GCC118 - Programação Matemática

Trabalho Prático Final

Bruno Crespo Ferreira

Departamento de Ciência da Computação – Universidade Federal de Lavras (UFLA) – Lavras – MG – Brazil

bruno.ferreira5@estudante.ufla.br

Resumo. *Este trabalho aborda a modelagem e resolução de um problema de otimização relacionado ao gerenciamento de pousos em um aeroporto. Foram utilizadas duas abordagens: programação matemática com o solver Gurobi e uma meta-heurística baseada na Busca Tabu. Os resultados computacionais foram analisados, comparando a qualidade das soluções e o tempo de execução de cada método. Os experimentos indicaram que o Gurobi obteve soluções ótimas globais, enquanto a Busca Tabu apresentou soluções próximas ao ótimo, mas com limitações devido à falta de diversificação.*

1. Introdução

A pesquisa operacional consiste no desenvolvimento de métodos científicos de sistemas complexos, com finalidade de prever e comparar estratégias ou decisões alternativas, pode-se considerá-la como uma abordagem científica para a tomada de decisões [1].

Um modelo matemático nada mais é do que um objeto abstrato, que procura imitar as principais características de um objeto real. Há diversos exemplos de modelos de programação matemática, como: programação linear (otimização linear), programação linear inteira (otimização discreta), entre outros.

Resolver um problema de otimização consiste em determinar uma solução ótima, isto é, determinar uma solução que satisfaça todas as restrições do problema e que atribua o melhor valor à função objetivo.

Otimização discreta, ou também programação inteira e combinatória, é uma formulação de um problema de otimização combinatória (OC) que consiste em achar o subconjunto de peso mínimo de F, isto é:

$$min\{\sum_{j \in S} c_j : S \in F\} \qquad \text{(I)}$$

Um problema OC pode ser formulado como uma problema inteiro (variáveis inteiras) ou problema binário (variáveis assumem 0 ou 1).

Problemas de programação inteira e combinatória são resolvidos por métodos ótimos (exatos), algoritmos aproximativos e métodos heurísticos.

Heurística e meta-heurísticas são algoritmos que buscam obter uma solução viável para um problema que forneça um bom limite primal com baixo custo computacional [2].

A partir de um problema e suas instâncias, foi formulado uma modelagem matemática inteira a fim de ser resolvida por um método exato, o qual será comparado com um método meta-heurístico específico. A essência é analisar e comparar os experimentos e resultados obtidos com o intuito de absorver o aprendizado em termos dos conteúdos de Programação Linear e Inteira.

Este documento está organizado como a seguir. A Seção 2 provê tanto o enunciado do problema quanto a sua formulação para a resolução em método exato, ademais uma alternativa de uma meta-heurística definida. A Seção 3 descreve como cada abordagem gera o arquivo com os resultados e como interpretá-los. A Seção 4 apresenta os resultados computacionais, com a intenção de comparar os métodos implementados e a Seção 5 conclui o trabalho.

2. Formulação

Esta seção introduz o problema a ser investigado e as formulações para cada método especificado, além de apresentar e definir a meta-heurística sorteada.

2.1. Enunciado do Problema

Um grande aeroporto brasileiro recebeu uma rodada de investimentos privados e pretende automatizar o processo de gerenciamento da aterrissagem de aeronaves em seu pátio principal. Diariamente, um conjunto de n aviões pousa nesse aeroporto. Cada avião i possui as seguintes informações.

- n: Quantidade de aviões;
- R_i: Tempo de detecção do avião i;
- E_i: Tempo inicial permitido para o pouso do avião i;
- T_i: Tempo ideal de pouso do avião i;
- L_i: Tempo final permitido para o pouso do avião i;
- g_i: Penalidade por unidade de tempo se o avião i pousar antes de T_i;
- h_i: Penalidade por unidade de tempo se o avião i pousar depois de T_i;
- s_{ij}: Tempo mínimo de separação entre o pouso do avião i e o pouso do avião j.

Considere uma matriz $S = [s_{ij}]$ (informada logo acima) com valores s_{ij}, $i, j = 1, ..., n$, representando o tempo de separação requerido após o pouso do avião i e antes do pouso do próximo avião j. A gerência de operações do aeroporto deseja determinar a sequência de aterrissagem dos aviões considerando apenas a pista principal de voo, bem como o tempo t_i em que cada avião deve aterrissar. Como critério de qualidade, pretende-se minimizar as penalidades em relação ao espaço de tempo em

que cada avião i pousou antes ou depois, respectivamente, que o seu tempo ideal para pouso.

2.2. Modelagem - Gurobi

O *solver* genérico escolhido para a resolução do problema descrito foi o Gurobi. A seguir está apresentado a modelagem matemática.

Os parâmetros/dados do problema já estão definidos na seção anterior, logo as variáveis de decisão definidas foram:

- t_i: Tempo de pouso do avião i;
- x_{ij}: Variável binária que assume 1 se o avião i pousar antes do avião j, 0 caso contrário;
- a_i: Tempo que o avião i pousa antes de T_i;
- d_i: Tempo que o avião i pousa depois de T_i.

A função objetivo consiste em minimizar a soma das penalidades associadas aos pousos antecipados e atrasados:

$$min \sum_{i=1}^{n} g_i a_i + h_i d_i \qquad \text{(II)}$$

As restrições se encontram abaixo:

➔ Restrição 1: O tempo de pouso do avião i deve estar dentro dos limites permitidos.

$$E_i \leq t_i \leq L_i, \quad \forall i \in \{1, \ldots, n\} \qquad \text{(III)}$$

➔ Restrição 2: Definição das variáveis a_i e d_i.

A lógica para o cálculo dessas variáveis é bem simples.

$$a_i \geq T_i - t_i, \quad \forall i \in \{1, \ldots, n\} \qquad \text{(IV)}$$

$$d_i \geq t_i - T_i, \quad \forall i \in \{1, \ldots, n\} \qquad \text{(V)}$$

A equação (IV) indica que $T_i \geq t_i$, logo o avião i pousou mais cedo que o esperado.

A equação (V) indica que $t_i \geq T_i$, logo o avião i pousou atrasado do esperado.

➔ Restrição 3: O sequenciamento dos pousos deve respeitar o tempo mínimo de separação entre os aviões.

$$t_j \geq t_i + s_{ij}, \quad \forall i \neq j, \quad se \; x_{ij} = 1 \qquad \text{(VI)}$$

$$t_i \geq t_j + s_{ij}, \quad \forall i \neq j, \quad se\ x_{ij} = 0 \qquad \text{(VII)}$$

A equação (VI) indica que o avião *i* pousou antes do avião *j*, isso quer dizer que o tempo de pouso de t_j deve, obrigatoriamente, pelo menos ser maior que a soma do tempo de pouso do avião i somado com o tempo mínimo de separação entre ambos aviões, nesse caso o valor s_{12} ou s_{21}.

De forma análoga, a equação (VII) indica que o avião *j* pousou antes do avião *i*, idem pensamento.

Percebe-se que as duas restrições são excludentes, pois os aviões *i* e *j* não devem pousar ao mesmo tempo. Portanto, tendo como referência ([1], pág. 169) foi designado um valor *M* grande para a restrição (VI), o pensamento algébrico se encontra a seguir.

$$t_j \geq t_i + s_{ij}$$

$$t_j - t_i \geq s_{ij}$$

$$t_j - t_i - s_{ij} \geq 0 \equiv -t_j + t_i + s_{ij} \leq 0$$

$$-t_j + t_i + s_{ij} \leq M(1 - x_{ij})$$

$$-t_j \leq -t_i - s_{ij} + M(1 - x_{ij})$$

$$t_j \geq t_i + s_{ij} - M(1 - x_{ij}) \qquad \text{(VIII)}$$

Tendo como referência a equação (VIII), se $x_{ij} = 1$, então a restrição é ativada, ou seja, é equivalente à equação (VII). Se $x_{ij} = 0$, por consequência a restrição é desativada e t_j pode assumir um valor suficientemente grande (M), o que garante que não será maior que t_i.

➔ Restrição 4: Como exposto anteriormente, dois aviões não podem pousar ao mesmo tempo.

$$x_{ij} + x_{ji} = 1, \quad \forall i \neq j \qquad \text{(IX)}$$

2.3. Introdução - Busca Tabu

A meta-heurística sorteada para este documento foi a Busca Tabu, o qual é um algoritmo determinístico que aceita soluções de piora para escapar de ótimos locais.

É uma meta-heurística *single solution-based*, ou seja, dedicada a melhorar uma única solução por vez. É um procedimento iterativo, onde a geração compreende em criar um conjunto de soluções candidatas a partir da solução atual e selecionar a melhor solução candidata para substituir a solução atual.

Além disso, utiliza o conceito de memória, assim sendo a Lista Tabu guarda soluções ou movimentos aplicados recentemente na busca, com isso permite proibir soluções/movimentos que levam a soluções já visitadas, exceto no caso em que um critério de aspiração permite uma solução/movimento tabu se ele melhora a melhor solução global. Certas implementações sugerem que os elementos da Lista Tabu permanecem nela por um determinado número de iterações, denominação chamada *tabu tenure*, não há um valor ótimo para ele (assim como outros parâmetros). O tamanho da Lista Tabu também pode ser curta, longa; estática, dinâmica ou adaptativa.

Há também o conceito de intensificação e diversificação. A primeira faz referência em utilizar uma memória de médio prazo que busca áreas promissoras do espaço de busca, assim permitindo revisitar boas soluções com algumas variações com o objetivo de melhorar a função objetivo. Já a diversificação, utiliza uma memória de longo prazo para forçar a busca em novas áreas do espaço de soluções, dessa maneira evita-se de ficar preso em uma única região de busca.

A explicação descrita pode ser resumida no pseudocódigo da meta-heurística pelo Algorithm 1.

Algorithm 1 Busca Tabu

Input: s // Solução inicial gerada
Output: Melhor solução encontrada

1: $Lista\_Tabu \leftarrow \emptyset$
2: **repeat**
3: $candidatos \leftarrow gerar\_vizinhos(s)$
4: // Solução não tabu ou com critério de aspiração satisfeito
5: Encontrar melhor solução vizinha s' viável
6: **if** $f(s') \geq f(s)$ **then**
7: Insere s na $Lista\_Tabu$
8: **end if**
9: $s \leftarrow s'$
10: Atualizar $Lista\_Tabu$ e $memorias$
11: **if** $criterio\_intensificacao$ **then**
12: Aplicar intensificação
13: **end if**
14: **if** $criterio\_diversificacao$ **then**
15: Aplicar diversificação
16: **end if**
17: **until** critério de parada satisfeito

2.4. Modelagem - Busca Tabu

Ao projetar a modelagem de uma heurística, ademais a Busca Tabu, deve-se ter as seguintes questões em mente:

- Como representar o problema?

- Como determinar a vizinhança?
- Qual é o movimento?
- Como selecionar o vizinho?
- Como representar os elementos da solução?
- Qual é a função objetivo e como calculá-la?
- Como definir a Lista Tabu?
- Qual é o critério de parada?
- Qual é o critério de intensificação e como aplicá-la?
- Qual é o critério de diversificação e como aplicá-la?

A representação proposta é uma lista ordenada da sequência de pouso do aviões. Por exemplo, a solução (3, 1, 2) indica que o avião 3 pousou primeiro, depois o avião 1, e por último o avião 2.

A vizinhança pode ser determinada realizando permutações simples (troca de dois aviões). Por exemplo, a solução (3, 1, 2) tem como vizinhança as candidatas (1, 3, 2), (3, 2, 1) e (2, 1, 3).

O movimento consiste na troca de dois aviões na sequência de pouso. Isso significa que temos $O(n^2)$ possibilidades por iteração.

Por definição, o vizinho selecionado é sempre aquele que é melhor, no caso do problema, seria o vizinho com menor penalidade.

Os elementos da solução são representados da seguinte forma:

★ Sequência de pouso;
★ Tempos de pouso;
★ Penalidade (valor da função objetivo).

O cálculo da função objetivo é o mesmo que na modelagem com o Gurobi, ou seja, a equação (II).

A seguir, foi definido os parâmetros da Busca Tabu:

- Lista Tabu
    - Armazena soluções, ao invés de movimentos;
    - Foi desconsiderado utilizar o critério de aspiração;
    - O tamanho de armazenamento é indefinido; porém
    - *tabu tenure* é calculado como:

$$tabu\_tenure = floor(\frac{n}{2}) \qquad \text{(X)}$$

- Critério de parada: Sem melhora por $max\_iter\_sem\_melhoria = 20$ iterações consecutivas.

Intensificação e diversificação serão discutidos posteriormente.

Seguindo o Algorithm 1, o primeiro passo é gerar uma solução inicial viável. Para tal, optou-se por implementar uma heurística construtiva gulosa baseada no seguinte critério: gera-se a solução inicial baseado na ordem crescente de tempo ideal de pouso. Isso quer dizer que a solução inicial partirá de uma solução perfeita, todavia na

maioria dos casos a “solução perfeita” não será a solução ótima, pelo fato de possuir uma grande chance de violar alguma restrição.

Dessa forma, constrói-se duas soluções iniciais viáveis. Na primeira assumimos que o avião na primeira posição da sequência de pouso irá pousar idealmente, logo assumimos também que o próximo avião também pousará no tempo ideal, se e somente se:

$$T_{atual} \geq t_{anterior} + s_{atual,\, anterior} \qquad \text{(XI)}$$

E caso contrário:

$$t_{atual} = t_{anterior} + s_{atual,\, anterior} \qquad \text{(XII)}$$

Ou seja, essa estratégia criará uma solução inicial viável onde os tempos que não podem ser ideais, serão atrasos de pouso.

De forma análoga, a segunda solução inicial viável que pode-se construir é considerar que o avião na última posição da sequência de pouso irá pousar idealmente, logo assumimos que o avião anterior pousará no tempo ideal, se e somente se:

$$T_{atual} \leq t_{prox} - s_{atual,\, prox} \qquad \text{(XIII)}$$

E caso contrário:

$$t_{atual} = t_{prox} - s_{atual,\, prox} \qquad \text{(XIV)}$$

Ou seja, essa estratégia criará uma solução inicial viável onde os tempos que não podem ser ideais, serão adiantamentos de pouso.

Na realidade, essa estratégia foi utilizada a todo momento quando o algoritmo calcula os tempos de pouso dos aviões. Assim sendo, obtém-se os vizinhos da solução inicial e seus tempos de pouso são calculados dessa forma apresentada. Logo, capta-se a melhor solução vizinha viável para aplicar uma intensificação na solução.

Embora o pseudocódigo da Busca Tabu afirme que para se aplicar a intensificação é necessário respeitar um critério, decidiu-se sempre intensificar a solução atual.

A ideia do algoritmo de intensificação é tentar reduzir as penalidades locais da solução atual sem alterar a ordem de pouso. Inicia-se na primeira posição, o avião é forçado a pousar no tempo ideal, então analisamos seu avião adjacente. Propositalmente, iremos obrigar que o avião adjacente assuma um valor de pouso baseado nas seguintes equações:

$$t_{prox} = t_{atual} + s_{atual,\, prox} \qquad \text{(XV)}$$

$$t_{anterior} = t_{atual} - s_{atual,\, anterior} \qquad \text{(XVI)}$$

Diferentemente das equações (XIII) e (XIV), assuma que a posição $atual$ é a posição do avião que tem seu pouso forçado em tempo ideal. A equação (XV) é o ajuste do próximo avião, enquanto (XVI) é o ajuste do avião anterior.

Com essa lógica, podemos criar três soluções novas: uma que considera somente o ajuste do próximo, outra que considera somente o ajuste do anterior, e a última que considera ambas.

Em cada solução nova criada, porém, deve-se verificar se o avião adjacente está no limite inicial/final viável e também lembrar de respeitar os valores da matriz de separação, com o intuito da solução manter uma sequência de tempos crescente.

O próximo passo seria verificar o critério de diversificação e aplicar uma estratégia para sair da busca local, porém essa implementação foi infrutífera pelo fato de como são criadas as soluções viáveis. Infelizmente, não foi possível pensar em uma estratégia que perturbasse significativamente a solução atual para a mesma ir para uma distante.

3. Descrição da Solução

Esta seção provê como interpretar os resultados obtidos das execuções. O arquivo README.md possui as instruções de execução e também explicação dos diretórios, dito isso não será reiterado essas funcionalidades neste documento.

3.1. Solução - Gurobi

Este trabalho usufruiu de uma licença acadêmica, a qual foi possível testar as instâncias mais pesadas. O Gurobi proporciona uma interface em linha de comando chamada *gurobi_cl* [3].

O código *gera_modelo.py* lê a instância desejada e cria o modelo matemático seguindo o padrão de escrita do Gurobi (formato .lp), o usuário pode escolher o nome do arquivo de saída e também qual instância quer analisar.

Executa-se o comando *gurobi_cl* sobre o modelo.lp, é possível salvar o log da impressão no terminal utilizando “LogFile=” e o resultado em si é guardado em “ResultFile=” que tem a extensão .sol.

Essa extensão .sol possui o valor da função objetivo e também o resultados de todas as variáveis de decisão, usa-se o código *imprimir_resultado.py* para analisar os resultados de maneira mais amigável. Igualmente, é possível executar o código *imprimir_log.py* que na realidade obtém informações da solução inicial e o tempo de execução da instância analisada.

3.2. Solução - Busca Tabu

Interpretar a solução da Busca Tabu é mais fácil, pois só é necessário executar o código *busca_tabu.py* passando como argumento o nome do arquivo de saída e o caminho da instância, ou até alterar mesmo alterar algum parâmetro (somente é possível *max_iter_sem_melhoria* e *tabu_tenure*).

O programa já irá apresentar os resultados em terminal ou em um arquivo caso o usuário prefira.

4. Resultados

Esta seção apresenta os resultados computacionais obtidos dos experimentos, envolvendo:

- Desvio percentual entre solução inicial (SI) e solução final (SF):

$$Desvio\ \%\ entre\ SI\ e\ SF\ = \frac{100\ *\ (SI - SF)}{SI} \qquad \text{(XVII)}$$

- Desvio percentual entre SF e solução ótima (SO):

$$Desvio\ \%\ entre\ SF\ e\ SO\ = \frac{100\ *\ (SF - SO)}{SF} \qquad \text{(XVIII)}$$

- Tempo computacional.

Tabela 1: Desvio percentual entre SI e SF - Gurobi

Instância	SI	SF	Desvio (%)
1	7710	700	90,92
2	17070	1480	91,33
3	34940	820	97,65
4	15590	2520	83,84
5	32180	3100	90,37
6	32180	24442	24,05
7	1150	1150	0
8	97045	1950	97,99

É notável que as soluções iniciais começam bastante longe, na maioria dos casos, até chegar à solução final. Entretanto, não há muita confiança nesses resultados, pois o modo como foi realizado a obtenção de SI não é garantida. Não foi possível encontrar uma fonte confiável que garantisse como obter a SI de um modelo Gurobi.

Tabela 2: Desvio percentual entre SI e SF - Busca Tabu

Instância	SI	SF	Desvio (%)
1	1210	1060	12,40

2	2030	2030	0
3	2870	1250	56,45
4	4480	4420	1,34
5	7120	4800	32,58
6	24442	24442	0
7	3974	3974	0
8	4190	2185	47,85

Percebe-se que na maioria dos casos, SI e SF são praticamente iguais e não se consegue obter solução fora do ótimo local. Isso se dá, como dito anteriormente, por causa da falta de uma diversificação, e também a medida para construção das soluções infelizmente não explora muitas ramificações.

Tabela 3: Desvio percentual entre SF e SO - Gurobi

Instância	SF	SO	Desvio (%)
1	700	700	0
2	1480	1480	0
3	820	820	0
4	2520	2520	0
5	3100	3100	0
6	24442	24442	0
7	1150	1150	0
8	1950	1950	0

É notável que a modelagem do Gurobi e o próprio *solver* foram perfeitos, todos os ótimos finais encontrados realmente eram ótimos globais.

Tabela 4: Desvio percentual entre SF e SO - Busca Tabu

Instância	SF	SO	Desvio (%)
1	1060	700	33,96
2	2030	1480	27,09

Percebe-se que não foi possível aproximar muito da solução ótima global, devido aos assuntos já retratados. Entretanto, os valores obtidos não são exatamente muito ruins.

Tabela 5: Tempo computacional (segundos)

Instância	Gurobi	Busca Tabu
1	0,03	0,02
2	0,18	0,06
3	0,08	0,15
4	2,69	0,15
5	10,70	0,14
6	0,00	0,08
7	0,10	0,37
8	0,51	3,08

É interessante observar que as instâncias 4 e 5 foram as mais custosas ao Gurobi e para a heurística elas foram rápidas, em contrapartida as instâncias 7 e 8 (principalmente) não foram eficazes na Busca Tabu.

5. Conclusões

O estudo comparou a programação matemática e a Busca Tabu na resolução do problema de otimização de pousos em um aeroporto. Os resultados mostraram que o Gurobi foi eficiente na obtenção de soluções ótimas globais, com um tempo computacional adequado.

A Busca Tabu, apesar de apresentar desempenho razoável, teve dificuldades para escapar de ótimos locais devido à ausência de uma estratégia eficaz de diversificação. Esse fator limitou sua capacidade de alcançar soluções mais próximas do ótimo global.

Em contrapartida, a heurística apresentou tempos de execução menores para algumas instâncias, demonstrando seu potencial para problemas maiores onde o uso de *solvers* exatos pode ser inviável.

Como trabalhos futuros, sugere-se aprimorar a Busca Tabu com técnicas de diversificação e explorar outras heurísticas ou híbridos entre métodos exatos e heurísticos para melhorar a qualidade das soluções encontradas.

6. Bibliografia

[1] ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H.; **Pesquisa Operacional**. Rio de Janeiro: Campus -Elsevier, 2007.

[2] SUMIKA, Fernanda. **Minicurso I: Introdução às Metaheurísticas**, 5-6 de abril de 2018. UFSJ. Slides de Aula.

[3] **How do I use the Gurobi Command Line Interface (gurobi_cl)?** Disponível em: <https://support.gurobi.com/hc/en-us/articles/13444017491857-How-do-I-use-the-Gurobi-Command-Line-Interface-gurobi-cl>. Acesso em: 6 fev. 2025.