UNICAMP - Instituto de Computação

MO420 - Programação Linear Inteira

Trabalho Final

Explorando Abordagens para o Trigger Arc Traveling Salesperson Problem

Aluno: Vitor Antônio Pimenta Silva

Professor: Prof. Dr. Flávio Keidi Miyazawa

Resumo

Neste trabalho foram abordadas diferentes técnicas para produzir limitantes superiores e inferiores para o *Trigger Arc Traveling Salesperson Problem* (TATSP). Nesta variante do TSP o objetivo é minimizar os custos de um ciclo hamiltoniano porém com a adição de efeitos dinâmicos sobre as arestas. Por se tratar de um TSP com restrições adicionais, é um problema NP-Difícil e, portanto, abordagens alternativas serão exploradas para contornar a complexidade do modelo. Foram elaboradas soluções para o programa linear inteiro (ILP) baseando-se em um modelo matemático de um trabalho prévio e, ao todo, foram desenvolvidas: uma abordagem de solução direta do ILP, utilizando o Gurobi; uma solução via Relaxação Lagrangiana (RL); e uma metaheurística *Simulated Annealing* baseada na relaxação linear e em fixação de variáveis via operadores.

Julho, 2025

1 Descrição do Problema

No trabalho de Cerrone et al. [1] o TATSP foi descrito como sendo uma variante do TSP original, sendo descrito, basicamente, como um problema para encontrar um ciclo Hamiltoniano de custo mínimo em um grafo direcionado onde existem relacionamentos entre arcos. Esses relacionamentos são responsáveis por dinamizar os custos de arestas no grafo. De agora em diante referenciamos estes relacionamentos utilizando as expressões de "gatilho"ou "*trigger*"para arcos (ou arestas) que ativam alteração de custo para arestas "alvo"ou "*target*". O modelo matemático proposto por Cerrone et al. [1] servirá como base de referência para nosso modelo de programação linear inteira mista.

No trabalho de Cerrone et al. [1] é fornecido um exemplo explicativo para representar o impacto da existência de relacionamentos entre as arestas do grafo. Além disso, na competição *Metaheuristics Competition Race at MESS 2024* [2] também foram apresentados exemplos de casos de uso reais onde o TATSP é aplicável, ilustrando um depósito de estantes móveis, onde os arcos simulam as movimentações das estantes e os relacionamentos exploram como determinadas movimentações podem afetar os custos de movimentação de estantes no futuro.

Formalmente, em um grafo direcionado (e não necessariamente completo) $G = (N, A)$, onde N é o conjunto de vértices ou nós e A é o conjunto de arcos ou arestas direcionados. O objetivo é determinar um ciclo Hamiltoniano de custo mínimo considerando a existência de relacionamentos entre as arestas definidos como $r = (t, a)$, significando que o custo da aresta a será alterado caso a aresta t seja escolhida para o caminho e esteja **antes** de a na ordem de visitação.

Seja $c(a) \in R^+ \forall a \in A$ o custo de passar pelo arco a. Para cada arco $a = (h, k)$ existe um conjunto de relacionamentos $R_a = \{(a_1, a) | a_1 \in A\}$ representando todos os relacionamentos que afetam o arco a. Além disso, define-se $c(r) \in R^+ \forall r \in R_a$ como os novos custos para o arco a caso a relação r esteja **ativa**. Além disso, definimos R como sendo o conjunto de todos os relacionamentos (para todos os arcos) da instância. Considerando essas entradas,

utilizamos, integralmente, o modelo matemático de Cerrone et al. [1], descrito a seguir.

São quatro conjuntos de variáveis de decisão binárias e um conjunto de variáveis inteiras auxiliar, sendo definidos como:

- x_a: binárias com valor 1 quando o arco a está presente no caminho, e 0 caso contrário. Estas variáveis indicam quais arcos estarão no caminho construído.
- y_a: binárias com valor 1 quando não há relacionamento ativo em R_a e $x_a = 1$, e 0 caso contrário.
- y_r: binárias com valor 1 quando o relacionamento r está ativo, e 0 caso contrário.
- $\hat{y}_r$: binárias com valor 1 quando, para uma relação $r = (a_1, a)$, o arco a precede o arco a_1 no caminho, e 0 caso contrário.
- u: inteiras utilizadas como auxiliares para definir a ordem de visitação de cada nó do grafo. Esta variável é importante para as restrições que evitam a existência de subciclos na solução.

Assim, a formulação completa segue:

$$\text{(TATSP)} \quad \min Z = \sum_{r \in R} c(r) y_r + \sum_{a \in A} c(a) y_a \tag{1}$$

sujeito a

$$\sum_{(i,j) \in A} x_{ij} = |N| \tag{2}$$

$$u_i + 1 \leq u_j + M(1 - x_{ij}) \qquad \forall (i,j) \in A / j \neq 0 \tag{3}$$

$$\sum_{(i,j) \in A} x_{ij} = 1 \qquad \forall j \in N \tag{4}$$

$$\sum_{(i,j) \in A} x_{ij} = 1 \qquad \forall i \in N \tag{5}$$

$$y_a + \sum_{r \in R_a} y_r = x_{ij} \qquad \forall a = (i,j) \in A \tag{6}$$

$$y_r \leq x_{ij} \qquad \forall r = ((i,j),(h,k)) \in R \tag{7}$$

$$u_i + 1 \leq u_h + M(1 - y_r) \qquad \forall r = ((i,j),(h,k)) \in R \qquad (8)$$

$$u_h + 1 \leq u_i + M(1 - \hat{y}_r) \qquad \forall r = ((i,j),(h,k)) \in R \qquad (9)$$

$$x_{ij} \leq (1 - x_{hk}) + (1 - y_{hk}) + \hat{y}_r \qquad \forall r = ((i,j),(h,k)) \in R \qquad (10)$$

$$u_{\hat{i}} - M(\hat{y}_{r2}) \leq u_i + M(2 - y_{r1} - x_{\hat{i}\hat{j}}) - 1 \qquad \forall r1 = ((i,j),(h,k)) \in R, \quad \forall r2 = ((\hat{i},\hat{j}),(h,k)) \in R \qquad (11)$$

$$x_{ij}, y_a \in \{0,1\} \qquad \forall a = (i,j) \in A \qquad (12)$$

$$y_r, \hat{y}_r \in \{0,1\} \qquad \forall r \in R \qquad (13)$$

$$u_i \in \{0, |N|\} \qquad \forall i \in N \qquad (14)$$

Começando pela Equação (1), temos o objetivo de minimizar o custo do caminho. Cada um dos termos indica se o custo de cada arco do caminho será o custo base (original) ou o custo modificado por um relacionamento. A Equação (2), referida no código como restrição $C1$, é responsável por garantir que o número de arcos do caminho seja igual à quantidade de nós da entrada, sendo uma condição mínima para o ciclo Hamiltoniano. A Equação (3) controla a posição dos vértices, servindo de auxílio para conferir a ordem de visitação do caminho, além de garantir que não existam subciclos – aqui vale destacar que o *Big M* adotado para todas as restrições é o número de vértices, $M = |N|$. Já as Equações (4)–(5) garantem que cada nó tenha apenas um arco de entrada e outro de saída, além de também tornarem a Equação (2) redundante e desnecessária ao modelo. As restrições da Equação (6) são responsáveis por garantir que somente um custo pode ser associado a um arco (seja o modificado, o base ou zero). Em seguida, a Equação (7) produz restrições que requerem que a relação $r = ((i,j),(h,k))$ esteja ativa se o caminho passar pelo arco gatilho (i,j). A Equação (8) garantem o controle de precedência entre os gatilhos e alvos, ou seja, se o arco gatilho (i,j) não precede o alvo (h,k), então $y_r = 0$. Já a Equação (9) visa garantir que, se o arco (h,k) não precede (i,j) na solução, então $\hat{y}_r = 0$. A restrição descrita pela Equação (10) trabalha com diferentes condições para cada $r = ((i,j),(h,k)) \in R$, onde o arco gatilho (i,j) só pode ser usado se o arco (h,k) não é usado; ou se a variável y_{hk} associada ao custo base do arco (h,k) não está ativa; ou se ambos

os arcos (i, j) e (h, k) são usados na solução, (i, j) aparece após (h, k). E, por fim, o conjunto de restrições da Equação (11) define uma relação de precedência entre os relacionamentos, garantindo que um arco alvo (h, k) tenha seu custo alterado pelo arco gatilho mais recente (mas ainda anterior a ele). As restrições dadas pelas Equações (12)–(14) indicam os tipos das variáveis utilizadas.

2 Métodos

Nesta seção trataremos sobre as abordagens utilizadas no desenvolvimento do trabalho. Foram feitas implementações para três opções de ataque ao TATSP: Programação Linear Inteira (ILP), Relaxação Lagrangiana (RL) e uma metaheurística *Simulated Annealing* baseada na relaxação linear (SA).

2.1 Programação Linear Inteira – ILP

A primeira abordagem a ser codificada foi a de programação linear inteira, utilizando o modelo matemático base [1] e resolvendo o problema com o *solver* comercial Gurobi. A ideia principal foi de tentar alcançar soluções exatas para as instâncias do problema. Matematicamente, o modelo é suficiente para produzir soluções ótimas, porém, a realidade da execução é que a medida em que as instâncias crescem, a complexidade computacional associada ao problema também cresce, principalmente em problemas de natureza combinatória [3].

Utilizando a linguagem de programação Julia e os pacotes JuMP, Gurobi, dentre outros utilitários, foi codificada uma solução baseada no modelo matemático TATSP, porém adicionando uma restrição para garantir que a primeira posição do caminho fosse ocupada pelo vértice 1 (1-indexado) que, em termos práticos, seria o vértice que representa o ”depósito”. No código foi chamada de ”C0”e é da forma $u[1] = 1$.

2.2 Relaxação Lagrangiana – RL

Percebendo a dificuldade de convergir para uma solução, a segunda abordagem utiliza a Relaxação Lagrangiana como base para produzir limitantes superiores e inferiores. A ideia por trás disso é produzir limitantes de maneira iterativa para tentar reduzir o *gap* aproximando-se ao valor ótimo. O limitante inferior, chamado de *lower bound*, vem de uma solução produzida após dualizar restrições consideradas problemáticas (geralmente restrições que estendem

o modelo e o tornam mais complexo, por exemplo). Basicamente essas restrições passam a ser termos de penalização na função objetivo e seus pesos são dados por multiplicadores de Lagrange. Em seguida, com essa solução (não necessariamente viável), pode-se utilizar diferentes técnicas para produzir um limitante superior (*upper bound*), que indica uma solução viável para o problema original mas que não necessariamente é ótima.

Para o TATSP, foi codificada uma versão que se aproveita do ILP e usa alguns artifícios para modularizar a dualização de restrições. A ideia por trás disso foi ser capaz de explorar quais restrições poderiam beneficiar a relaxação e também a viabilização para a construção de um *upper bound*. A implementação foi feita baseada no material de aula e no livro de Bertsimas e Tsitsiklis [4].

Para o TATSP, dualizar as restrições $C6 - C10$ (Equações (7)–(11)), reduz o problema, praticamente, a um TSP. Mesmo que ainda seja um problema NP-Difícil, para chamadas iterativas ao *solver*, isso já representa uma redução de complexidade. Além disso, ao dualizar as restrições relacionadas somente aos gatilhos, pode-se garantir que o *Gurobi* irá produzir caminhos viáveis, sendo assim, para construir um *upper bound* basta, após a otimização, recomputar os custos reais dos arcos a partir do caminho feito. Porém, é importante reconhecer que para instâncias maiores, resolver o TSP por si só já é muito desafiador.

A cada iteração é computada a violação de cada restrição dualizada e são atualizados os multiplicadores de Lagrange para ela. Além disso, a cada iteração é recalculado o passo utilizando o valor da função objetivo com o melhor *upper bound* encontrado até o momento (*Polyak*). Isso é importante para ajudar na convergência do método do subgradiente.

No capítulo de experimentos são mostrados os resultados de exploração das diferentes dualizações.

2.3 Metaheurística Simulated Annealing – SA

Além da relaxação Lagrangiana, foi desenvolvida uma heurística baseada na técnica de Simulated Annealing (SA) aplicada sobre a relaxação linear (LP) do modelo exato do TATSP. Essa abordagem visa explorar o espaço de soluções fracionárias da LP para obter boas soluções

inteiras viáveis, construindo assim limites superiores de qualidade para o problema.

A ideia central consiste em relaxar a integridade das variáveis binárias do modelo original (ILP), permitindo que assumam valores contínuos entre 0 e 1. A solução da LP contêm valores fracionários para as variáveis de decisão x_a e y_r, os quais serão utilzados como "guias probabilísticos"para a construção de soluções inteiras (*upper bounds*). No entanto, essas soluções fracionárias não necessariamente satisfazem, todas as restrições do TATSP, especialmente aquelas relacionadas à ativação condicional dos arcos. Para isso, empregamos uma estratégia de viabilização baseada em fixações progressivas das variáveis de ativação y_r.

A ideia de utilizar o SA partiu de uma curiosidade de exploração e após ver alguns trabalhos que utilizavam essa metaheurística para problemas de variantes baseadas no TSP. Porém, foi mais comum encontrar o SA inserido em híbridos mais complexos. Aqui, foi utilizado o SA puro visando também explorar a abordagem para um problema relativamente novo (introduzido em 2025 por Cerrone et al. [1]) na literatura.

O algoritmo proposto visa utilizar os mecanismos de exploração do SA para aplicar fixações sucessivas às variáveis y_r a cada iteração. São utilizados três operadores para definir o critério de fixação das variáveis e, além disso, o modelo relaxado é modificado para que x_a e u retomem suas características de binárias e inteiras, respectivamente. A intenção por trás disso é reduzir o problema por meio da fixação de variáveis ligadas aos gatilhos (que se mostraram ordens de grandeza mais impactantes nas instâncias) mas ainda ser capaz de usar a solução do solver como um caminho viável. Dessa forma, a cada iteração é possível produzir um *upper bound* em tempo polinomial $O(NTriggers * NArcs)$ no pior caso.

A cada iteração a busca realizada escolhe, aleatoriamente uma estratégia de fixação dentre as três implementadas. Essas estratégias se baseiam num sistema de pontuação para os gatilhos. Essa pontuação é construída a partir do peso do arco na relaxação linear (x_a) e na diferença entre o custo base do arco alvo e seu custo modificado após ter o gatilho ativado. Sendo assim, os operadores disponíveis tem políticas próprias que definem quais gatilhos y_r serão zerados por possuir um *score* considerado ruim.

O primeiro operador é o *topk* que seleciona uma fração definida (20%, por exemplo) das piores pontuações encontradas para ser fixada em zero. O segundo é o *weighted* que atribui

pesos aos gatilhos e coleta uma fração definida deles para fixar. E o terceiro é o *threshold* que basicamente olha para os valores das variáveis fracionárias y_r na relaxação e busca aqueles que superam um determinado valor (*threshold*) para serem fixados.

Em seguida, após aplicar o operador, o novo modelo é enviado ao *solver* e resolvido, produzindo um caminho viável porém com um custo distorcido. Para tanto é utilizado um método que recomputa o valor da função objetivo para atender ao problema original, produzindo assim um limitante superior válido.

E, seguindo o fluxo do *Simulated Annealing*, é utilizado o critério de Metropolis [5] para definir o aceite do novo limitante como sendo o melhor. Ao inserir uma probabilidade de aceitar soluções piores, é possível sair de eventuais ótimos locais e explorar melhor o espaço de solução.

Essa heurística de viabilização baseada em SA pode ser promissora, visto que permite um controle mais refinado da complexidade computacional e por aproveitar o custo informativo do LP relaxado. Sua flexibilidade permite adaptações tanto na definição dos operadores quanto nas estratégias de resfriamento, viabilizando uma busca balanceada entre exploração e intensificação[5].

Por fim, vale destacar a importância de um refinamento na calibração dos parâmetros, tanto da Relaxação Lagrangiana quanto da metaheurística do *Simulated Annealing*.

3 Experimentos Computacionais

Com o problema delineado, um modelo matemático pronto (TATSP) e três abordagens distintas implementadas, seguimos para a seção de experimentos visando avaliar o desempenho de cada ideia proposta.

Para realizar os experimentos foi utilizado um ambiente Ubuntu 22.04, com 16GB de RAM e um processador Intel Core $i7 - 12700H$ de 2.30 GHz. Além disso o *solver* escolhido foi o *Gurobi* sob uma licença acadêmica e a linguagem de programação utilizada para implementar as abordagens foi Julia na versão 1.11.

Os experimentos realizados se utilizam de instâncias fornecidas na competição MESS2024 [2]. Dada a extensa lista de instâncias, foram selecionadas três instâncias de cada grupo, totalizando 6 instâncias, sendo duas consideradas pequenas, duas médias e duas grandes. Para cada uma delas foram testados os algoritmos.

Para as instâncias menores, é possível notar que ambos os algoritmos conseguiram explorar o espaço de busca. Para a Relaxação Lagrangiana em ambos os casos (Figura 1 e Figura 3) houve uma parada antes do número de iterações estipulado. Isso foi uma regra adicional para interromper a execução caso se passassem 30% do total de iterações sem haver melhoras em quaisquer limitantes. Porém é interessante notar que há uma flutuação no decorrer da execução.

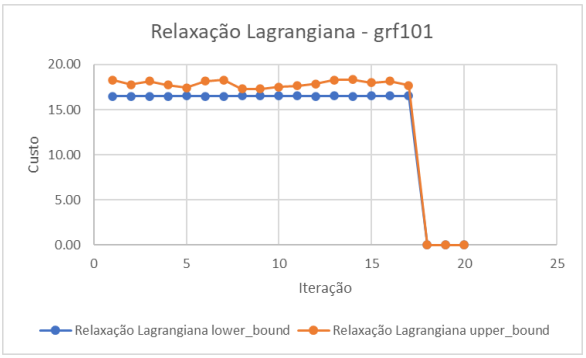

Figura 1: Relaxação Lagrangiana - grf101

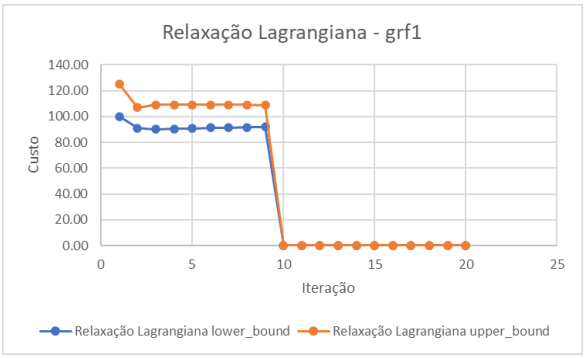

Figura 2: Simulated Annealing - grf101

Para o caso do *Simulated Annealing* (Figura 2 e Figura 4), notam-se picos na execução do

algoritmo ao longo das iterações, o que é um comportamento esperado para essa metaheurística. A parametrização, apesar de básica, conseguiu gerar diversidade nas soluções e, de certa forma, explorar as possibilidades. No que diz respeito ao uso de tempo, para as instâncias pequenas, o tempo limite de 10 minutos para cada uma foi suficiente para execução total de cada passo. Para *grf101* a RL utilizou 515.0 segundos e para o SA 339.0 segundos. Já para a *grf1*, a RL utilizou 279.0 segundos (*early stop* citado anteriormente) e para o SA 605.0 segundos.

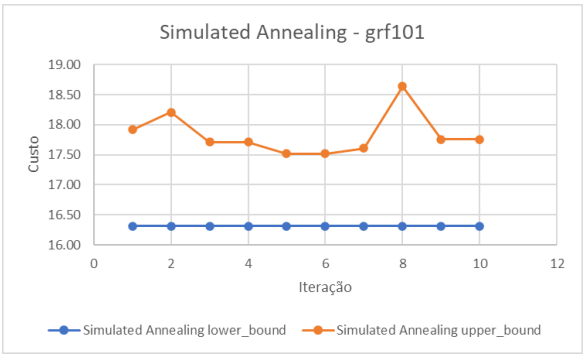

Figura 3: Relaxação Lagrangiana - grf1

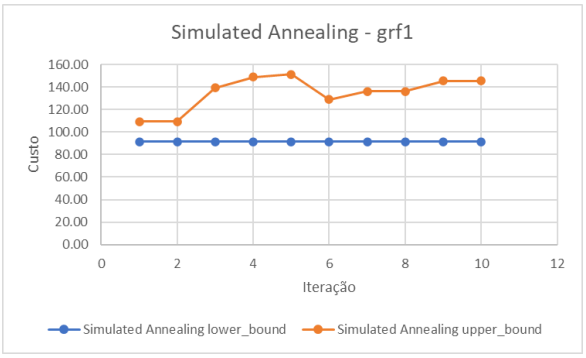

Figura 4: Simulated Annealing - grf1

Para as instâncias médias e grandes selecionadas, *grf8, grf112, grf18 e grf129*, infelizmente, foram encontradas barreiras com relação ao uso de memória para a execução tanto da Relaxação Lagrangiana, pois o número de gatilhos estoura a memória para o vetor de violações, especialmente para a restrição $C10$; e também para o SA, pois o modelo continua grande demais para o tempo limite dividido entre as iterações, atingindo 60 segundos ainda durante a fase de *pre-solve* do Gurobi.

Porém, para o ILP foi possível construir os modelos inteiros e iniciar a otimização, como mostra a Tabela 1. Apesar de não atingir *gaps* razoáveis para os casos em questão, foi ao menos possível obter uma solução viável, exceto para a instância *grf129*. E, Com exceção à instância *grf101*, considerada a menor, todas as outras atingiram o tempo limite sem alcançar valores ótimos. Apesar disso, para as instâncias menores, foi possível notar que os limitantes alcançados são representativos para os valores encontrados pelo *solver*, ou seja, limitam superior e inferiormente o ótimo do problema original.

Instâncias	Custo	Status
grf101	16.94	OPTIMAL
grf1	101.70	TIME_LIMIT
grf112	50.93	TIME_LIMIT
grf8	223.43	TIME_LIMIT
grf129	0.00	OUT_OF_MEMORY
grf18	355.55	TIME_LIMIT

Tabela 1: ILP para as seis instâncias.

Por último, foi realizado um experimento focado na instância *grf101* visando observar o desempenho da Relaxação Lagrangiana para diferentes dualizações das restrições de gatilhos. A Tabela 2 mostra como a dualização de restrições afeta os resultados mesmo em instâncias pequenas. A medida em que aumentamos o número de restrições dualizadas, nota-se uma piora na produção de limitantes, tanto inferiores quanto superiores. A queda mais brusca está ligada à restrição C10, tanto em tempo quanto qualidade. A suspeita mais forte é de que esse seja um conjunto de restrições muito grande, por ser construído aos pares de gatilhos, dois a dois. Além disso, isso afeta mais ainda a penalização da função objetivo.

C6	C7	C8	C9	C10	Lower Bound	Upper Bound	Time(s)
✓	✗	✗	✗	✗	16.53	17.32	515.0
✓	✗	✗	✓	✗	16.51	17.32	606.0
✓	✗	✗	✓	✓	16.42	17.75	98.0
✓	✓	✓	✓	✓	16.30	17.57	5.0

Tabela 2: Impacto da dualização progressiva das restrições C6-C10 sobre os limitantes inferior e superior.

E, por último, vale citar que, ao dualizar restrições que contém *Big M*, é notável a queda no tempo de execução, porém também nota-se uma flutuação expressivamente maior nos valores das violações, gerando problemas até de grandeza quando não controlados.

4 Conclusões

Neste projeto foi feito um trabalho de exploração de um problema recente e aplicação de métodos vistos ao longo da disciplina de Programação Linear Inteira. Foi possível ainda experimentar uma linguagem de programação nova e voltada à otimização e também experimentar heurísticas diferentes.

Para o método de fixação de variáveis ainda há espaço para explorar mais tipos de fixações e também de operadores, além disso, buscar métodos mais eficazes para viabilizar soluções na parte de relaxação Lagrangiana também é uma alternativa. Mesmo que durante todos os testes a impressão mais forte que ficou é que a Relaxação Lagrangiana talvez não seja um método muito adequado para este problema, ou que pelo menos ainda existem mais possibilidades de dualizações a serem exploradas. Por último, talvez também seja interessante aliar as propostas atuais ao *Concorde TSP solver*[6], que é reconhecido na literatura como extremamente eficiente para o TSP – mesmo que seja para o TSP simétrico, é possível adaptar o processamento das entradas para encaixar com as necessidades dele.

Com as implementações realizadas e após os experimentos, nota-se que ainda há muito espaço para melhora, tanto no uso de recursos quanto na parametrização dos algoritmos desenvolvidos. Apesar disso, é importante destacar o nível de discussão produzido ao atacar um problema novo. Adicionando uma nota pessoal: tive a oportunidade de discutir extensamente com os colegas sobre as abordagens, as limitações e dificuldades. Com essa rede de apoio foi possível, em um prazo relativamente curto, superar barreiras de conhecimento tanto teórico quanto técnico.

Bibliografia

[1] Carmine Cerrone, Maria Truvolo, Raffaele Dragone e Raffaele Battaglia. “The Trigger Arc TSP: Optimise the Picking Process in Warehouses with Compactable Storage Systems”. Em: *Decision Sciences*. Ed. por Angel A. Juan, Javier Faulin e David Lopez-Lopez. Cham: Springer Nature Switzerland, 2025, pp. 279–289. ISBN: 978-3-031-78238-1.

[2] *Metaheuristics Competition Race at MESS 2024*. 2024. DOI: https://fourclicks.eu/fck/mess2024/frontend/#/home/dashboard.

[3] L. A. Wolsey. *Integer Programming*. Wiley, 2020.

[4] Dimitris Bertsimas e John Tsitsiklis. *Introduction to Linear Optimization*. Jan. de 1998.

[5] M. Gendreau e J. Y. Potvin. *Handbook of Metaheuristics*. 2nd. Springer Publishing Company, Incorporated, 2010.

[6] *Concorde TSP Solver*. 2020. DOI: https://www.math.uwaterloo.ca/tsp/concorde/index.html.