# Documentação Trabalho Prático II de Computação Natural: *Longest Path Problem* usando *Ant Colony Optimization*

Ronald Davi Rodrigues Pereira
ronald.pereira@dcc.ufmg.br

Universidade Federal de Minas Gerais

03 de Novembro de 2018

## 1 Introdução

Em otimização por colônia de formigas, uma aplicação muito recorrente é a de se minimizar ou maximizar uma solução iterativamente por meio de uma representação em grafos. A convergência desses algoritmos se dá pela utilização de feromônios, pois um objeto que represente a formiga não possui inteligência o bastante para chegar à uma solução aproximada da ótima. Portanto, necessitamos que o comportamento emerja nesse meio à partir de diferentes estratégias de comunicação.

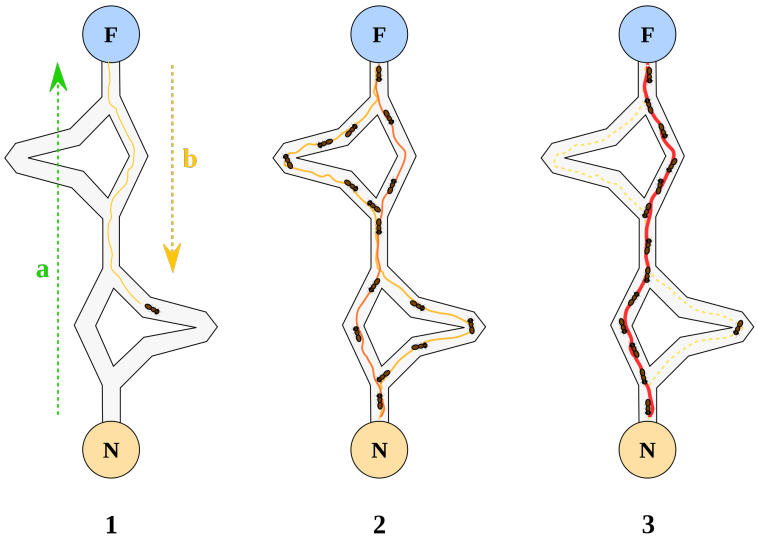


Figura 1 – Exemplo ilustrativo de uma otimização por colônia de formigas

Como qualquer outro algoritmo probabilístico[1], não é garantido que essa solução ótima será alcançada. Porém, por meio de heurísticas e métodos empíricos, é possível se garantir que pelo menos uma aproximação satisfatória da solução ótima foi alcançada, caso

sejam utilizados parâmetros que, em troca de tempo de execução e complexidade, reduzem ao máximo o erro das suas possíveis soluções.

O objetivo desse trabalho é de resolver o problema de *Longest Path Problem*[2] utilizando otimização por colônia de formigas[3] (*Ant Colony Optimization*). Portanto, buscamos uma solução que se aproxime da solução ótima do problema proposto através da iteratividade de uma população inicial que caminhará randomicamente. Vários métodos empíricos podem ser utilizados para a otimização desses algoritmos e isso será discutido e mostrado nesse documento para algumas bases de dados específicas.

# 2 Modelagem e Implementação

## 2.1 Descrição dos Arquivos

**longest_path.py** Programa principal que realiza o controle do fluxo de execução do programa

**ant_colony_optimization.py** Biblioteca customizada que realiza a construção da solução de acordo com uma função de transição probabilística

**ant.py** Biblioteca customizada que representa o objeto "formiga"e suas operações e atributos

**arg_parse_config.py** Biblioteca customizada que realiza a configuração do *parsing* de argumentos para o programa principal

**graph.py** Biblioteca customizada que realiza a leitura e representa o grafo

**pheromone_update.py** Biblioteca customizada que realiza a atualização dos feromônios no grafo (evaporação e depósito)

**requirements.txt** Arquivo que explicita todas as bibliotecas externas (não-padrões do Python 3) utilizadas na confecção do trabalho, bem como as suas versões

**tests.sh** Arquivo interativo para execução de testes repetidos automaticamente, bem como a separação em diferentes pastas das saídas do programa

**input/** Pasta que contém todas as entradas

**output/** Pasta que contém todos os testes executados

## 2.2 Estatísticas

Ao final de cada geração, são impressas as seguintes estatísticas:

**Iteration** Iteração atual

**Best Path Cost** Peso total do melhor caminho encontrado até aquela iteração

Como por exemplo:

```
0 113
1 128
2 147
3 147
4 147
5 147
6 147
7 147
8 147
9 147
10 147
```

Ao final da execução do algoritmo (quando o número de iterações é alcançado), são impressas as seguintes estatísticas:

**Path** Representação da melhor solução encontrada

**Best Path Cost** Peso total da melhor solução encontrada

Como por exemplo:

```
[1, 19, 15, 2, 12, 17, 18, 16, 6, 5, 4, 7, 13, 9, 8, 10, 11, 3, 14, 20]
163
```

## 2.3 Fitness

Para o cálculo da *fitness*, foi utilizado uma função de maximização do peso total da solução, que pode ser expressa como:

$$P^* = arg \max_{P \in P} \sum_{e_i \in P} w(e_i)$$

ou seja, queremos encontrar o caminho simples de $u$ a $v$ que maximize o peso total do caminho.

## 2.4 Decisões de Implementação

Serão demonstradas abaixo todas as decisões de implementação que foram possíveis de serem tomadas durante o desenvolvimento desse algoritmo, de forma a explicitar quais estratégias foram utilizadas por mim nesse trabalho.

### 2.4.1 Representação da Solução

Um caminho armazenado em cada formiga nesse problema representa sempre uma solução válida, ou seja, uma configuração de transições dos vértices de 1 a $n$. Para tal representação dessas formigas, foi escolhido uma abordagem por meio da instanciação de uma classe que possui como atributos a solução candidata e o seu peso total, bem como operações intrínsecas a cada formiga, como depositar feromônio e transicionar de um vértice para outro realizando as operações necessárias para tal.

### 2.4.2 Representação do Grafo

O grafo dado na entrada foi representado como uma lista de adjacência. Essa representação foi escolhida pelo fato de que, caso fosse realizada por meio de uma matriz de adjacência, ela seria muito esparsa. Uma representação mais simplificada pode ser vista abaixo:

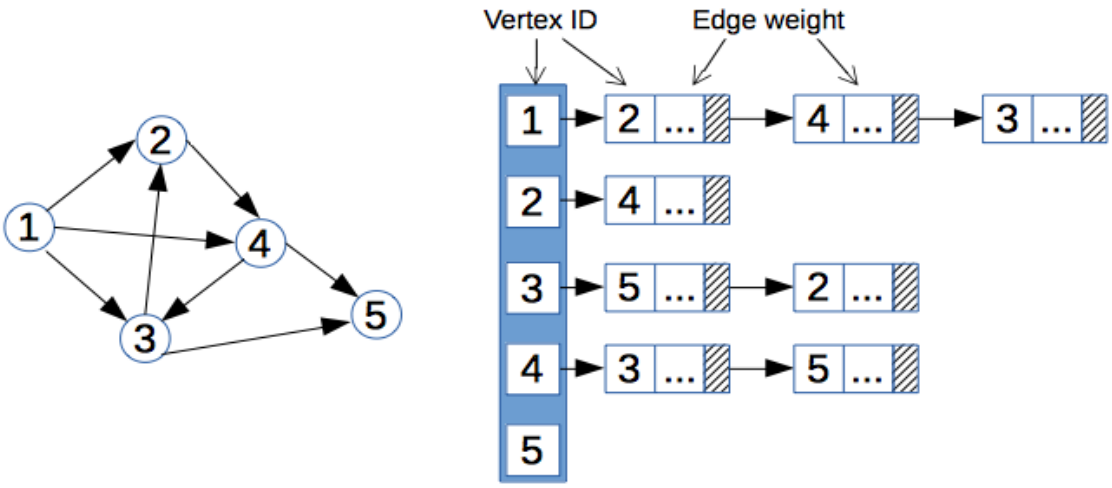


Figura 2 – Representação de um grafo por meio de uma lista de adjacência

#### 2.4.2.1 Cálculo da Transição Probabilística

Para cada formiga que deseja se movimentar para outro vértice à partir de um vértice atual, foi necessário construir uma função de transição probabilística que retornasse o próximo vértice a ser escolhido a partir de uma distribuição proporcional ao tamanho da aresta, representada como:

$$T_i = \sigma_i * W_i$$

sendo $T_i$ o tamanho da aresta $i$, $\sigma$ a quantidade de feromônio naquela aresta e $W$ o peso da aresta.

Desse modo, é realizado uma escolha aleatória por uma aresta que respeite essa distribuição e é retornado o vértice que essa aresta é ligada para que a formiga possa se mover para ele.

### 2.4.3 Soluções Inválidas

Nesse problema, em específico, foi necessário realizar um tratamento especial das soluções inválidas. Essas soluções se dão ao fato de que, possivelmente, uma formiga caminhe para um vértice distinto do vértice final e que todos os seus vértices vizinhos já tenham sido visitados por ela. Para tal, cada objeto que representa uma formiga possui um atributo do tipo boleano $validPath$. Essa variável é inicializada como $True$, de modo que o caminho seja válido inicialmente. Porém, caso essa proposição citada anteriormente aconteça, atribuímos $False$ à essa variável.

Desse modo, todas as operações que realizam utilizações dessas formigas como solução verificam esse atributo para que seja realizada com sucesso. Por exemplo, o depósito de feromônio ao final de cada iteração é realizada apenas pelas formigas que possuem esse atributo como $True$. Portanto, caso a solução seja inválida, a formiga é desconsiderada em qualquer operação que a deseje utilizar, evitando que o algoritmo convirja para uma solução inválida no final de sua execução.

### 2.4.4 Parâmetros e Execução

As variações de parâmetros foram implementadas de forma a facilitar testes e execuções distintas do algoritmo. Todos os parâmetros tidos como minimamente importantes foram externalizados para execução do programa principal com esses parâmetros passados por linha de comando.

Abaixo, um exemplo de uma execução com todos os parâmetros:

```
# Um menu de ajuda pode ser visualizado executando o comando:
# python3 longest_path.py -h
python3 longest_path.py -a 50 -e 0.1 -i 50 -k 1 input/datasets/graph1.txt
```

**-a** Número de formigas(*integer* $\in [1, \infty)$). Padrão: 50

**-e** Taxa de evaporação do feromônio (*float* $\in [0, 1]$). Padrão: 0.1

**-i** Número de iterações (*integer* $\in [1, \infty)$). Padrão: 50

**-k** Quantidade de k melhores formigas que depositarão feromônio (*integer* $\in [1, ants]$). Padrão: 1

## 3 Experimentos e Resultados

Os experimentos abaixo foram executados repetidas vezes variando os parâmetros designados por cada seção abaixo. Esses testes foram executados em um computador com processador Intel Core I7 de 3.60GHz e 8 núcleos de processamento, com 16GB de memória DDR4 e sistema operacional Ubuntu 18.04.1 LTS.

Vale ressaltar que, como os testes teriam que ser variados, a semente para as funções *random* foi desativada, a fim de conseguir alcançar diferentes resultados para cada execução do algoritmo. Para executar o *script* interativo de execução de testes, basta executar os comandos abaixo:

```
cd src/
chmod +x tests.sh
./tests.sh
# Você será perguntado de quantas repetições do teste deseja executar
```

Todos os testes abaixo foram executados com os valores padrões, com exceção do parâmetro que está sendo variado, conforme descrito abaixo:

```
-a | --ants = 50
-e | --evaporation_rate = 0.1
-i | --iterations = 50
-k | --k_ants = 1
```

e foram realizados em todos os grafos (*graph1*, *graph2*, *graph3*).

## 3.1 Número de formigas

O número de formigas variou no conjunto {10, 25, 50, 75, 100}. Conforme o número aumentou, o tempo de execução do algoritmo ficou muito maior, porém a solução encontrada no final foi relativamente melhor ($fitnessBest_{ants=10} \leq fitnessBest_{ants=100}$), conforme pode ser observado pelos gráficos abaixo:

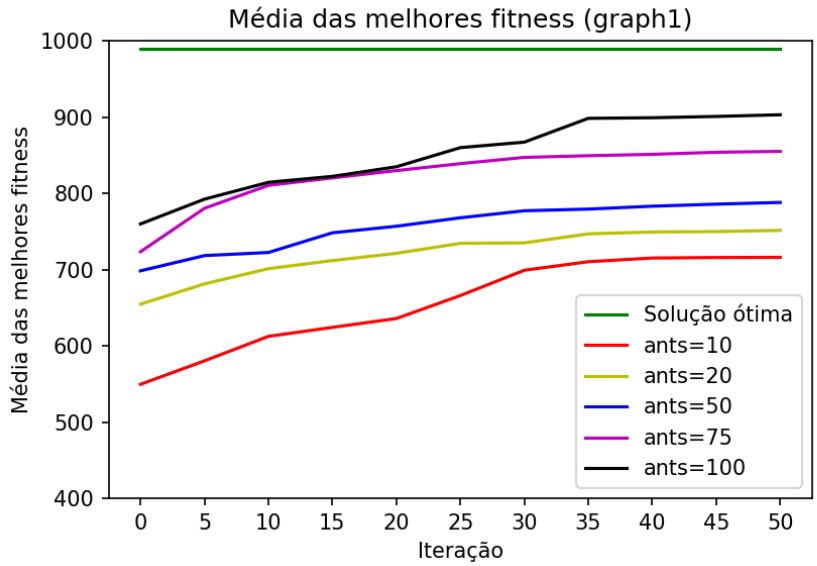


Figura 3 – Fitness do melhor caminho ao longo das iterações com a variação do número de formigas no *graph1*

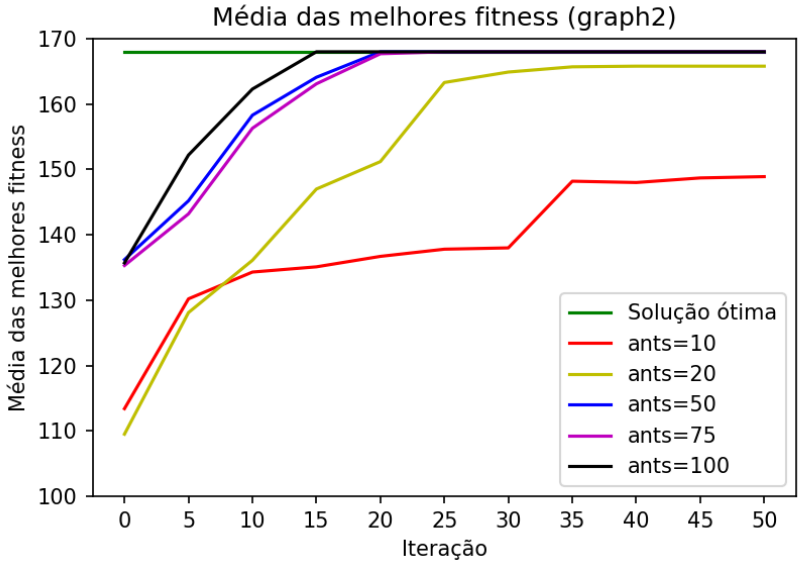


Figura 4 – Fitness do melhor caminho ao longo das iterações com a variação do número de formigas no *graph2*

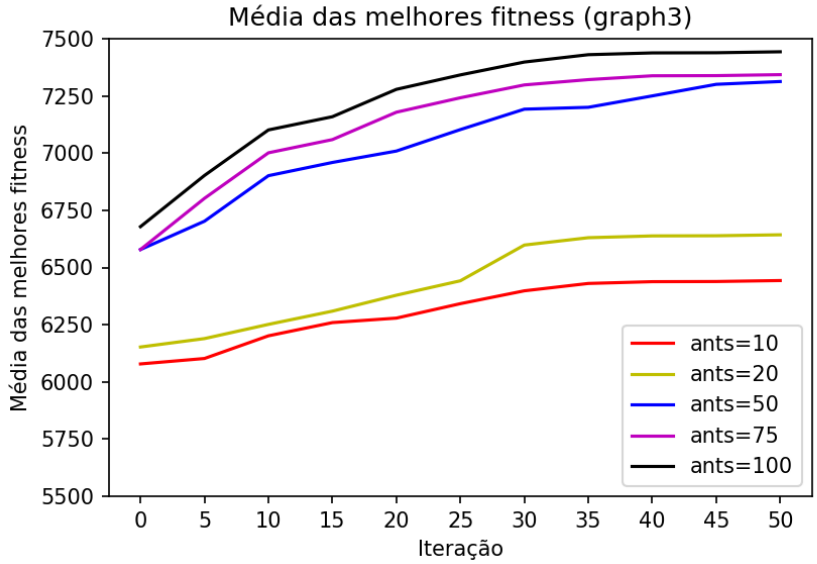


Figura 5 – Fitness do melhor caminho ao longo das iterações com a variação do número de formigas no *graph3*

Portanto, a variação desse parâmetro gera um comportamento na melhoria da solução que é facilmente observado. Desse modo, o número de formigas se mostrou um importante parâmetro a ser considerado.

## 3.2 Taxa de evaporação

A taxa de evaporação do feromônio variou no conjunto {0.01, 0.05, 0.1, 0.5, 0.8}. Conforme o número aumentou, o tempo de execução do algoritmo se manteve quase estável, porém as soluções encontradas no final foram variando proporcionalmente ao aumento dessa taxa, conforme pode ser observado pelo gráfico abaixo:

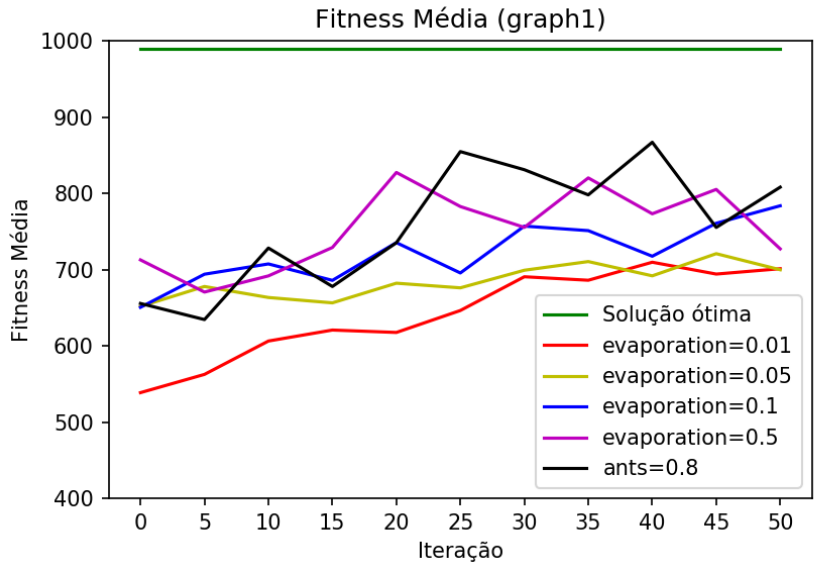


Figura 6 – Fitness média ao longo das iterações com a variação da taxa de evaporação no *graph1*

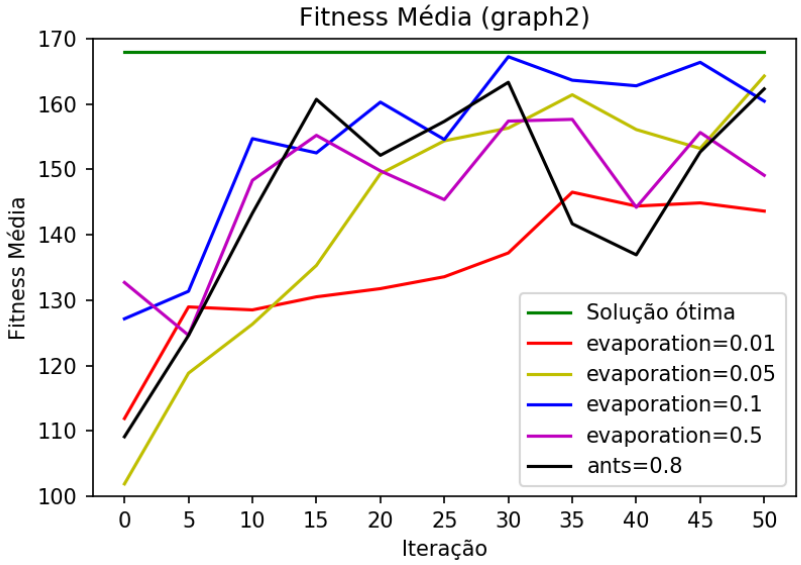


Figura 7 – Fitness média ao longo das iterações com a variação da taxa de evaporação no *graph2*

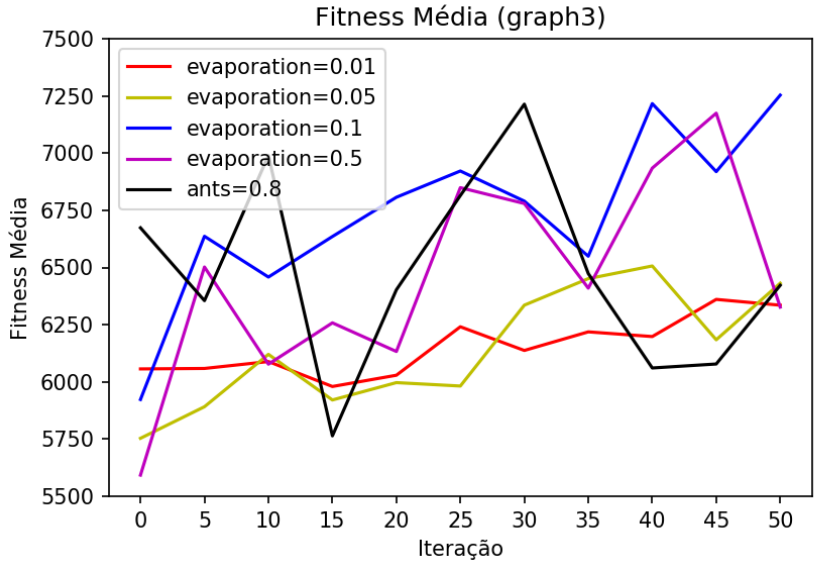


Figura 8 – Fitness média ao longo das iterações com a variação da taxa de evaporação no *graph3*

No entanto, essa melhoria foi pequena, quando comparado aos outros parâmetros que foram variados. Concomitantemente, devido à alta variância provocada pela maior quantidade desse parâmetro, ele se torna uma medida complicadora para se convergir para uma solução candidata a ótima.

## 3.3 Número de Iterações

O número de iterações variou no conjunto {10, 20, 50, 75, 100}. Conforme o número aumentou, o tempo de execução do algoritmo ficou um pouco maior, porém a solução encontrada no final foi relativamente melhor ($fitnessBest_{iterations=10} \leq fitnessBest_{iterarions=100}$), conforme pode ser observado pelo gráfico abaixo:

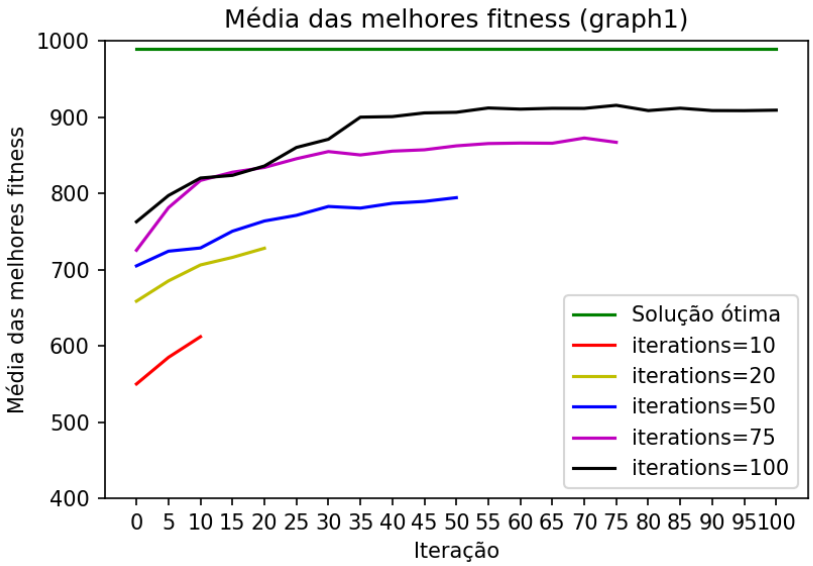


Figura 9 – Fitness do melhor caminho ao longo das iterações com a variação do número de iterações no *graph1*

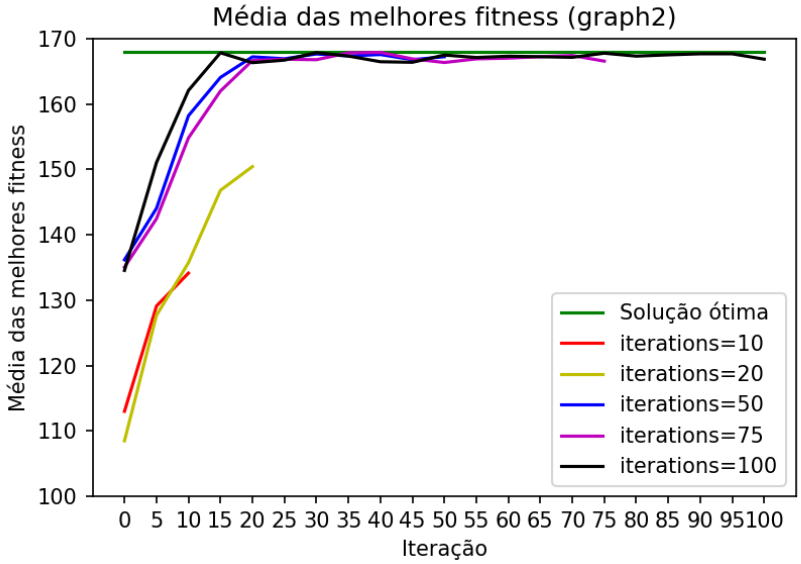


Figura 10 – Fitness do melhor caminho ao longo das iterações com a variação do número de iterações no *graph2*

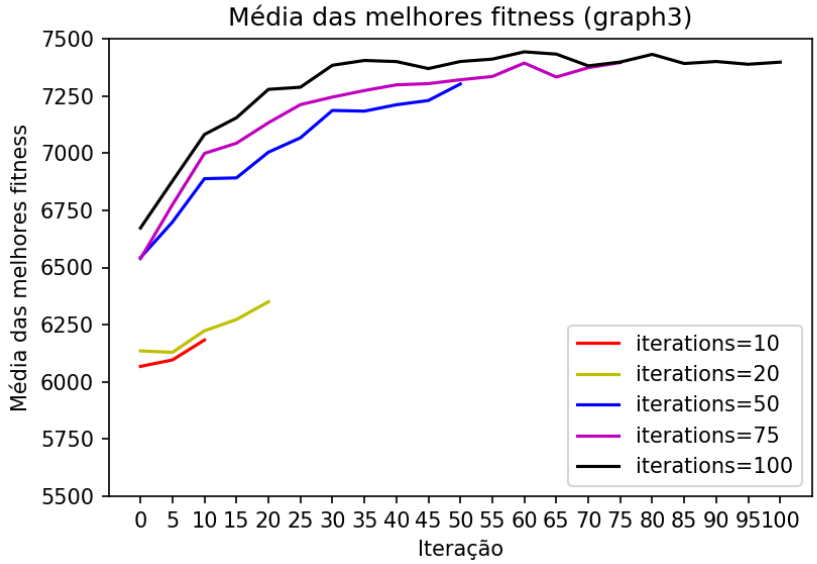


Figura 11 – Fitness do melhor caminho ao longo das iterações com a variação do número de iterações no *graph3*

Portanto, a variação desse parâmetro gera um comportamento na melhoria da solução que é facilmente observado. Desse modo, o número de iterações se mostrou um importante parâmetro a ser considerado.

## 3.4 K

O K variou no conjunto {1, 10, 20, 40}. Conforme o número aumentou, o tempo de execução do algoritmo ficou muito maior (inclusive do que aumentou o número de formigas), porém a solução encontrada no final não se modificou muito, apenas a convergência das soluções foi variada, conforme pode ser observado pelo gráfico abaixo:

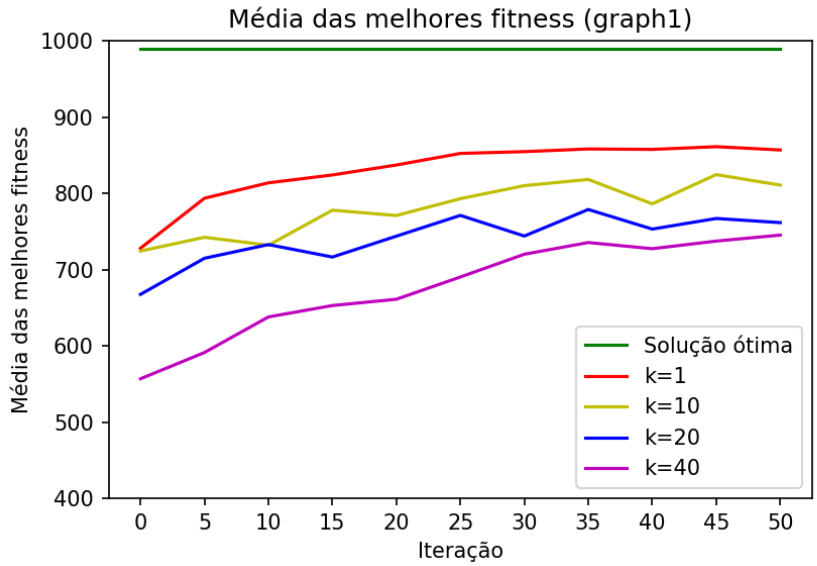


Figura 12 – Fitness do melhor caminho ao longo das iterações com a variação do $k$ no *graph1*

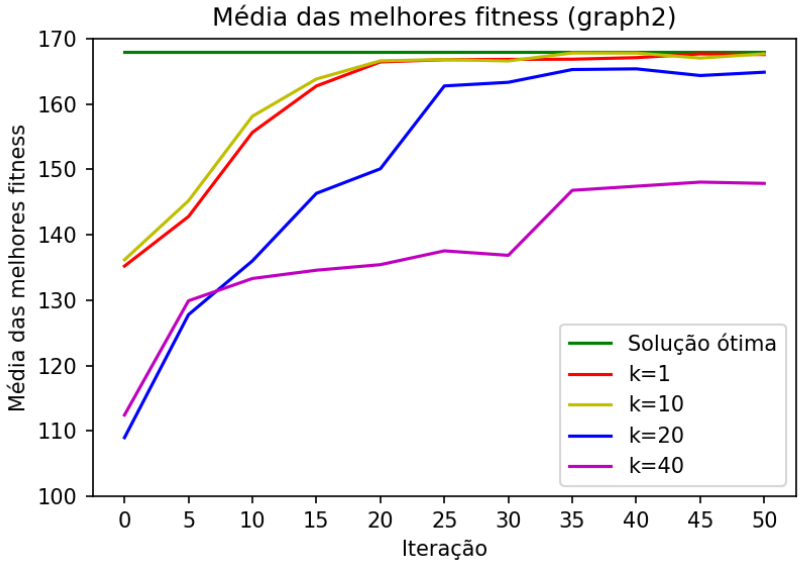


Figura 13 – Fitness do melhor caminho ao longo das iterações com a variação do $k$ no *graph2*

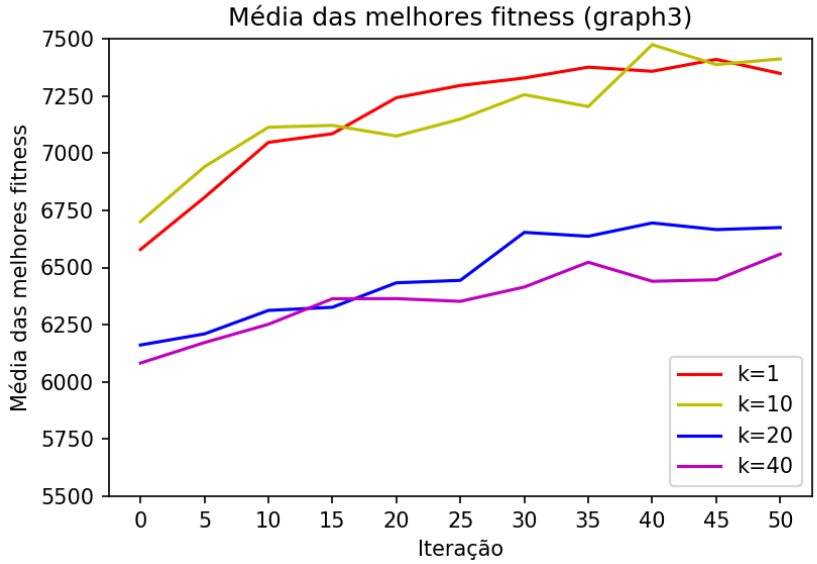


Figura 14 – Fitness do melhor caminho ao longo das iterações com a variação do $k$ no *graph3*

No entanto, essa melhoria também foi pequena, quando comparado aos outros parâmetros que foram variados.

### 3.5 Parâmetros Ótimos

Os parâmetros que se mostraram mais efetivos para a aproximação da solução ótima foram:

```
-a | --ants = 50
-e | --evaporation_rate = 0.1
-i | --iterations = 50
-k | --k_ants = 1
```

e, portanto, foram setados como parâmetros padrões na execução do programa principal. Desse modo, futuras execuções já poderão utilizar de parâmetros padrões otimizados anteriormente à partir dessas execuções experimentais anteriores.

## 4 Conclusões

Esse trabalho prático foi muito agregador no sentido de exercitar e procurar entender o quanto algumas variações de parâmetros impactam na solução final do algoritmo. Ele ajudou a fixar alguns conceitos de Otimização por Colônia de Formigas, bem como geração aleatória da população inicial, como representar uma solução, como alterar a velocidade de convergência das soluções, entre outros diversos tópicos importantíssimos para a área de Computação Natural e para algoritmos probabilísticos como um todo.

Para tal, a confecção de testes variando esses parâmetros se mostrou uma tarefa exaustiva e demorada (pela repetições e tempo de finalização da execução de alguns

testes), porém totalmente necessária para o entendimento do assunto proposto. No entanto, uma combinação exata desses parâmetros, de forma a prover uma solução ótima ou até mesmo aproximada à ótima, ainda se mostrou de uma complexidade enorme, mas foi possível determinar certos subconjuntos de combinações que melhoravam relativamente essas execuções em busca dessas soluções.

Embora em alguns casos a melhor solução alcançada tenha sido significativamente menor que a solução ótima (para os grafos *graph1* e *graph2*, aonde a solução ótima era conhecida), em outros foi possível encontrar caminhos que se aproximavam bastante ou eram exatamente iguais à solução ótima. Talvez o fato de que o poder computacional ao meu dispor para a realização desses testes é pequeno em relação ao tempo que era necessário para a execução completa do algoritmo me privou de encontrar soluções mais complexas, porém mais aproximadas do ótimo no caso citado anteriormente.

Para a resolução do problema de *Longest Path Problem*, a Otimização por Colônia de Formigas quase sempre tem um ótimo desempenho quando comparado à outros demais métodos, incluindo várias técnicas de aprendizagem de máquina. Isso demonstra o tamanho poder de algoritmos de Computação Natural para a resolução de problemas complexos em que são necessários diversos cálculos e estratégias distintas de se encontrar uma solução aproximada à ótima.

## 5 Referências Bibliográficas


[1] WIKIPEDIA. Randomized algorithm. https://en.wikipedia.org/wiki/Randomized_algorithm. Acessado em 03 de Setembro de 2018.

[2] WIKIPEDIA. Longest path problem. https://en.wikipedia.org/wiki/Longest_path_problem. Acessado em 03 de Novembro de 2018.

[3] WIKIPEDIA. Ant colony optimization algorithms. https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms. Acessado em 03 de Novembro de 2018.