Trabalho Computacional - Otimização em Redes Problema de Sequenciamento de Máquina Simples com Data de Entrega Comum

Ana Júlia de Lima Martins (2024659580) e Samuel Souza da Silva (2023733108)

***Resumo*— Este trabalho consiste no trabalho computacional da disciplina de otimização em redes do programa de pós-graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais. O projeto consiste em otimizar o problema conhecido na literatura como o problema de sequenciamento de máquina simples com data de entrega comum, do inglês *Single Machine Common Due Date* (SMCDD). Para isso, foi utilizado o algoritmo BVNS (*Basic Variable Neighborhood Search*) junto com a heurística matemática *fix-and-optimize* para busca local. Resultados mostram que o algoritmo empregado foi capaz de ter resultados melhores do que os valores de referência da função objetivo destacados no enunciado do trabalho.**

***Palavras chaves*— Trabalho computacional, SMCDD, BVNS, *fix-and-optimize*.**

I. Introdução

PROBLEMAS envolvendo o sequenciamento de tarefas vêm tendo significativa relevância no contexto atual, haja vista a crescente demanda de empresas que precisam proporcionar mercadorias aos clientes em um determinado prazo, bem como a adoção de princípios como o *just-in-time (JIT)* [1], [2]. Assim, de acordo com o JIT, a antecipação e o atraso são considerados prejudiciais e, devem, por consequência, ser minimizados [2]. Neste contexto, o problema de sequenciamento de máquina simples com data de entrega comum - *Single Machine Common Due Date* (SMCDD), consiste em um problema que está de acordo com tais princípios, sendo que ele visa a minimizar a penalidade imposta por atraso e por adiantamento de tarefas que possuem a mesma data de entrega e que devem ser executas por uma única máquina.

Dada a sua complexidade, há, na literatura, diversos trabalhos que visam a otimizar o problema SMCDD [3], [4], [5]. Especificamente em [3], por exemplo, foi proposto um algoritmo, denominado por Algoritmo Genético-Híbrido de Abelhas (*Hybrid Genetic-Bees Algorithm*) que pôde ser empregado no problema SMCDD. Resultados mostraram que o algoritmo proposto teve melhores resultados quando comparado com métodos convencionais, sendo, ao mesmo tempo, competitivo com o estado da arte.

Dado isso, o presente trabalho visa a utilizar algoritmos e ferramentas de otimização para o problema SMCDD. A abordagem utilizada consiste em empregar a meta-heurística BVNS (*Basic Variable Neighborhood Search*) e, no processo de busca local, utilizar a heurística matemática *fix-and-optmize*. Para a implementação do algoritmo *fix-and-optmize*, foi empregado o solver Gurobi [6] e a linguagem de programação usada foi a linguagem Python, por ser uma linguagem de alto nível. Ademais, valores de referências da função objetivo foram determinados no enunciado do trabalho e foi constatado que a abordagem proposta foi capaz de ter resultados satisfatórios.

O restante deste trabalho é dividido nas seguintes seções: a seção II apresenta uma breve descrição do problema SMCDD e a formulação matemática adotada; a seção III aborda a escolha dos algoritmos utilizados; por sua vez, a seção IV mostra a metodologia usada no desenvolvimento deste projeto; a seção V evidencia os resultados obtidos e, por fim, a seção VI apresenta as conclusões deste trabalho.

II. Apresentação do problema

A. Descrição do problema SMCDD

Dado um determinado conjunto de tarefas que possuem uma mesma data de entrega determinada, o problema de sequenciamento de máquina simples com data de entrega comum consiste em definir a sequência de execução dessas tarefas, executadas por uma mesma máquina, de forma a minimizar a soma das penalidades impostas por tempo de atraso ou de adiantamento na entrega dessas tarefas. Dado isso, a função objetivo do problema é dado pela equação 1:

$$min\ f_{obj} = \sum_{i=0}^{n} \alpha_i \cdot e_i + \beta_i \cdot t_i \tag{1}$$

Em que:

1) n é o número de tarefas;
2) α_i é a penalidade imposta por unidade de tempo (u.t.) em que a tarefa i é entregue em adiantamento;
3) $e_i = max(0, D_D - d_i)$ é o número de unidades de tempo em que a tarefa i é entregue em adiantamento;
4) β_i é a penalidade imposta por unidade de tempo em que a tarefa i é entregue em atraso;
5) $t_i = max(0, d_i - D_D)$ é o número de unidades de tempo em que a tarefa i é entregue em atraso;
6) d_i é a data de entrega da tarefa i, em unidades de tempo, ou seja, o tempo de conclusão da tarefa i;
7) D_D é a data de entrega comum das tarefas, em unidades de tempo.

B. Formulação matemática

Para a formulação matemática, é necessário definir as restrições do problema. Sendo assim, as restrições definidas nas equações 2 e 3 definem o tempo de atraso e de adiantamento da tarefa i, respectivamente.

$$t_i \geq d_i - D_d, \quad i \in N \tag{2}$$

$$e_i \geq D_d - d_i, \quad i \in N \tag{3}$$

Além disso, foi necessário determinar uma variável binária, que consiste em uma matriz $n \times n$, sendo n o número total de tarefas, de tal maneira que o elemento J_{ik} assume o valor 1 se a tarefa i for executada na ordem k ou zero caso contrário, conforme pode ser visto na equação 4.

$$J_{ik} = \begin{cases} 1, \text{se a tarefa } i \text{ foi executada na ordem } k \\ 0, \text{caso contrário} \end{cases} \tag{4}$$

Com isso, a equação 5 garante que apenas uma tarefa é executada para uma dada ordem k e a equação 6 garante que cada tarefa é executada uma vez.

$$\sum_{i=1}^{n} J_{ik} = 1, \quad k = 1, 2, \ldots, n \tag{5}$$

$$\sum_{k=1}^{n} J_{ik} = 1, \quad i = 1, 2, \ldots, n \tag{6}$$

Também foi necessário definir a variável τ_k que consiste no tempo de conclusão da tarefa de ordem k, sendo $k > 0$. Assim, a equação 7 fornece o tempo de processamento da tarefa de ordem 1 e a equação 8 o tempo de processamento da tarefa de ordem k, para k maior que 1, sendo p_i igual ao tempo de processamento da tarefa i.

$$\tau_1 = \sum_{i=1}^{n} J_{i1} \cdot p_i \tag{7}$$

$$\tau_k = \tau_{k-1} + \sum_{i=1}^{n} J_{ik} \cdot p_i, \quad k = 2, \ldots, n \tag{8}$$

A equação 9 define uma variável M que assume um valor maior do que a soma do tempo de processamento de todas as tarefas e, com isso, a restrição mostrada na equação 10 define o valor mínimo da data de entrega da tarefa i.

$$M = \sum_{i=1}^{n} p_i + 1 \tag{9}$$

$$d_i \geq \tau_k - M(1 - J_{ik}), \quad \forall\, i \in N, k \in N \tag{10}$$

Dando sequência, a equação 11 garante que o somatório do tempo de conclusão de todas as tarefas em ordem deve ser igual ao somatório do tempo de conclusão de todas as tarefas.

$$\sum_{k=1}^{n} \tau_k = \sum_{i=1}^{n} d_i \tag{11}$$

Por fim, as equações 12, 13, 14, 15 e 16 determinam o domínio das variáveis de entrada do sistema.

$$e_i \geq 0, \forall\, i \in N \tag{12}$$

$$t_i \geq 0, \forall\, i \in N \tag{13}$$

$$d_i \geq 0, \forall\, i \in N \tag{14}$$

$$\tau_k \geq 0, \forall\, k \in N \tag{15}$$

$$J_{ik} \in \{0, 1\}, \forall\, k \in N, i \in N \tag{16}$$

III. Algoritmos

A. Algoritmos utilizados

O algoritmo empregado neste trabalho consiste no BVNS (*Basic Variable Neighborhood Search*), uma vez que ele é uma meta-heurística simples de ser implementada e bastante utilizada na literatura. O BVNS é uma variante do VNS (Variable Neighborhood Search) e é um método que pode ser aplicado em problemas complexos de otimização [7]. Dado isso, o algoritmo 1 mostra o pseudocódigo do BVNS:

Algorithm 1 VNS Básico

Require: $t_{max} > 0$, $k_{max} > 0$
  function BVNS(x_0, k_{max}, t_{max})
    $t \leftarrow 0$
    $x \leftarrow x_0$ ▷ Inicializar solução
    while $t < t_{max}$ **do**
      $k \leftarrow 1$
      repeat
        $x' \leftarrow Shake(x, k)$
        $x'' \leftarrow LocalSearch(x', k)$
        $(x, k) \leftarrow NeighborhoodChange(x, x'', k)$
      until $k = k_{max}$
      $t \leftarrow CpuTime()$
    end while
    return x
  end function

Nota-se que o BVNS consiste em um algoritmo baseado em estrutura de vizinhança. Assim, partindo de uma condição inicial e tendo pré-determinadas estruturas de vizinhanças, o algoritmo realiza uma perturbação na entrada, x, e, em seguida, executa um método de busca local, para encontrar o melhor ponto na vizinhança de x. Com isso, baseado no resultado obtido por meio da busca local empregada, o método $NeighborhoodChange$ é executado para determinar qual será a próxima estrutura de vizinhança a ser utilizada e para determinar o valor de x da próxima iteração. Sendo assim, o método $NeighborhoodChange$ deve ser implementado seguindo o pseudocódigo 2. Ademais, o BVNS é interrompido quando se alcançou um tempo de processamento que é passado como parâmetro ao algoritmo.

Algorithm 2 *Neighborhood Change*

  function $NeighborhoodChange(x, x', k)$
    if $f(x') < f(x)$ **then**
      $x \leftarrow x'$
      $k \leftarrow 1$
    else
      $k \leftarrow k + 1$
    end if
    return x, k
  end function

As estruturas de vizinhanças utilizadas neste trabalho foram baseadas na referência [1] e, dessa maneira, três estruturas de vizinhanças foram adotadas, são elas:

- Aleatoriamente selecionar uma tarefa que está atrasada e colocar ela como a primeira tarefa de execução.
- Aleatoriamente selecionar uma tarefa que está atrasada e uma tarefa adianta e, com isso, trocar suas posições de execução.
- Aleatoriamente selecionar uma tarefa que está adiantada e colocar ela como sendo a última tarefa para executar.

Além disso, como requisito do trabalho, faz-se necessário utilizar uma heurística matemática para resolver o problema de otimização. A heurística matemática utilizada consiste na heurística *fix-and-optimize*, uma vez que ela é uma heurística de busca local e, portanto, pôde ser utilizada junto com o BVNS. O algoritmo *fix-and-optimize* iterativamente decompõe um problema de otimização em vários subproblemas mais simples que o problema original [8]. Em cada iteração, o algoritmo fixa parte das variáveis inteiras e otimiza o restante das variáveis que não foram fixadas. Com isso, cada subproblema pode ser otimizado de forma significativamente mais rápida do que o problema original. Assim, a figura 1 mostra um diagrama exemplificando o método. Nota-se que, a cada iteração do método, o subconjunto de variáveis inteiras livres é deslocado. Se a solução é melhorada após a resolução de um subproblema, então as variáveis são fixadas com novos valores obtidos e reinicia-se o processo de busca partindo do primeiro subproblema. Caso contrário, iremos para a próxima iteração até que todos os subproblemas sejam resolvidos.

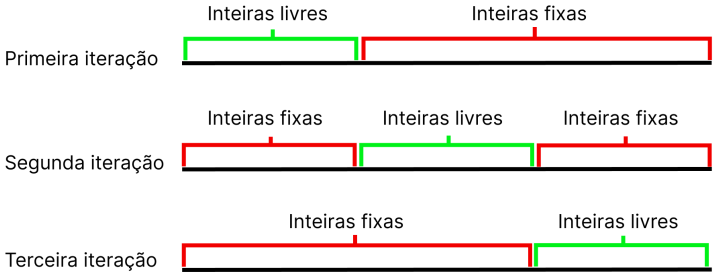

Fig. 1. Iterações do algoritmo *fix-and-optimize*.

É válido destacar que, na primeira iteração do método *fix-and-optmize* implementado, 30% das variáveis de entrada foram variáveis inteiras livres e os 70% restantes foram variáveis inteiras fixas. Na segunda interação, houve 35% de variáveis inteiras livres e esse subconjunto de variáveis foi deslocado, porém houve uma sobreposição de 5% de variáveis inteiras livres da iteração anterior. Por fim, na última iteração, houve 45% de variáveis inteiras livres com uma sobreposição de 5% de variáveis inteiras livres da iteração anterior, no qual as demais 55% de variáveis foram fixadas. Essa adaptação foi feita de forma empírica, sendo que tal sobreposição de variáveis foi capaz de trazer melhores resultados.

B. Solução inicial

A solução inicial do problema foi determinada utilizando um método semelhante ao feito em [1]. No qual as tarefas do problema foram ordenadas em ordem decrescente de β_i/α_i. Assim, tarefas que tenham alta penalidade por tempo de atraso ou baixa penalidade por adiantamento tendem a ser as primeiras a serem executadas.

IV. Metodologia

Para implementar os algoritmos propostos, escolhemos a linguagem de programação Python, haja vista que ela é uma linguagem de alto nível que possui diversas bibliotecas relativas à otimização. Além disso, para a implementação da heurística matemática, o solver Gurobi [6] foi utilizado, em sua implementação em python. Para isso, foi necessário emitir licenças acadêmicas para a utilização do Gurobi, sendo elas ofertadas aos alunos da Universidade Federal de Minas Gerais.

Duas instâncias do problema foram empregadas para realizar testes e obter resultados, sendo elas dadas pelos arquivos sch100k1.csv (contendo 100 tarefas) e sch200k1.csv (contendo 200 tarefas). Tais arquivos de entrada foram disponibilizados junto com o enunciado do trabalho. Cada linha do arquivo possui três valores separados por virgulas, que são: o tempo de processamento da tarefa i (p_i), a penalidade imposta por unidade de tempo em que a tarefa i é entregue em adiantamento (α_i) e a penalidade imposta por unidade de tempo em que a tarefa i é entregue em atraso (β_i).

Para facilitar o desenvolvimento do trabalho e o compartilhamento de código entre os integrantes do grupo, a plataforma de hospedagem de códigos-fonte GitHub foi adotada, sendo que o código fonte deste trabalho pode ser encontrado na seguinte URL:

- https://github.com/ssdasilva/Trabalho-Computacional-Otimizacao-em-Redes

Para executar os algoritmos implementados, basta executar o script “main.py”, localizado dentro do diretório “src” do repositório mencionado. Um menu “help” pode ser visualizado passando o argumento “--help” a este script python. Tal script pode receber os seguintes argumentos:

- --license: corresponde ao caminho da licença gurobi a ser usada, sendo o valor padrão igual a “$HOME/Downloads/gurobi.lic”.
- --file: caminho do arquivo .csv de entrada, sendo o padrão igual ao caminho do arquivo “sch100k1.csv” localizado no diretório “data” do repositório.
- --due_date: tempo de entrega comum das tarefas (padrão: 454).
- --output: caminho do arquivo de saída (padrão: diretório corrente)
- --verbose: habilita modo verboso (valor padrão: “True”).
- --max_iterations: máximo número de iterações para o algoritmo BVNS. Sendo assim, se o algoritmo BVNS tiver um número de iterações maior que o valor passado como argumento, então o algoritmo é interrompido, retornando a melhor solução encontrada (valor padrão: 30).
- --max_time: tempo máximo, em segundos, para cada iteração do algoritmo de busca local *fix-and-optmize*. Se o algoritmo ultrapassar o valor passado, então ele é interrompido, retornando a melhor solução encontrada (padrão: 15).

Foram feitos testes utilizando diferentes parâmetros para os argumentos “max_iterations” e “max_time”. Os melhores resultados foram obtidos ao executar o script “main.py” da seguinte forma:

- Para o arquivo “sch100k1.csv”, podemos usar os valores padrão dos argumentos “max_iterations” e “max_time”:

```
cd <ROOT_REPOSITORY_FOLDER>

python src/main.py --license <PATH>
--due_date 454 --file data/sch100k1.csv
```

- Para o arquivo “sch200k1.csv”:

```
cd <ROOT_REPOSITORY_FOLDER>

python src/main.py --license <PATH>
--due_date 851 --file data/sch200k1.csv
--max_iterations 50 --max_time 160
```

V. Resultados

Como foi escolhido um algoritmo não exato, a execução do algoritmo proposto foi feita cinco vezes, conforme um dos requisitos do trabalho. Sendo assim, a tabela I destaca os resultados obtidos tendo por base o arquivo sch100k1.csv, de 100 tarefas.

TABELA I

Resultados obtidos para o arquivo de entrada sch100k1.csv. Em negrito destaca-se a melhor execução obtida.

Execução	Função Objetivo	Referência	CSV
1	88.461	89.588	Link
2	88.604	89.588	Link
3	88.525	89.588	Link
4	88.818	89.588	Link
5	**87.288**	**89.588**	**Link**

A tabela II, por sua vez, apresenta os resultados obtidos tendo por base o arquivo sch200k1.csv, de 200 tarefas.

TABELA II

Resultados obtidos para o arquivo de entrada sch200k1.csv. Em negrito destaca-se a melhor execução obtida.

Execução	Função Objetivo	Referência	CSV
1	300.729	301.449	Link
2	304.205	301.449	Link
3	307.842	301.449	Link
4	**298.655**	**301.449**	**Link**
5	300.446	301.449	Link

Nota-se, portanto, que em ambos os casos, com 100 e com 200 tarefas, foi possível obter resultados melhores do que o valor de referência.

VI. Conclusão

Pode-se concluir que obtivemos resultados satisfatórios para o problema de sequenciamento de máquinas simples com data de entrega comum, por meio do algoritmo BVNS junto com a heurística matemática *fix-and-optimize* para busca local. Sendo assim, considerando o arquivo de entrada de 100 tarefas, o algoritmo proposto obteve como melhor solução o valor da função objetivo igual a 87.288, melhor que o valor de referência dado por 89.588. Além disso, considerando o arquivo de entrada de 200 tarefas, a melhor solução encontrada teve o valor de 298.655, que, por sua vez, é melhor que o valor de referência de 301.449. Dessa maneira, conclui-se, portanto, que os requisitos destacados no enunciado do trabalho foram atingidos.

Referências

[1] Natália Antunes, *Estudo de heurísticas matemáticas para o problema de escalonamento de máquina simples*, Dissertação de Mestrado, Universidade Federal de Minas Gerais, Brasil, December 2019. Disponível em: http://hdl.handle.net/1843/31901

[2] Débora P. Ronconi and Márcio S. Kawamura, “The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm,” *Computational & Applied Mathematics*, vol. 29, no. 2, pp. 193–208, Brazilian Society for Computational and Applied Mathematics (SBMAC), June 2010. Disponível em: http://dx.doi.org/10.1590/S1807-03022010000200002

[3] B. Yuce, F. Fruggiero, M. S. Packianather, D. T. Pham, E. Mastrocinque, A. Lambiase, and M. Fera, “Hybrid Genetic Bees Algorithm applied to single machine scheduling with earliness and tardiness penalties,” *Computers & Industrial Engineering*, vol. 113, pp. 842–858, Elsevier BV, November 2017. Disponível em: http://dx.doi.org/10.1016/j.cie.2017.07.018

[4] Wei Weng and Shigeru Fujimura. ”Self evolution algorithm for common due date scheduling problem.” In *2008 IEEE International Conference on Automation Science and Engineering*, Aug. 2008. Publisher: IEEE. DOI: 10.1109/coase.2008.4626419. URL: http://dx.doi.org/10.1109/COASE.2008.4626419.

[5] Ching-Jong Liao and Che-Ching Cheng. ”A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date.” *Computers & Industrial Engineering*, volume 52, number 4, pp. 404–413, May 2007. Publisher: Elsevier BV. ISSN: 0360-8352. DOI: 10.1016/j.cie.2007.01.004. URL: http://dx.doi.org/10.1016/j.cie.2007.01.004.

[6] Gurobi Optimization, *The Leader in Decision Intelligence Technology - Gurobi Optimization*, https://www.gurobi.com/, (Accessed on 06/11/2024).

[7] Variable Neighborhood Search: 7th International Conference, ICVNS 2019, Rabat, Morocco, October 3–5, 2019, Revised Selected Papers, Lecture Notes in Computer Science, Springer International Publishing, 2020, ISBN: 9783030449322, ISSN: 1611-3349, http://dx.doi.org/10.1007/978-3-030-44932-2, DOI: 10.1007/978-3-030-44932-2.

[8] Dorneles, Árton P., de Araújo, Olinto C.B., e Buriol, Luciana S. *A fix-and-optimize heuristic for the high school timetabling problem*. Computers Operations Research, vol. 52, pp. 29–38, Elsevier BV, 2014. Disponível em: http://dx.doi.org/10.1016/j.cor.2014.06.023.