Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Computação Natural

Trabalho 2: Colônia de Formigas

Aluna: Larissa Fernandes Leijôto
Email: larissaleijoto@ufmg.br
Professor: Gisele Lobo Pappa

Belo Horizonte,
22 de dezembro de 2015

Sumário

1 Introdução

Otimização por colônia de formigas(ACO, *Ant Colony Optimization*) é uma metaheurística que compõe a classe de algoritmos que são baseados em inteligência de enxames. O ACO é utilizado para obter soluções aproximadas em problemas de otimizações cujas soluções exatas são impossíveis de computar em tempo polinomial. Por isso, seu uso é indicado para determinar a solução do problema de encontrar o caminho máximo em uma grafo. O ACO é inspirado em como as formigas se comportam no meio ambiente; seu funcionamento é baseado na comunicação por meio de uma substância denominada feromônio.

Ao encontrar comida, uma formiga deixa um rastro no caminho de volta para a colônia. E esse rastro é seguido por outras formigas ao voltarem à colônia. Quando alimento acaba, as trilhas não são mais modificadas pelas formigas que retornam, e o cheiro (feromônio) se perde. Esse comportamento ajuda as formigas a se adaptarem a mudanças em seu meio. Se um caminho estabelecido como fonte de comida é bloqueado por um obstáculo, as formigas o deixam para explorar novas rotas. E, se a busca é bem sucedida, a formiga retorna e marca um novo rastro para a rota mais curta. Trilhas bem sucedidas são seguidas por mais formigas, e cada uma as reforça com mais feromônio.

Esse comportamento das formigas é simulado no computador, em que cada formiga simulada representa uma possível solução para o problema abordado. O ambiente que será percorrido por elas é representado por uma matriz de adjacências, que armazena a concentração inicial de feromônio sob o grafo de entrada. À medida em que as formigas vão percorrendo o seu caminho, deixam uma taxa constante de feromônio, acumulada à taxa anterior. Esse feromônio é indica caminhos mais frequentemente percorridos, ou seja: quanto maior a concentração de feromônio, mais formigas percorrem o caminho; e assim, maior a probabilidade de novas formigas o percorrerem. Cada formiga guarda o caminho por ela percorrido; consequentemente, a avaliação dele, calculada como a soma das distâncias das arestas que o compõem.

2 Modelagem e solução proposta

Nesta seção será discutida a modelagem das formigas, bem como sua avaliação. Apresentaremos também como os parâmetros do algoritmo foram utilizados e quão sensível o programa é a eles. O problema em questão é encontrar o caminho máximo entre dois vértices de um grafo. Dados vértices i e f, sendo i o vértice inicial e f o vértice final de um grafo direcionado com distâncias positivas nas aresta, o problema consiste em encontrar um caminho máximo de i a f.

2.1 O algoritmo de colônia de formigas

Formigas artificiais são heurísticas construtivas. Elas constroem soluções de forma probabilística utilizando duas informações. Primeiro, a trilha de feromônio, que muda dinamicamente durante a execução do programa, refletindo assim a experiência já adquirida durante a busca. Segundo, a informação heurística específica do problema a ser resolvido. O algoritmo implementado neste trabalho segue o fluxo básico de um algoritmo de otimização por colônia de formigas. O pseudo-código pode ser visto no Algoritmo 1.

Algoritmo 1: Esboço do algoritmo implementado

Entrada: Conjunto de vértices
Saída: Melhor caminho

```
1 Inicializar formigas;
2 repita
3 |   calcular probabilidades;
4 |   evaporar feromônio das arestas;
5 |   atualizar feromônio das arestas;
6 |   atualizar fomigas;
7 até (Alcançar critério de convergência);
```

2.2 Implementação

O algoritmo de otimização por colônia de formigas foi implementado na linguagem C++, com dependência das bibliotecas padrão do c++11 (iostream, fstream, string, vector, e chrono).

2.2.1 Formiga

Para a representação das formigas foram utilizadas *structs*, compostas pelo caminho que as formigas trilham, e pelas distâncias percorridas por elas. Para a representação do caminho foi utilizado um vetor denominado *path*; assim, cada formiga possuirá seu próprio caminho de acordo com as probabilidades de transição. A variável *distance* é a soma das distâncias das arestas que a formiga esteve.

```
1 struct Ant
2 {
3   double distance;
4   vector<int> path;
5 };
```

2.2.2 Distâncias

Para a representação das distâncias entre a cidades utilizamos uma matriz de adjacência, cujas linhas e colunas representam distâncias entre as cidades.

```
vector<vector<double>> adjacencies;
```

2.2.3 Feromônio

Para a representação do ambiente que as formigas circulam, utilizamos uma matriz de feromônio. Essa substância é depositada pelas formigas à medida em que percorrem o grafo. Sendo assim, cada posição da matriz associa uma quantidade de feromônio à aresta correspondente.

```
vector<vector<double>> pheromone;
```

2.2.4 Probabilidades

A probabilidade da formiga k que está no vértice i escolher a cidade j é dada pela regra demonstrada na Equação 1.

$$p^k{}_{ij} = \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum (\tau_{ij})^\alpha (\eta_{ij})^\beta} \tag{1}$$

- τ_{ij} é feromônio associado a aresta (i,j).

- α e β são parâmetros para determinar a influência do feromônio e da informação heurística.

Associada à aresta (i, j) existe um valor heurístico η_{ij} dado pela Equação 2, que representa a atratividade da formiga visitar a cidade i depois de visitar a cidade j. O valor η_{ij} é inversamente proporcional a distância d_{ij} entre as cidades i e j.

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{2}$$

2.2.5 Métodos importantes

```
void antOptimization(int , int, int);
void MakeGraphDistances(vector<vector<double>>&);
void init_ant(vector<Ant>&, int );
void seed_initial_pheromone(vector<vector<double>>&,
     vector<vector<double>>&);
void build_solution(vector<Ant> &, vector<vector<double>> &,
     vector<vector<double>> &);
void pheromone_evaporates(vector<vector<double>> &);
void update_pheromone(vector<Ant> &, vector<vector<double>> &);
```

- **antOptimization:** Método principal do algoritmo implementado, realiza a chamadas de todos os outros métodos implementado no programa
- **MakeGraphDistances:** Monta a matriz de adjacências para ser usada pelo algoritmo.
- **init_ant:** Inicaliza as formigas e as trilhas delas de forma aleatória
- **seed_initial_pheromone:** Inicializa o feromônio baseado em uma fórmula para o cálculo do mesmo.
- **build_solution:** Constrói a trilha que será percorrida.
- **pheromone_evaporates:** Subtrai uma taxa constante da matriz de feromônio
- **update_pheromone:** Atualiza o feromônio de cada cidade de acordo com os caminhos que já foram percorridos pelas formigas.

3 Arquivos, compilação e execução

3.1 Arquivos

- **main.cpp:** Arquivo principal que realiza a chamadas dos principais métodos.
- **util.cpp:** Métodos úteis que foram utilizados na implementação do algoritmo.
- **database.cpp:** Arquivo utilizado para a leitura da base de dados.
- **util.h:** Header da classe util
- **database.h:** Header da database

- **antOptimization.cpp** Arquivo princial onde é implementado o alogritmo de colônia de formigas
- **antOptimization.h** Header do algoritmo de colônia de formigas

3.2 Compilação

A compilação do programa pode ser feito por meio de um Makefile contido na pasta raiz do trabalho. Outro Makefile que pertence a pasta src é executado, assim que o primeiro Make é acionado. Portanto, não é necessário executá-lo uma vez que, sua execução é a partir do que está contido na pasta raiz.

- $tp2 - naturalComputing/make$

3.3 Execução

Para que o programa seja executado é necessário o seguinte comando:

- $bin/antOptimization[nAnts, nIteration, evaporationRate, input/File]$

4 Experimentos

Os experimentos foram executados em um computador com processador Intel Core I5 com 1.70GHz, memória DDR3 de 8 GB e sistema operacional Ubuntu versão 14.04.3 LTS.

4.1 Metodologia

O algoritmo implementado é influenciado pelos parâmetros apresentado na Tabela 1. Por opção de tempo de processamento, o algoritmo foi implementado de forma que pudesse gerar soluções inválidas. Isso possibilitou encontrar soluções em menor tempo em relação à abordagem que utiliza *backtraking* para gerar somente soluções válidas.

Para a análise do resultado, foi criado um *script* que varia o número de formigas, a quantidade de iterações e a taxa de evaporação. A semente utilizada durante essas variações foi fixada, – a saber, 1834 –, e a configuração de parâmetros que obteve o melhor resultado foi utilizada para executar o algoritmo na fase final. Essa abordagem foi adotada para que fosse possível analisarmos a convergência do algoritmo e significância do resultado. Para as três entradas definidas na especificação: limitamos o número de iterações de 10 a 500; restringimos a quantidade de formigas ao número de vértices de cada entrada; e a taxa de evaporação foi variada de 0.1 a 0.9. As constantes associadas ao

algoritmo foram definidas utilizando a Entrada 2, sobre a qual realizamos uma sequência de testes com diversas variações.

Tabela 1: Parâmetros

Parâmetros
Influência do feromônio
Influência da visibilidade
Número de formigas
Constante de acréscimo de feromônio
Constante de inicialização do feromônio

5 Resultados

Para determinar a significância dos resultados, o algoritmo foi executado 30 vezes sobre cada entrada disponibilizada. Utilizando todas as execuções, foram calculadas média, variância e desvio padrão. O resultado para tais métricas é apresentado na tabela 2.

Tabela 2: Métricas

	Entrada 1	Entrada 2	Entrada 3
Média	960.76	166.26	9524.40
Desvio Padrão	5.50	1.36	272.21
Variância	30.32	1.85	74101.28
Max	973	167	9753
Min	953	163	8941

Observando a execução de um dos experimentos realizados, podemos perceber que o algoritmo converge rapidamente para uma solução. No experimento demostrado na Figura 1, o algoritmo convergiu para um ótimo local. Isso ocorreu também nos outros 30 experimentos. Sendo assim, em nenhuma das execuções para essa entrada o algoritmo conseguiu encontrar o ótimo. Apesar disso, em algumas execuções a solução obtida foi bastante próxima, pois o máximo alcançado foi 973 e o valor ótimo informado é de 990.

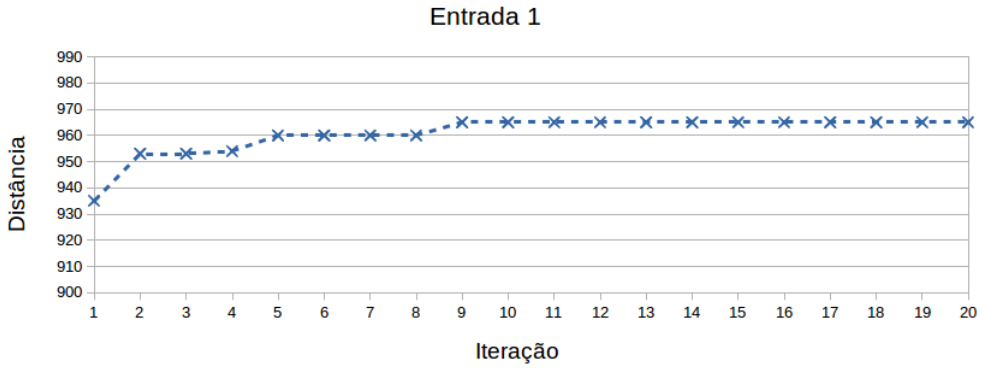

Figura 1: Demostração de uma execução para a primeira entrada.

No experimento demostrado na Figura 2, podemos observar que o algoritmo novamente convergiu muito rápido. No entanto, como essa entrada é relativamente mais fácil que a anterior, concluímos que a solução obtida é relativamente mais próxima do ótimo. O máximo alcançado foi de 167, sendo que a solução ótima é de 168.

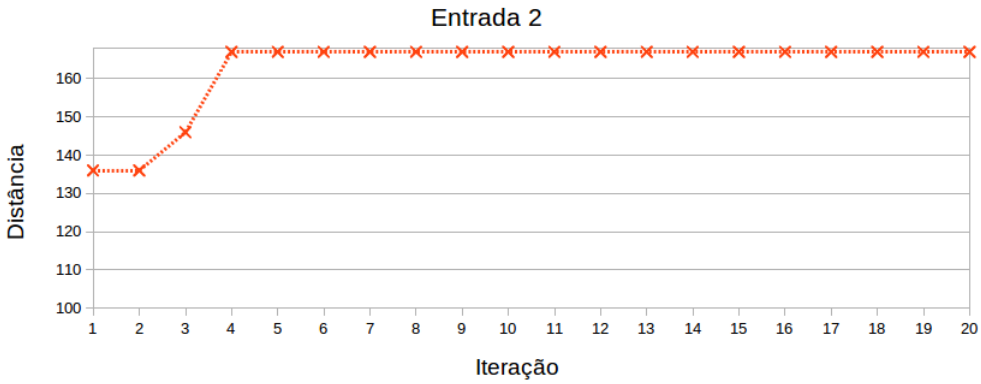

Figura 2: Demostração de uma execução para a segunda entrada.

O experimento realizado na última base seguiu o mesmo comportamento das anteriores, por isso pressupomos que o valor máximo encontrado seja um ótimo local. O valor máximo foi de 9753, e, como nessa base não possuímos o ótimo, não é possível dizer o quão acurada é a nossa solução. Como mostrado na Tabela 2, a variância entre as execuções foi muito alta, indicando que o algoritmo poderia ter sido executado por mais iterações. Entretanto, executá-lo por mais iterações não necessariamente melhoraria o resultado, e por isso achamos esse teste dispensável.

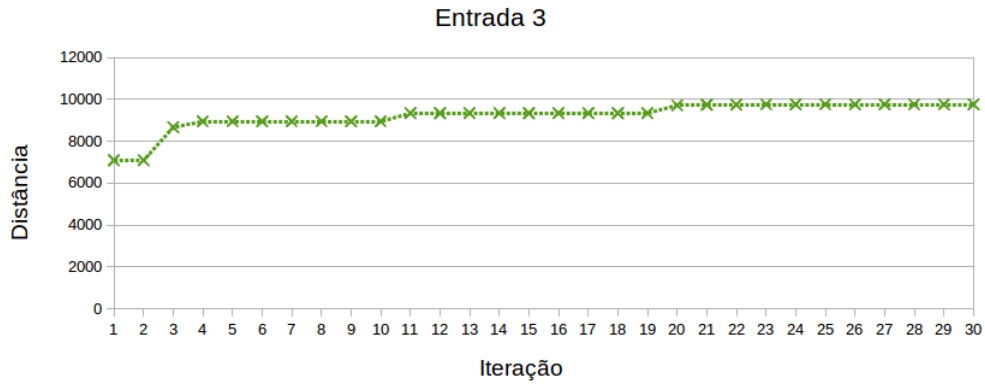

Figura 3: Demostração de uma execução para a terceira entrada.

6 Conclusão

Na implementação do algoritmo, optamos por permitir a geração de soluções inválidas, considerando o fato dessa abordagem ser significativamente mais rápida que estratégias que geram somente soluções válidas. Com os experimentos, podemos concluir que a geração das soluções inválidas não acarretou prejuízos às soluções geradas. Como foi descrito na seção de resultados, as soluções não atingiram o ótimo esperado. Entretanto, para a Entrada 2 o algoritmo chegou muito próximo do ótimo, e para a Entrada 1 o fator de erro não foi grande. Na Entrada 3, vimos que a variância das execuções foi muito alta, indicando que seria necessária uma maior quantidade de iterações. Vimos também que aumentar o número de iterações possivelmente não melhoraria o resultado, uma vez que esse já estava estagnado em uma solução há muitas iterações. Assim, concluímos que o algoritmo de colônia de formigas implementado cumpriu o seu papel: achar soluções próximas do ótimo para problemas de alta complexidade computacional.