UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Ciência da Computação
DCC191 -- Computação Natural
Trabalho Prático 2 - Inteligência Coletiva
Guilherme de Abreu Lima Buitrago Miranda - 2018054788

1 - Introdução

O trabalho prático 2 da disciplina de Computação Natural tem como objetivo implementar um algoritmo de colônia de formigas (ACO) para resolver o Job-Shop Scheduling (JSSP) - um problema de otimização combinatorial NP-Difícil. Dessa forma, o algoritmo deve considerar um conjunto de m máquinas, em que cada tarefa é composta por um conjunto de operações, cuja ordem de execução é pré-definida. O objetivo, portanto, é encontrar a sequência de operações que minimizem o tempo de execução de todas elas, seguindo restrições, como a não atribuição de uma tarefa para uma mesma máquina duas vezes. Também não se pode interromper uma operação, não há uma ordem de execução em relação a operações de tarefas diferentes e cada máquina pode processar apenas uma tarefa de cada vez.

Abaixo, encontram-se detalhes relativos à implementação do trabalho, experimentos, conclusões e referências bibliográficas.

2 - Implementação

Levando em consideração as dificuldades enfrentadas na realização dos experimentos para o TP1, optou-se, dessa vez, pela implementação do programa utilizando a linguagem Java. A expectativa, portanto, é que a execução dos testes seja mais rapidamente concluída, possibilitando um feedback mais veloz na análise experimental, em razão da natureza da linguagem em questão quando comparado ao Python.

Dessa forma, para executar o trabalho prático, é necessário, primeiro, compilar todos os arquivos .java com o comando “javac -d . *.java”. Em seguida, executa-se o Main: “java Main”.

O programa está dividido em quatro arquivos: Main.java, Aco.java, Pair.java e Pheromone.java. A implementação da classe pair é bem simples, apenas com construtores, gets, sets e um override do método equals. A classe usa tipos genéricos para o primeiro e segundo membro do par de forma que facilite sua reutilização no programa, tanto para armazenar pares inteiro-inteiro, quanto para ser usada ao retornar dois objetos de uma função, por exemplo. A implementação da classe Pheromone também é suficientemente simples, já que a mesma armazena apenas um par de pares, representando um vértice no grafo. Assim, a classe também conta apenas com construtor, gets e sets.

Assim, partindo para a classe Main, além da definição de algumas variáveis finais como o nome do arquivo, nome da instância e makespan ótimo, temos a leitura do arquivo propriamente dita. Após seu processamento, tem-se, portanto, o número de jobs, sua respectiva ordem e o tempo gasto por cada um.

Já, dentro da classe Aco, além da declaração dos parâmetros, encontram-se os métodos de maior interesse. A modelagem do problema foi implementada com base no artigo “An Ant Colony Optimization Algorithm for Job Shop Scheduling Problem”, conforme destacado na última seção deste documento. Assim, modela-se um grafo, representando o caminho que uma formiga segue. O primeiro passo é adicionar, neste grafo, dois nós “virtuais” para serem o início e o fim do caminho. Em seguida, cria-se um grafo disjunto, cujos nós representam uma tarefa, e tarefas de um mesmo job são conectadas por arestas direcionadas; por fim, tarefas rodando em uma mesma máquina são conectadas por arestas não direcionadas. As operações para construção de tal grafo estão representadas no início do método run e getJsspGraph da classe em questão.

Posteriormente, existem três métodos também utilizados antes de ocorrer a entrada no loop iterativo. São eles: getTimesSpent, getDesirability e pheromonesInitializer. Como os próprios nomes evidenciam: o primeiro decompõe os tempos de cada uma das tarefas; já o segundo calcula a desirability de cada um dos jobs nas máquinas; por fim, o terceiro inicializa o feromônio, guardando o custo das transições de todos os vértices para todos os outros.

Todos os métodos supracitados são chamados em sequência, sendo o de leitura de arquivo nas chamadas de testes ou execução, e os outros, a partir do método run. Neste, após as chamadas, dá-se início um loop iterativo, variando de 0 ao número máximo de iterações, recebido como parâmetro (e decidido de acordo

com os experimentos posteriormente citados). Dentro do loop, itera-se também sobre o número de formigas. Assim, para cada uma das formigas, é invocado, em primeiro lugar, o método generatePath. Nele, também há dois loops, sendo que o primeiro varia com o número de tarefas, enquanto o segundo percorre os nós do grafo a partir do inicial. Dentro do loop mais interno, obtém-se os novos valores de feromônio e de desirability para cada uma das tarefas. Em seguida, já fora do loop mais interno, calcula-se a probabilidade da formiga percorrer cada um dos caminhos e, utilizando o método Math.random, escolhe-se um nó para a visita. Ao fim do loop, tem-se o caminho completo percorrido pela formiga, que é retornado ao método principal run.

Subsequentemente, calcula-se o makespan do caminho encontrado. Para tal, tem-se o método calcCakespan, que, logo em seu início, remove todas as arestas bidirecionais não percorridas pela formiga durante a execução. Em seguida, tem-se, uma busca no grafo direcionado, calculando as distâncias entre cada um dos nós percorridos (representando o tempo gasto para a execução das tarefas). O tempo total é, finalmente, reportado. No run, apenas para report, duas conferências são feitas: a checagem se o caminho e makespan encontrados são os melhores, atualizando variáveis impressas após todas as execuções (se for o caso), além de checar se trata-se do makespan ótimo (fornecido no enunciado no trabalho). Caso verdadeiro, também é impressa uma mensagem de report, contendo a iteração que possibilitou tal encontro.

Por fim, ainda dentro do loop das iterações, o run chama o método updatePheromones que, conforme revela o próprio nome, atualiza os feromônios no grafo. Em primeiro lugar, cada uma das arestas tem seu feromônio atualizado a partir do valor de evaporação. Além disso, cada um dos caminhos reportados têm seus feromônios reforçados, sendo que o makespan obtido é um fator para maior ou menor reforço. Finalmente, o método principal da classe reporta o melhor caminho e o melhor makespan encontrados após o número de iterações desejados. Destaca-se, enfim, o não uso de elitismo no algoritmo, a fim de evitar grandes detenções a um máximo local.

Voltando para a classe Main, existem dois ifs desnecessários, usados apenas como forma de controle, para executar os testes ou o experimento final. Procedendo dessa forma, foi fácil preparar os testes (variando seus parâmetros) e coletar seus resultados. Tais procedimentos serão melhor explicitados na seção a seguir.

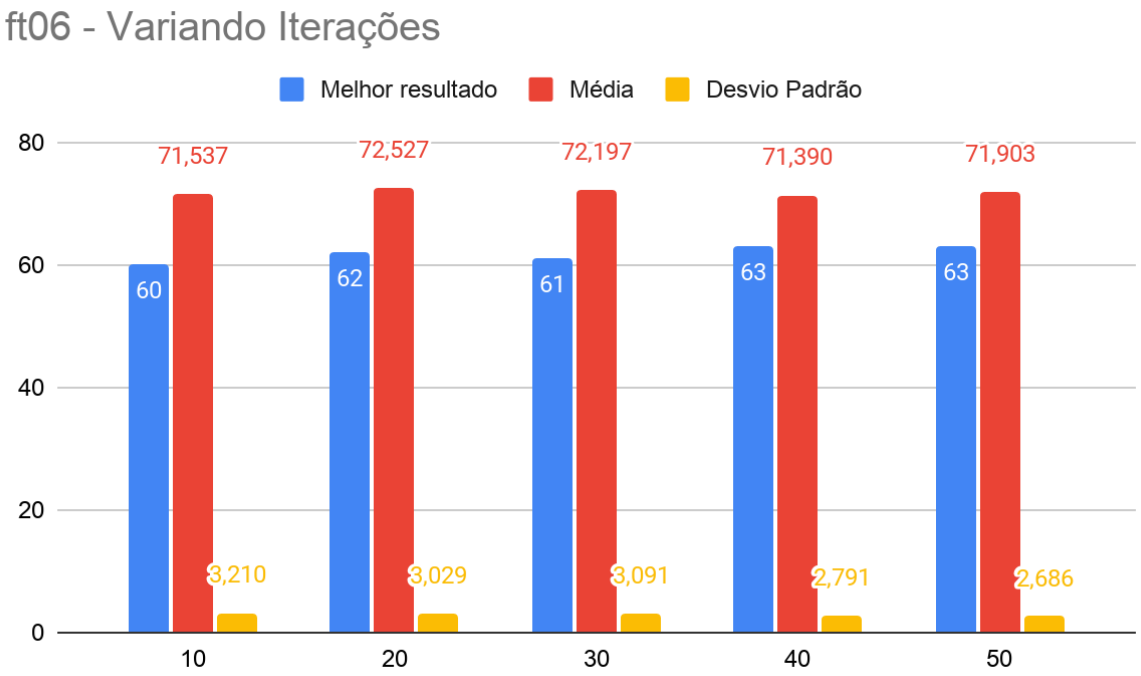
3 - Experimentos

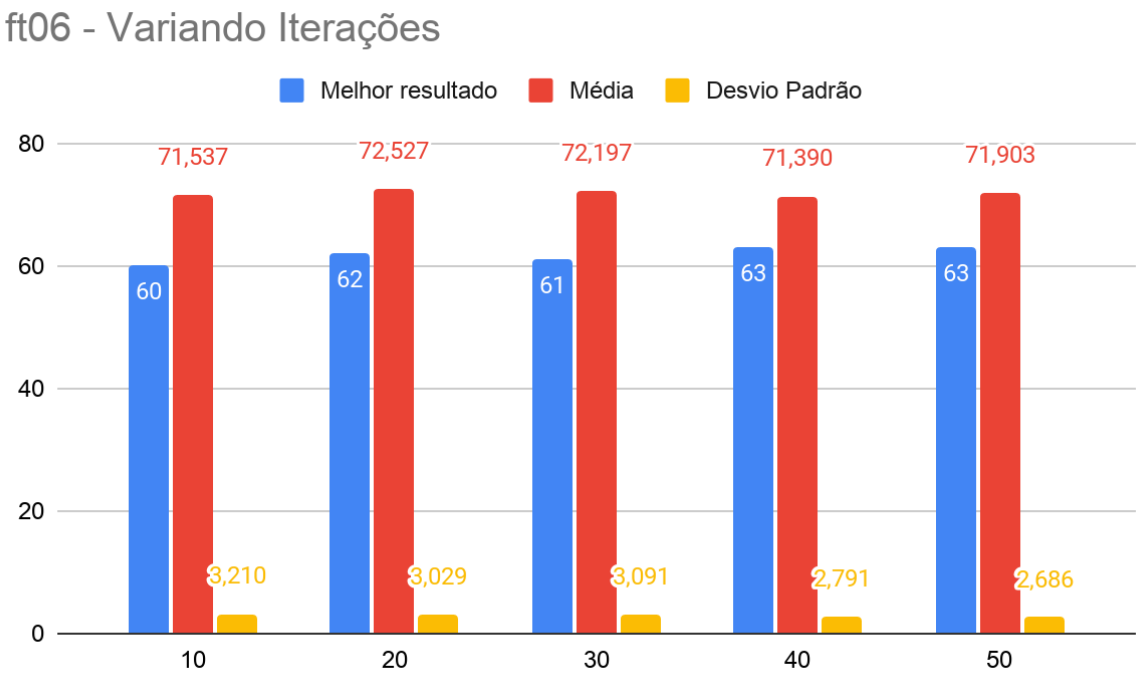
3.1 - Metodologia

Para a condução dos experimentos, em primeiro lugar, optou-se por escolher as instâncias ft06 e la01, por serem executadas mais rapidamente e, portanto, oferecerem um feedback ágil. Assim, fixaram-se parâmetros razoáveis e, em sequência, cada um deles foi variado.

Dessa forma, uma instância típica era executada com número de formigas igual a 50, máximo de iterações em 30, alpha e beta iguais a 1 e, por fim, evaporação em 0,2 e feromônio a 0.001. O primeiro parâmetro a ser variado foi composto pelas iterações. Após 30 execuções, os seguintes gráficos foram gerados:

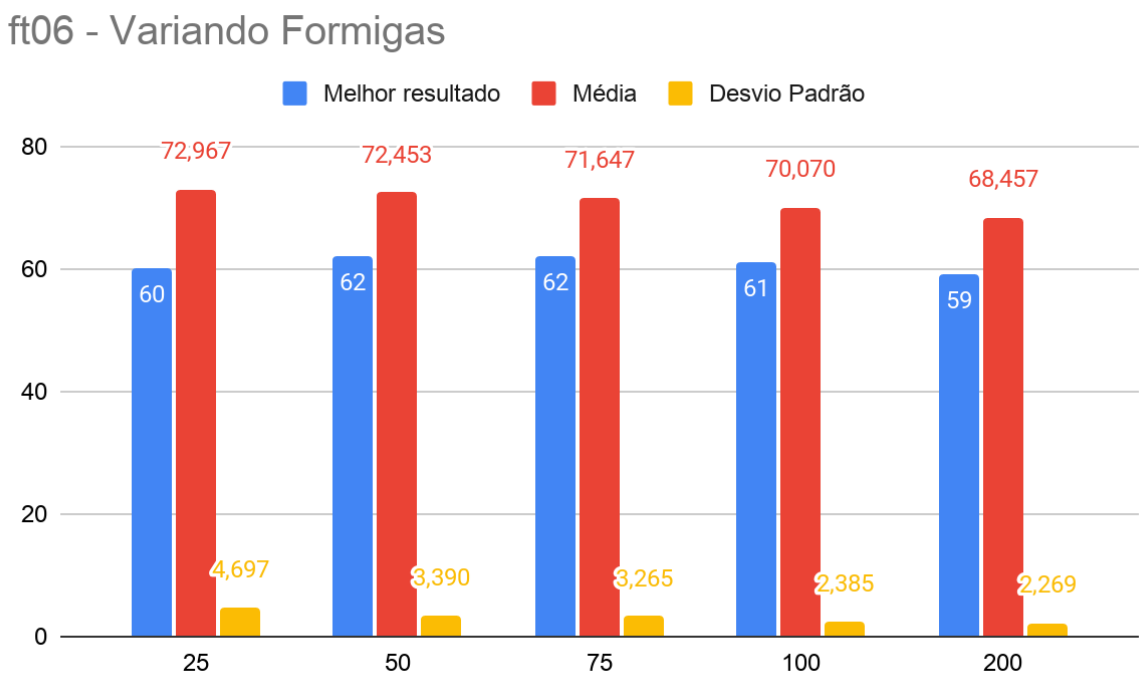
3.2 - Experimentos (próx página)

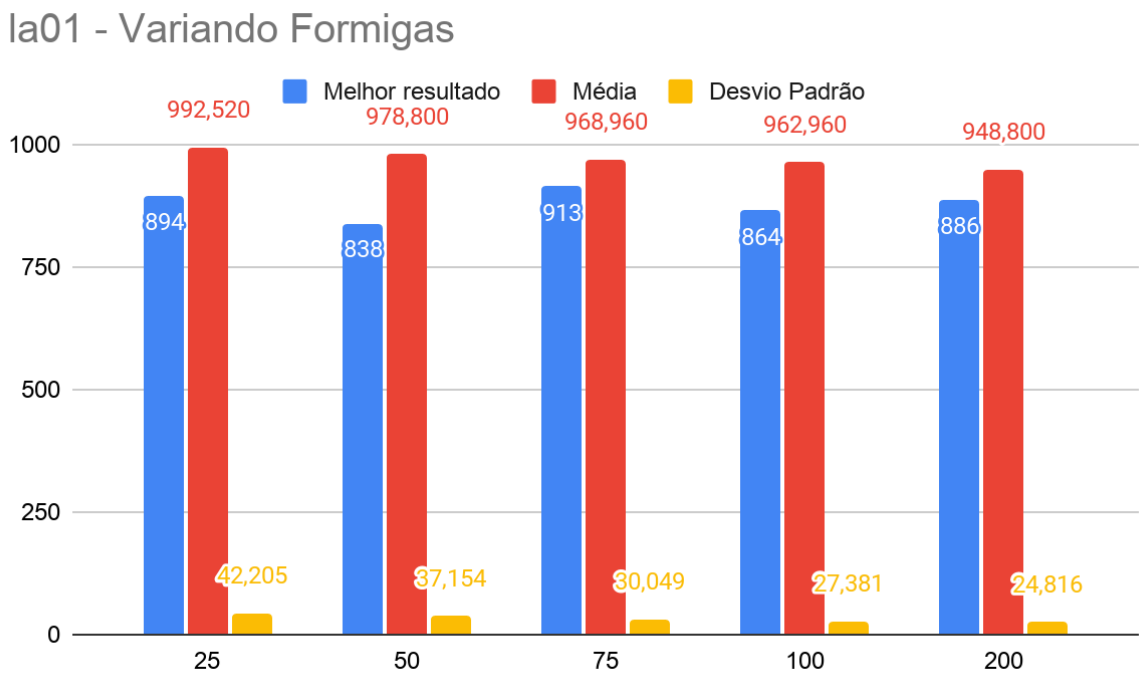




É bastante notável, portanto, que o número de iterações não é um parâmetro que afeta tão intensamente os resultados. Dessa forma, opta-se por manter a escolha previamente feita: 30.

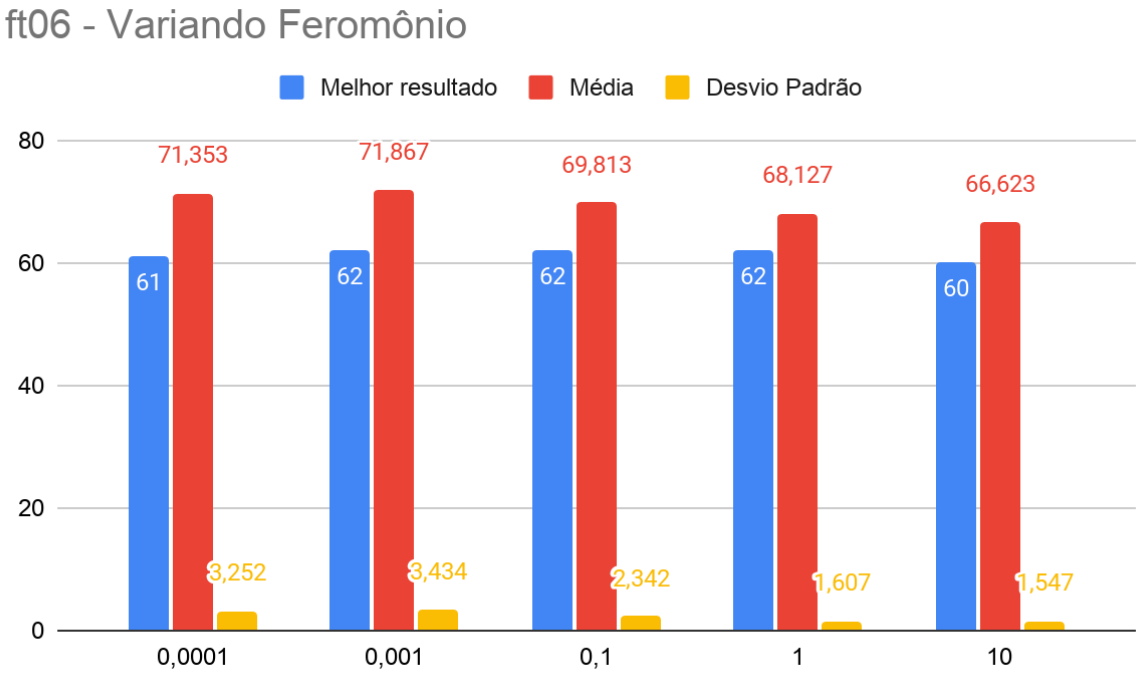
Prontamente, variou-se o número de formigas, obtendo os gráficos:

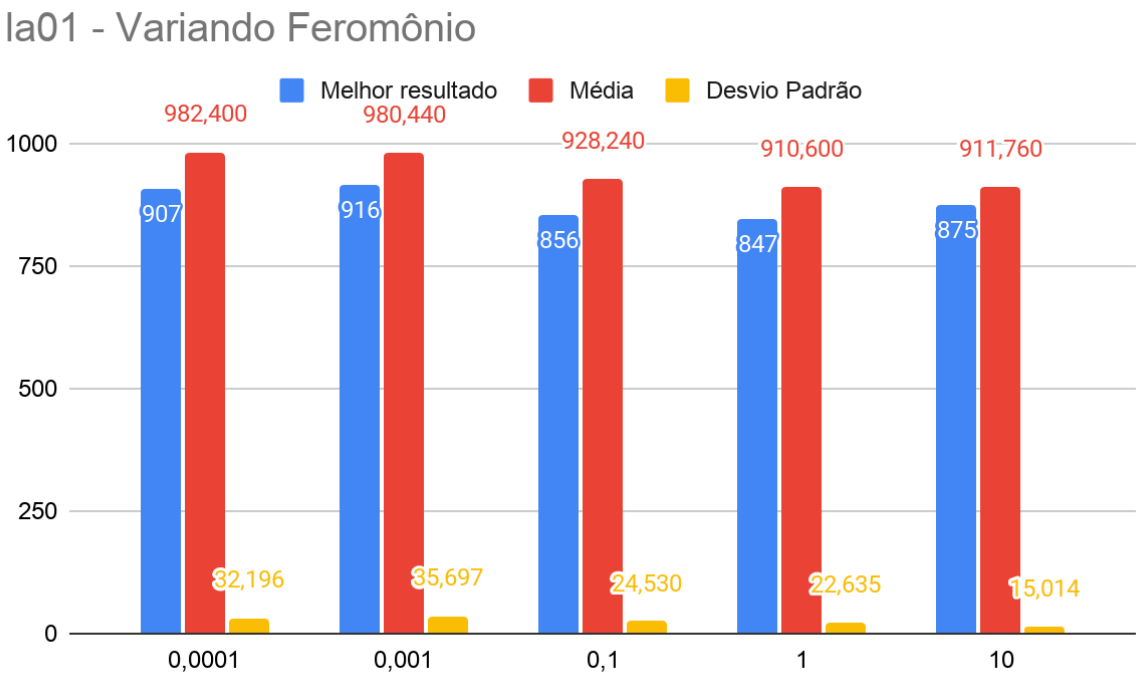




O número de formigas, em contrapartida, altera bastante os resultados encontrados - sobretudo a média. Assim, visando diminuir o makespan, define-se o número de formigas para os testes finais como 200.

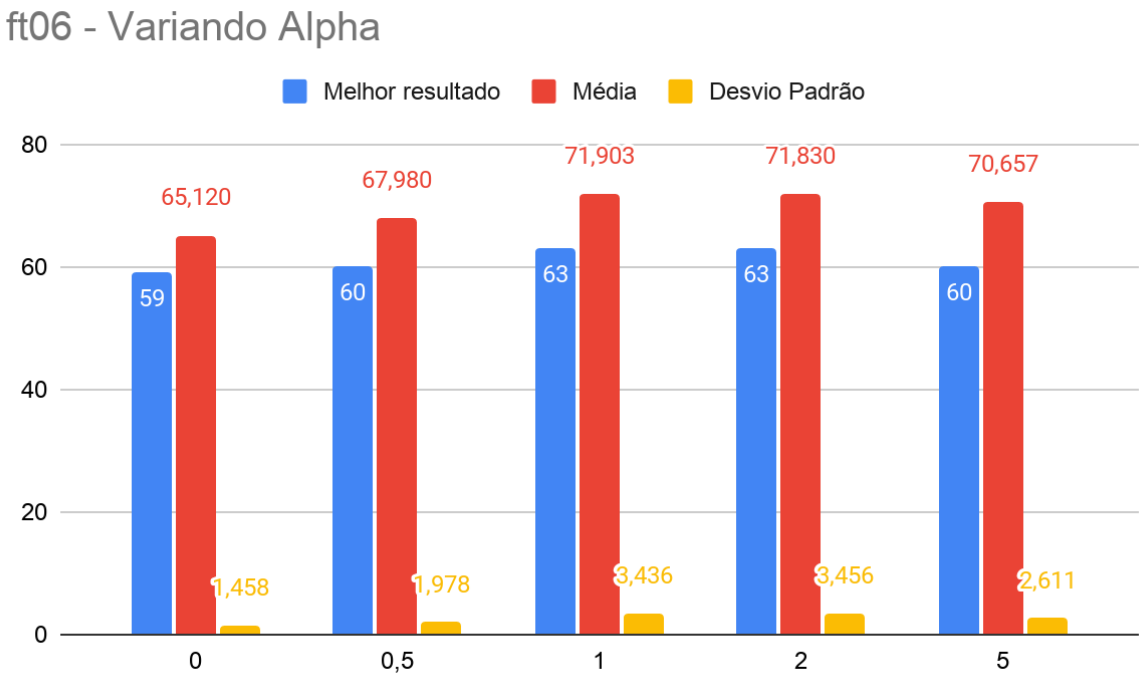
O feromônio, variado entre 0.0001, 0.001, 0.1, 1 e 10, gerou os seguintes gráficos:

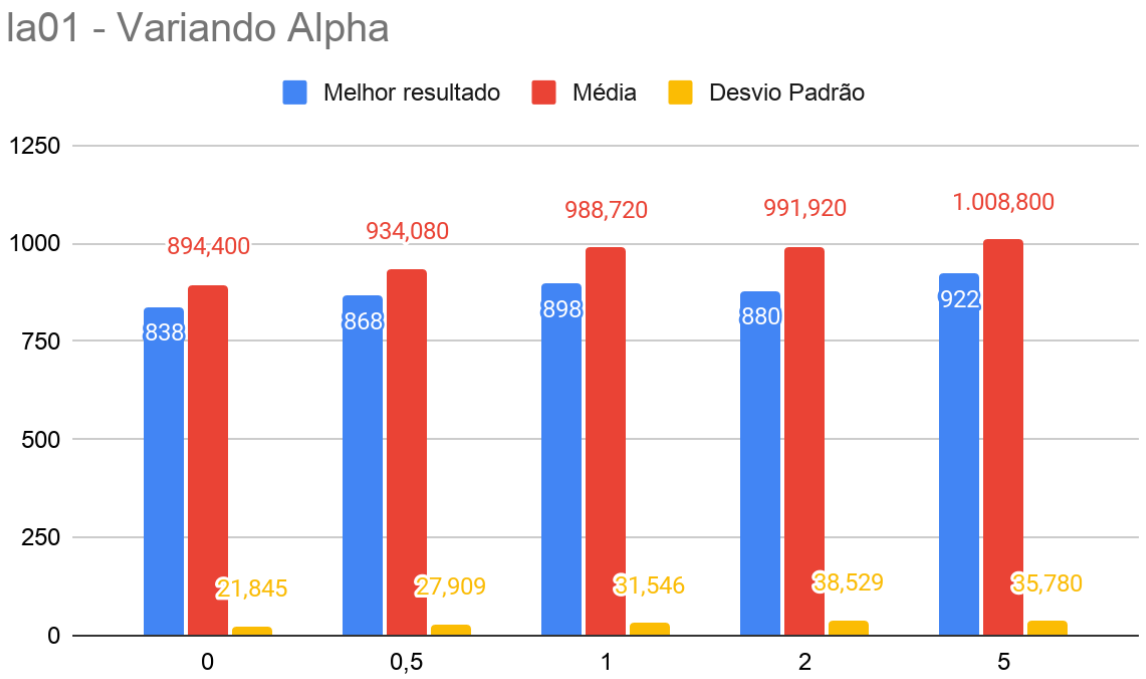




Como pode ser observado com base nos gráficos, quanto maior o valor atribuído, menor a média encontrada, o que justifica a escolha de 10 para os testes finais.

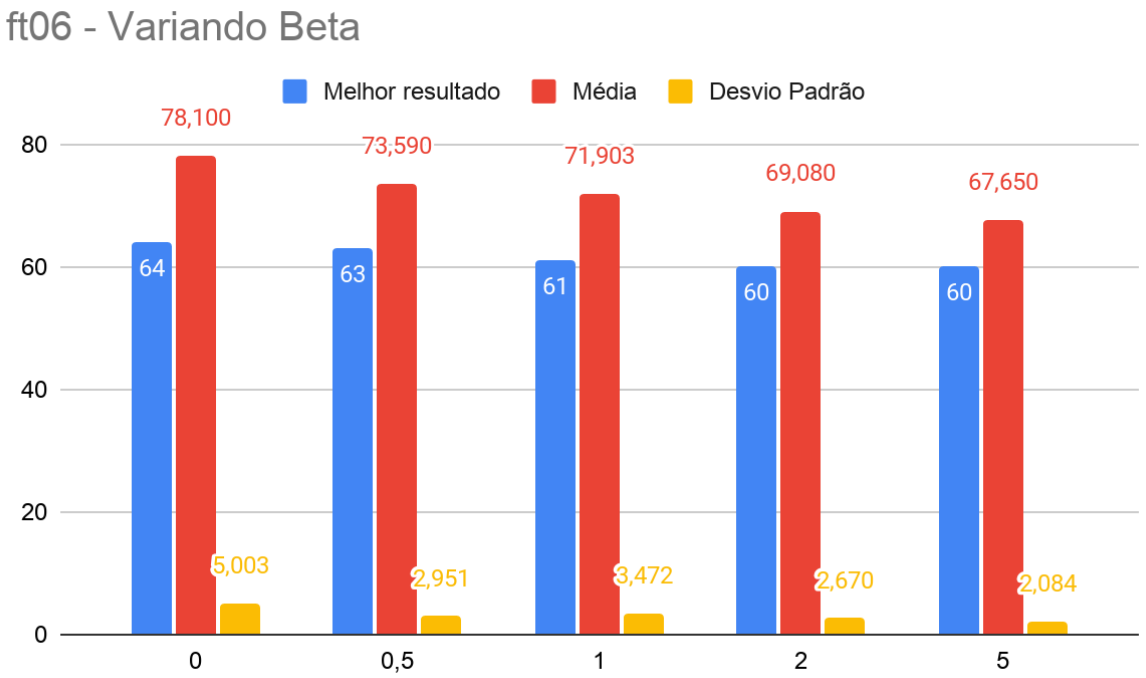
Para o valor alpha, tem-se:

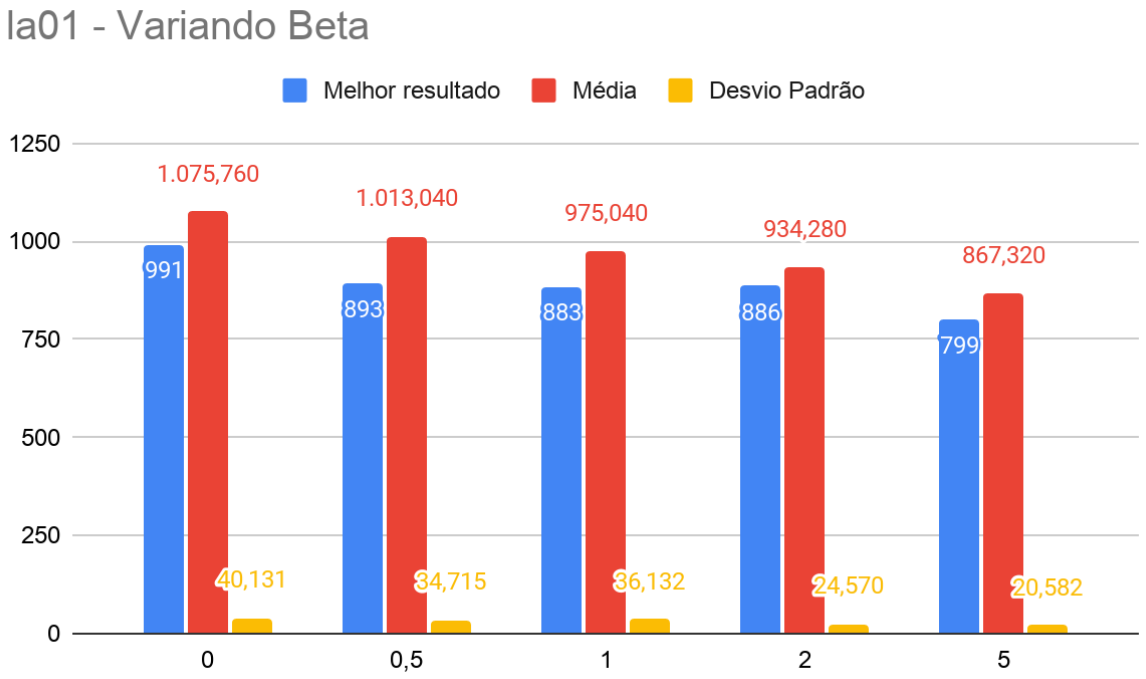




Ao contrário da variação do feromônio, pode-se observar que a média piora à medida que o valor de alpha é aumentado. Assim, o parâmetro alpha será zerado nos testes finais, levando em conta as quatro instâncias do problema.

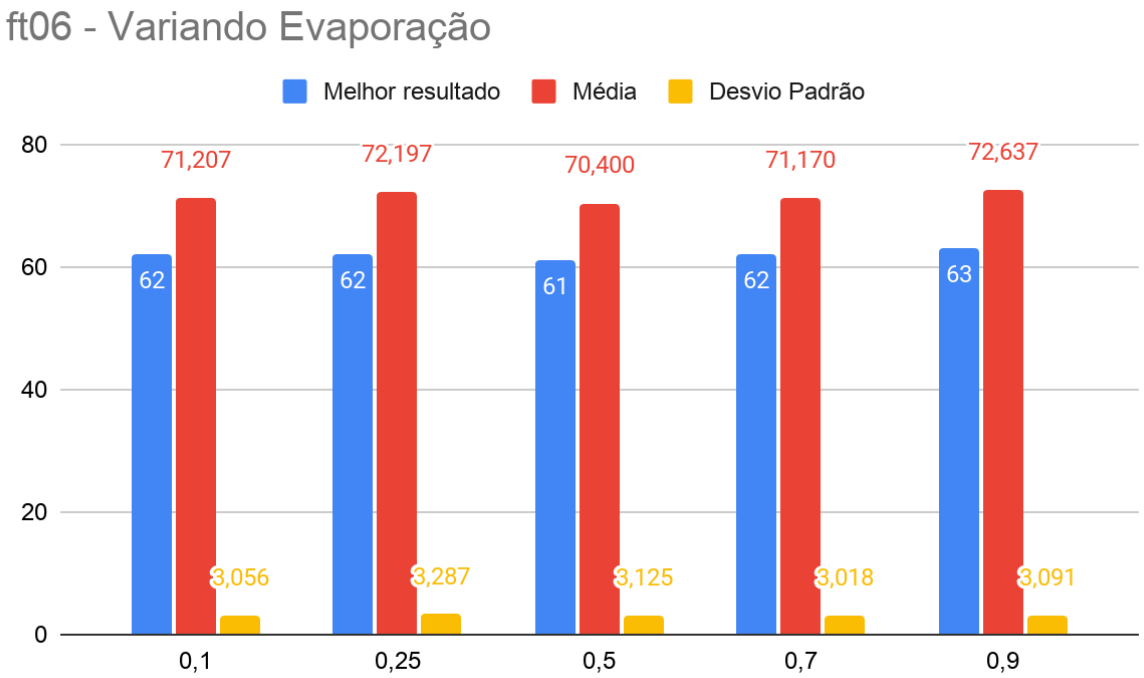
Por sua vez, para o valor de beta:

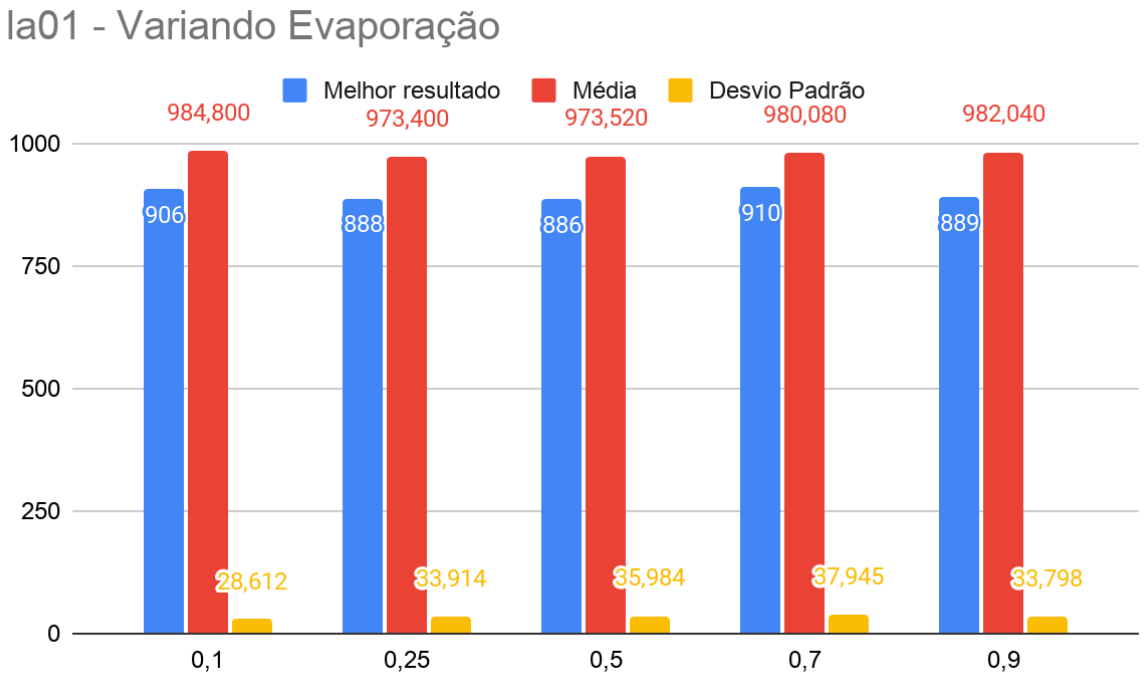




Em oposição ao alpha, o parâmetro beta melhora a média quando é aumentado. Diante disso, escolhe-se 5 como valor final.

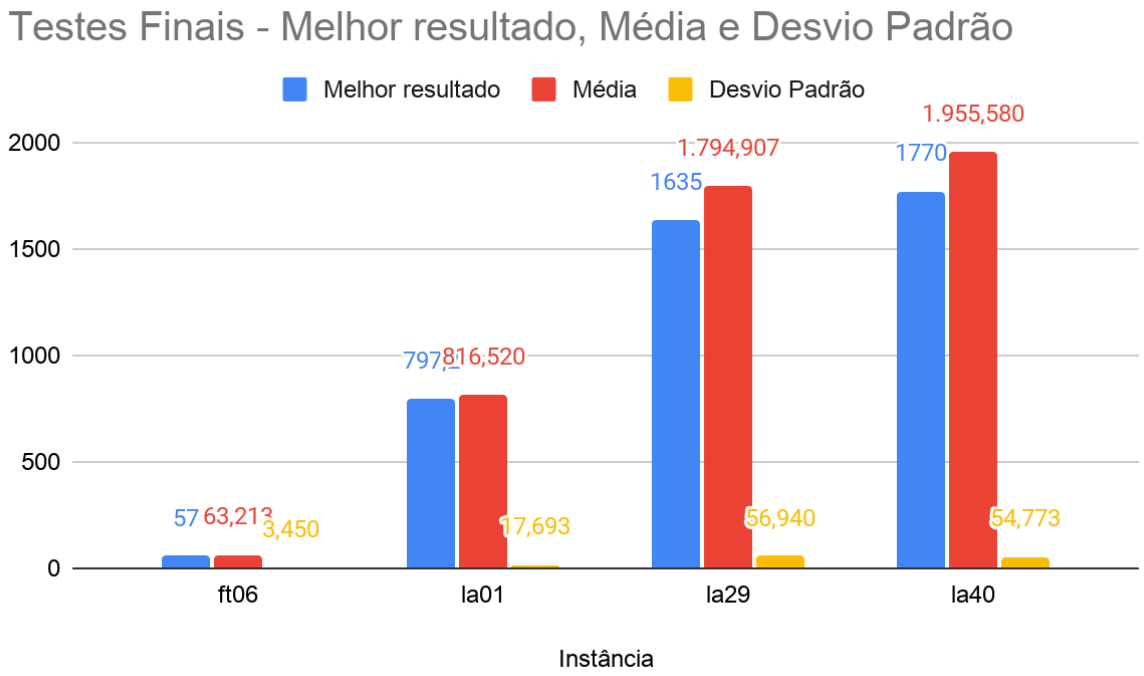
Por fim, considera-se o valor da evaporação:





Assim como o número de iterações, o valor da evaporação não parece alterar drasticamente os resultados encontrados. Logo, decide-se pelo valor de 0.5, intermediário e com as melhores médias.

Destarte, tais parâmetros foram utilizados para a execução das 4 instâncias do problema a serem consideradas: ft06, la01, la29 e la40. Obteve-se, assim, os seguintes resultados:



3.3 - Resultados

É possível notar como as médias do ft06 e do la01 melhoraram de maneira geral quando comparadas àquelas previamente observadas. As últimas duas instâncias, testadas pela primeira vez, obtiveram resultados entre razoáveis e bons. O la29 alcançou um mínimo de 1535 (contra um ótimo de 1157) e o la40 teve como makespan mínimo 1770, contra 1222 do ótimo. Tais diferenças representam, respectivamente, 32% e 44% de diferença, diante dos valores ótimos.

É possível que esses valores sejam melhorados realizando o mesmo procedimento descrito anteriormente (manter todos os parâmetros fixos, com exceção de um, que varia, a fim de observar qual a melhor combinação possível), mas com as bases de dados maiores. Além disso, crê-se que um maior número de formigas e de beta também poderiam auxiliar (os testes realizados utilizaram um valor máximo de 200 e 5, respectivamente). Contudo, a realização de novos testes demandaria ainda mais tempo (e tornaria essa documentação ainda mais extensa), o que julga-se não inteiramente adequado.

Dessa forma, após atingir bons resultados nas bases exaustivamente testadas e obter números apropriados para as bases restantes, considera-se que os experimentos obtiveram êxito em sua função.

4 - Conclusões

A implementação do trabalho foi bastante profícua, levando em consideração tanto a parte de escrita de código propriamente dita, quanto a análise experimental desempenhada. Além disso, o aprendizado se deu, sobretudo, no ato de transferir o conhecimento adquirido ao ler o artigo (citado na biografia abaixo) para o código, fazendo uma ponte com os conceitos vistos nas aulas. Em particular, embora a implementação na linguagem Java tenha sido feita visando ganhos de performance (que, felizmente, foram atingidos), em função da necessidade de declaração explícita de tipos e às comparações do HashMap feitas observando a posição de memória, alguns trechos do código podem ser desafiadores num olhar desatento e, portanto, necessitam de comentários para maiores explicações. Acredita-se que um possível equilíbrio entre simplicidade de código e performance poderia ser atingido usando uma terceira linguagem.

Não obstante, a análise experimental também foi de grande valor para refletir a respeito de detalhes do algoritmo de colônia de formigas, assim como para realizar uma análise crítica a respeito de quais valores utilizar em cada um dos parâmetros, com o intuito de obter valores cada vez melhores de makespan.

5 - Bibliografia

- An Ant Colony Optimization Algorithm for Job Shop Scheduling Problem - Edson Flórez, Wilfredo Gómez and MSc. Lola Bautista - https://arxiv.org/ftp/arxiv/papers/1309/1309.5110.pdf
- Generic pair class - StackOverflow - https://stackoverflow.com/questions/6044923/generic-pair-class
- Hashmap with Streams in Java 8 Streams to collect value of Map - StackOverflow - https://stackoverflow.com/questions/29625529/hashmap-with-streams-in-java-8-streams-to-collect-value-of-map
- Weighted randomness in Java - StackOverflow - https://stackoverflow.com/questions/6737283/weighted-randomness-in-java
- Standard deviation of an ArrayList - StackOverflow - https://stackoverflow.com/questions/37930631/standard-deviation-of-an-arraylist