algorítmica

Maringá
2017

Christian Takashi Nakata R.A: 90558
Kevin Levrone Rodrigues Machado Silva R.A: 89275

Modelagem e Otimização Algorítmica

Relatório apresentado como requisito parcial para obtenção de nota na disciplina de Modelagem e Otimização Algorítmica, ministrada pelo Professor Dr. Ademir Constantino.

Universidade Estadual de Maringá – UEM

Departamento de Informática – Ciência da Computação

Maringá

2017

Sumário

1 Contextualização do problema

1.1 Introdução aos algoritmos genéticos

Os algoritmos geneticos utilizam conceitos provenientes do princıpio de seleção natural para abordar uma serie ampla de problemas,(em especial, problemas de otimização).Genéricos e de fácil adaptação, suas consistências são técnicas amplamente estudadas e utilizadas em diversas areas.

1.2 Funcionamento

O funcionamento dos Algoritmos Genéticos é inspirado na maneira que o Darwinismo explica o processo de evolução das espécies.Inventados por John Holland nos anos 60, e desenvolvidos por seus alunos da universidade de Michigan(nos anos 70), os Algoritmos Genéticos surgiram com o objetivo de serem usados como uma ferramenta de otimização para problemas na engenharia.A crença de Holland era que, eventos singulares da evolução natural poderiam ser importados e implementados aos sistemas computacionais.Consequentemente, uma versão computacional dos processos de evolução poderia ser a solução para um problema que apresentasse as mesmas características evolutivas.

O sistema utilizava uma cadeia de bits normalmente denominada "Indivíduo", constituída por cromossomos.Como uma analogia a lei natural, o sistema passava por um tipo de "evolução", até encontrar a melhor combinação cromossômica (solução) para resolver um determinado tipo de problema.

A proposta dos Algoritmos Genéticos é encontrar soluções aproximadas para problemas que envolvem alta complexidade computacional (variante entre NP-Completo e NP-Difícil).Para tal, é feita uma simulação evolutiva, em que o sistema não possui informações sobre o problema em questão, apenas sobre a função objetivo.

Segundo o conceito visto em AGUIAR(1996), um algoritmo genético genérico possui o seguinte funcionamento:

1) Uma população de cromossomos se mantém ao longo de todo o processo;

2) A cada um dos cromossomos associa-se um valor de adaptação que está diretamente relacionado com o valor da função objetivo a tomar

3) Cada cromossomo codifica um ponto no espaço de busca do problema

4) dois cromossomos são selecionados de acordo com o seu valor de adaptação para serem os geradores de duas novas configurações mediante um processo de reprodução.

5) estas novas configurações reservam seu espaço na nova geração.Esta parte do processo é repetida quantas vezes forem necessárias.

Conforme DAVIS(1991), o formato de um algoritmo genético pode ser descrito através dos seguintes componentes:

1) Uma forma de representação "cromossômica"das configurações assumidas no problema.

2) Parâmetros de entrada do algoritmo genético (tamanho populacional, número de gerações, taxas relativas aos operadores genéticos, entre outros);

3) Uma forma de criação ou inicialização das configurações assumindo a ideia de população inicial;

4) Uma função avaliativa que permita ordenar, classificar, e inserir valor aos cromossomos de acordo com o objetivo do algoritmo;

5)Operadores genéticos para geração, produção e/ou alteração da composição dos cromossomos durante a reprodução.

1.3 Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é um clássico exemplo de problema de otimização combinatória.Este problema é descrito formalmente: seja G = (N,E) um grafo em que N = 1, ..., n é o conjunto de nós e E = 1,..,m é o conjunto de arestas de G.Os custos cij, associam-se a cada aresta que interliga os vértices i e j. O problema consiste em localizar um caminho que passe por todos os vértices uma única vez.Ou seja, deve-se localizar o menor ciclo Hamiltoniano do Grafo G, em que a medida do ciclo é calculada pela soma dos custos das arestas que formam o ciclo.

O PCV proposto é simétrico, ou seja, cij = cji.Além disso, os grafos são completos, onde cada nó se liga a todos os outros nós.Para obter a solução do problema, é preciso processar todas as possibilidades de caminho, tornando a complexidade do problema na ordem de fatorial.Em outras palavras, este é um problema nível NP-Difícil, o que justifica o uso de Algoritmos Genéticos.

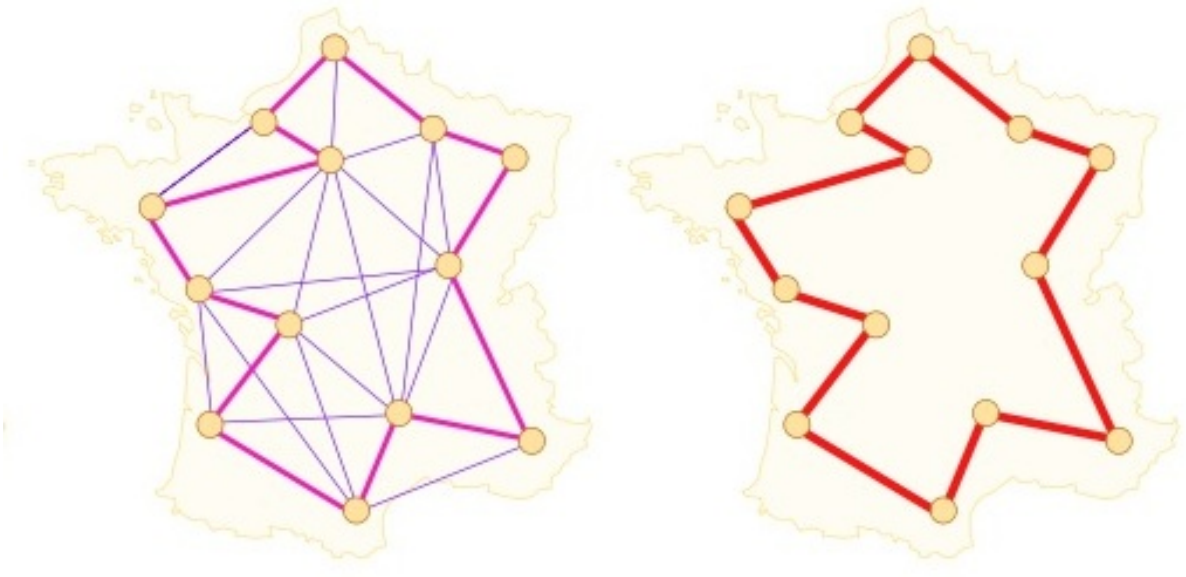

Figura 1: Exemplo de solução do PCV em um grafo

2 Desenvolvimento da solução

2.1 Ambiente de desenvolvimento

O algoritmo genético foi implementado na linguagem de programação C++, em um sistema operacional Linux com a distribuição Ubuntu 16.04.Devido as restrições do sistema RunCodes, o programa foi implementado em apenas um arquivo do tipo .cpp.Em outras palavras, o programa principal não receberá nenhum arquivo header.

2.2 Estruturas de Dados utilizadas

A seguir se encontram as representações algorítmicas das estruturas utilizadas para representar e resolver o problema.A estrutura "pega dados"armazena o índice dos vetores e suas respectivas coordenadas cartesianas(x,y).O indivíduo gerado possui uma sequência representada por um vetor e um custo representando o custo do Caixeiro Viajante para essa configuração.Para a lista de vértices, população e filhos gerados também foram utilizadas filas(vetores).

```
struct caso{
  int vertice;
  float x;
  float y;
};

struct individuo{
  vector<int> codigo;    // Estrutura que armazena um vetor de todos os
vertices pegos como entrada, e calcula o fitness.
  float fitness;

  bool operator < (const individuo& i) const { // Ordena populacao por
fitness
     return (fitness < i.fitness);
  }
};

vector<caso> grafo; // armazena todos os v rtices e suas respectivas
coordenadas
```

```
vector<individuo> vet_populacao; // Vetor que armazena a popula  o
vector<individuo> todos_filhos; // Vetor que armazena os filhos gerados.
```

2.2.1 Cruzamento utilizado

Para a função responsável pelo CrossOver, foi utilizado o cruzamento em dois cortes, devido ao bom ganho de variabilidade genética e bons resultados obtidos, o que tornou possível a geração de filhos (um máximo de 6 filhos) mais adaptados.A seguir, encontra-se a sua implementação algorítmica, onde i1 e i2 representam as 2 posições onde haverão os dois "cortes"no vetor para definir a recombinação de bits para gerar novos filhos.

```
    do{
    i1 = rand() % (n_vertice) + 1;
    i2 = rand() % (n_vertice) + 1;
  }while(i2 <= i1);
```

2.2.2 Mutação

Mutações permitem que cromossomos gerados tenham informações (características) diferentes dos cromossomos antecessores que os geraram, permitindo assim, que os processos de recombinação introduzam cromossomos diferentes na população, por meio da combinação do material genético dos seus geradores (pais).

A mutação dos cromossomos foi desenvolvida segundo a indagação: "Quantos vértices serão trocados de lugar em cada um dos filhos gerados?"O resultado do desenvolvimento se encontra no código localiado na página seguinte:

```
void Mutacao(int taxamutacao, int n_vertices){
  int i1, i2, aux, j,k=1;

 while(k<=taxamutacao){
   do{
     i1 = rand()%(n_vertices);
     i2 = rand()%(n_vertices);
   }while(i1==i2);

   for(j=0;j<todos_filhos.size();j++){
     aux = todos_filhos[j].codigo[i1];
     todos_filhos[j].codigo.at(i1) = todos_filhos[j].codigo.at(i2);
     todos_filhos[j].codigo.at(i2) = aux;
```

```
        break;
      }
       k++;

    for(j=0;j<todos_filhos[0].codigo.size();j++){
        cout << "|" << todos_filhos[0].codigo[j] << "|" << endl;
    }
    cout << endl << endl;
  }
}
```

3 Análise dos Resultados

Nos testes a seguir foi aplicado um estudo para demonstrar o comportamento do algoritmo aplicado aos grafos utilizando nove casos de teste. Os primeiros três casos de cada tabela testam o valor de população(que gera uma resposta mediana), então, para que não houvesse interferência dos outros parâmetros, a mutação foi colocada em zero e o número de gerações em mil. Dos casos quatro até seis são os testes de mutação. Para isso é selecionado o melhor resultado da população encontrado nos testes anteriores e o testa novamente com a mutação em diferentes parâmetros. Para não sofrer influência do número de gerações, esse parâmetro se mantém em mil. Dos casos sete até nove é testado o número de gerações. Para tal, o melhor número da população obtido e a melhor taxa de mutação encontrada são selecionados e o único parâmetro que varia é o número de gerações. Portanto, as últimas três linhas da tabela representam os melhores valores encontrados.

Tabela 1: Grafo de 100 vértices - Sol. ótima: 21282

	População	**Mutação**	**Gerações**	**Melhor Encontrado**
Caso 1	15	0	1000	158575
Caso 2	25	0	1000	157024
Caso 3	50	0	1000	152492
Caso 4	50	1	1000	62540.4
Caso 5	50	2	1000	72529.5
Caso 6	50	3	1000	76186.4
Caso 7	50	1	25000	34090.3
Caso 8	50	1	50000	33810.8
Caso 9	50	1	100000	33810.8

Tabela 2: Grafo de 130 vértices - Sol. ótima: 6110

	População	**Mutação**	**Gerações**	**Melhor Encontrado**
Caso 1	15	0	1000	44138
Caso 2	25	0	1000	38984.8
Caso 3	50	0	1000	41430.1
Caso 4	50	1	1000	19375.9
Caso 5	50	2	1000	19204.9
Caso 6	50	3	1000	19435.7
Caso 7	50	1	25000	7434.01
Caso 8	50	1	50000	6758.37
Caso 9	50	1	100000	6142.41

Tabela 3: Grafo de 657 vértices - Sol. ótima: 48912

	População	**Mutação**	**Gerações**	**Melhor Encontrado**
Caso 1	15	0	1000	840429
Caso 2	25	0	1000	831417
Caso 3	50	0	1000	813536
Caso 4	50	1	1000	598594
Caso 5	50	2	1000	589847
Caso 6	50	3	1000	565494
Caso 7	50	1	25000	215891
Caso 8	50	1	50000	174382
Caso 9	50	1	100000	149451

Tabela 4: Grafo de 2103 vértices - Sol. ótima: 80450

	População	**Mutação**	**Gerações**	**Melhor Encontrado**
Caso 1	15	0	1000	3.20003e+06
Caso 2	25	0	1000	3.2009e+06
Caso 3	50	0	1000	3.17112e+06
Caso 4	50	1	1000	2.83511e+06
Caso 5	50	2	1000	2.71433e+06
Caso 6	50	3	1000	2.71624e+06
Caso 7	50	1	25000	1.28043e+06
Caso 8	50	1	50000	1.0176e+06
Caso 9	50	1	100000	835856

Tabela 5: Grafo de 5934 - Sol. ótima: 556045

	População	**Mutação**	**Gerações**	**Melhor Encontrado**
Caso 1	15	0	1000	4.17165e+07
Caso 2	25	0	1000	4.14062e+07
Caso 3	50	0	1000	4.15851e+07
Caso 4	50	1	1000	3.95104e+07
Caso 5	50	2	1000	3.87815e+07
Caso 6	50	3	1000	3.84115e+07
Caso 7	50	1	25000	2.23466e+07
Caso 8	50	1	50000	1.76693e+07
Caso 9	50	1	100000	1.38167e+07

Tabela 6: Grafo de 14051 vértices - Sol. Ótima 469385

	População	**Mutação**	**Gerações**	**Melhor Encontrado**
Caso 1	15	0	1000	4.13782e+07
Caso 2	25	0	1000	4.14836e+07
Caso 3	50	0	1000	4.14775e+07
Caso 4	50	1	1000	4.03777e+07
Caso 5	50	2	1000	4.03183e+07
Caso 6	50	3	1000	4.02323e+07
Caso 7	50	1	25000	2.7915e+07
Caso 8	50	1	50000	2.27732e+07
Caso 9	50	1	100000	1.80643e+07

4 Conclusão

Observa-se que os parâmetros de teste possuem alta dependência com o tamanho do problema abordado.Para grafos pequenos é melhor uma população pequena com uma taxa mutacional também pequena, e um número de gerações mediano.Para os grafos maiores, é notável que um tamanho de população, taxa de mutação e número de gerações maiores retornou resultados superiores.Conclui-se então, que parâmetros proporcionais ao tamanho do problema em questão são fundamentais para encontrar a sua solução.

5 Referências Bibliográficas

http://moodle.din.uem.br/pluginfile.php/12091/mod$_r esource/content/2/AG-TPO_3.pdf$

http://moodle.din.uem.br/pluginfile.php/12092/mod$_r esource/content/1/AG-BL.pdf$

http://www.inf.ufrgs.br/ alvares/INF01048IA/ApostilaAlgoritmosGeneticos.pdf