Problema da Seleção de Pedidos Ótima - Abordagem metaheurística utilizando Busca Tabu

Leonardo Gaertner
UDESC
Ibirama, Santa Catarina
leonardo.gaertner@edu.udesc.br

Fernando Alexander Prim
UDESC
Ibirama, Santa Catarina
fernando.prim@edu.udesc.br

Kevin Grüfeld Strey
UDESC
Ibirama, Santa Catarina
kevin.strey@edu.udesc.br

RESUMO

Este trabalho apresenta uma abordagem baseada na metaheurística Busca Tabu para a seleção de pedidos e corredores em um ambiente logístico, respeitando restrições de capacidade e suprimento. A função objetivo visa maximizar a razão entre unidades totais processadas e o número de corredores utilizados. O algoritmo implementa estratégias de oscilação estratégica, diversificação, controle de estagnação e pós-processamento para aprimorar a qualidade das soluções. A geração de vizinhança incorpora movimentos de adição, remoção e troca de pedidos, com controle por lista tabu e critério de aspiração. A solução inicial é construída de forma aleatória e otimizada com base na demanda de itens.

PALAVRAS CHAVE. Busca Tabu, Otimização Combinatória, Logística.

ABSTRACT

This work presents an approach based on the Tabu Search metaheuristic for the selection of orders and aisles in a logistical environment, respecting capacity and supply constraints. The objective function aims to maximize the ratio between total processed units and the number of aisles used. The algorithm implements strategies such as strategic oscillation, diversification, stagnation control, and post-processing to improve solution quality. The neighborhood generation incorporates moves of order addition, removal, and exchange, with control through a tabu list and an aspiration criterion. The initial solution is constructed randomly and optimized based on item demand.

KEYWORDS. Tabu Search, Combinatorial Optimization, Logistics.

1. Introdução

Problemas de otimização combinatória estão presentes em diversos contextos logísticos, particularmente na seleção de pedidos e alocação de recursos sob restrições de capacidade. Tais problemas se caracterizam por um espaço de busca exponencialmente grande, o que inviabiliza abordagens exatas para instâncias de maior porte. Nesse contexto, metaheurísticas se destacam por sua capacidade de explorar espaços de solução complexos com eficiência.

Este trabalho propõe uma abordagem baseada na metaheurística *Busca Tabu* para selecionar um subconjunto de pedidos e corredores que maximize a eficiência da operação logística, definida como a razão entre o total de unidades atendidas e o número de corredores utilizados. A solução deve respeitar restrições de demanda, oferta e limites inferiores e superiores da quantidade total de unidades.

A implementação foi realizada em Python e utiliza estrutura modular com classes para representar instâncias e soluções. A geração da solução inicial é feita aleatoriamente, respeitando os limites de capacidade. A seleção de corredores é otimizada com base na cobertura da demanda utilizando um algoritmo guloso.

Durante o processo de busca, são gerados vizinhos por meio de três operações: adição, remoção e troca de pedidos. O controle de repetição é realizado por meio de uma lista tabu, e um critério de aspiração permite aceitar soluções tabu se estas forem significativamente melhores. A busca é encerrada por critérios de tempo ou número máximo de iterações.

Complementarmente, um conjunto de testes foi realizado para identificar os melhores parâmetros de execução da busca. Os experimentos foram conduzidos com diferentes instâncias e os resultados são discutidos ao longo do artigo.

2. Fundamentação Teórica

Nesta seção, apresenta-se os conceitos fundamentais que embasam o desenvolvimento do algoritmo: Busca Tabu, Metaheurísticas e formulação Matemática.

2.1. Metaheurísticas

Metaheurísticas são métodos de alto nível, independentes do problema, que direcionam a criação e combinação de heurísticas para encontrar boas soluções em espaços de busca muito grandes [Blum e Roli, 2003]. Essas técnicas combinam estratégias estocásticas e conhecimento acumulado para guiar a busca, sendo eficazes em problemas NP-difíceis, onde métodos exatos não são viáveis na prática [Blum e Roli, 2003]. Contudo, o teorema do “No Free Lunch” estabelece que nenhuma metaheurística é universalmente superior; sua eficiência depende da adequação ao problema específico [Ho e Pepyne, 2002; Wolpert e Macready, 1997].

2.2. Busca Tabu

A Busca Tabu é uma metaheurística de busca local desenvolvida por Glover nos anos 80, que utiliza estruturas de memória para evitar ciclos e escapar de ôtimos locais. Essa técnica explora a vizinhança de uma solução corrente, permitindo movimentos piores quando necessários, mas proibindo movimentos recentemente utilizados por meio de uma lista tabu com *tenure* limitada [Glover, 1989]. A memória de curto prazo previne a volta a soluções recentes, enquanto memórias de médio e longo prazo promovem intensificação e diversificação da busca [Glover, 1989; Coleman e Kelly, 2005].

O algoritmo geral da Busca Tabu pode ser representado conforme o pseudocódigo mostrado na Figura 1.

Figura 1: Pseudocódigo geral da Busca Tabu

```
1: s ← GerarSoluçãoInicial()
2: s_best ← s
3: tabuList ← ∅
4: while critério de parada não for atingido do
5:     N ← GerarVizinhança(s)
6:     N' ← {s' ∈ N | s' não é tabu ou satisfaz critério de aspiração}
7:     s ← melhor solução em N'
8:     if f(s) > f(s_best) then
9:         s_best ← s
10:    end if
11:    AtualizarListaTabu(tabuList, movimento aplicado)
12: end while
13: return s_best
```

Neste esquema, `N` representa o conjunto de soluções vizinhas geradas a partir da solução atual `s`. O conjunto `N'` consiste em soluções que não estão na lista tabu ou que satisfazem o critério de aspiração (como ser melhor que a melhor solução encontrada até o momento). O controle da lista tabu é fundamental para garantir uma busca exploratória eficiente.

A eficiência da Busca Tabu depende da qualidade do mecanismo de geração de vizinhos, da estrutura de memória e dos critérios de aspiração utilizados, aspectos que foram considerados para o desenvolvimento do algoritmo proposto neste trabalho.

2.3. Justificativa da Abordagem Heurística

A natureza combinatória e a presença de múltiplas restrições tornam o problema intratável para abordagens exatas em instâncias de grande escala. A ausência de formulação linear pura e a complexidade envolvida nas combinações viáveis justificam a escolha de uma metaheurística.

Dessa forma, optou-se por uma implementação da metaheurística Busca Tabu, desenvolvida inteiramente em Python, sem dependência de solvers comerciais como CPLEX ou Gurobi. Isso assegura maior flexibilidade, reprodutibilidade e adaptabilidade do algoritmo ao problema em estudo.

3. Formulação Matemática do Problema

Seja $O = \{1, 2, \ldots, n\}$ o conjunto de pedidos e $C = \{1, 2, \ldots, m\}$ o conjunto de corredores. Cada pedido $i \in O$ demanda um conjunto de itens I_i com respectivas quantidades d_{ij}. Cada corredor $j \in C$ possui um estoque de itens s_{jk}.

Variáveis de decisão:

- $x_i \in \{0, 1\}$: indica se o pedido i foi selecionado.
- $y_j \in \{0, 1\}$: indica se o corredor j foi utilizado.

Função objetivo:

$$\max \frac{\sum_{i \in O} x_i \cdot u_i}{\sum_{j \in C} y_j} \tag{1}$$

onde u_i representa o total de unidades no pedido i.

Sujeito a:

$$(1)\ LB \leq \sum_{i \in O} x_i \cdot u_i \leq UB \tag{2}$$

$$(2)\ \sum_{i \in O} x_i \cdot d_{ij} \leq \sum_{j \in C} y_j \cdot s_{jk}, \quad \forall \text{ item } k \tag{3}$$

4. Desenvolvimento

O desenvolvimento foi organizado em dois módulos principais: o primeiro, `main.py`, implementa a lógica do algoritmo de Busca Tabu; o segundo, `ajuste_parametros.py`, realiza a calibração automática dos parâmetros mais sensíveis do algoritmo. Ambos foram implementados em Python e testados em diversas instâncias com o objetivo de maximizar o desempenho da solução quanto à função objetivo proposta.

4.1. Módulo Principal: `main.py`

O módulo principal contém as classes que modelam o problema e a metaheurística. A geração da solução inicial respeita os limites inferiores e superiores de capacidade, conforme mostrado no trecho a seguir:

```
def generate_initial_solution(instance: Instance, max_attempts=100) -> Solution:
    for attempt in range(max_attempts):
        orders = set()
        total_units = 0
        order_indices = list(range(instance.num_orders))
        random.shuffle(order_indices)
        ...
        if solution.is_feasible(instance):
            solution.compute_objective(instance)
            return solution
```

Listing 1: Geração da solução inicial

O processo de geração de vizinhos realiza modificações pontuais na solução corrente com três tipos de movimentos: adição, remoção e troca. Apenas soluções viáveis são consideradas:

```
def generate_neighbor(current: Solution, instance: Instance) -> Solution:
    neighbor_orders = current.orders.copy()
    move_type = random.choice(['add', 'remove', 'swap'])
    ...
    if neighbor.is_feasible(instance):
        neighbor.compute_objective(instance)
        return neighbor
    else:
        return None
```

Listing 2: Geração de vizinhos

A busca tabu propriamente dita considera uma lista de movimentos proibidos (tabu) e critério de aspiração. Iterativamente, seleciona o melhor vizinho disponível com base na função objetivo:

```
def tabu_search(instance: Instance, time_limit=600.0,
                tabu_tenure=10, max_iterations=50, neighborhood_size=20):
    ...
```

```
    while iteration < max_iterations and (time.time() - start_time) < time_limit
        :
        ...
        if best_neighbor_obj > best_objective:
            best_solution = best_neighbor
            best_objective = best_neighbor_obj
```

Listing 3: Busca Tabu com controle de iterações e tempo

4.2. Ajuste de Parâmetros: `ajuste_parametros.py`

Este módulo executa uma série de experimentos automáticos para identificar os melhores valores dos três principais parâmetros da busca tabu:

- `TABU_TENURE`: tempo de permanência de movimentos na lista tabu;
- `MAX_ITERATIONS`: número máximo de iterações da busca;
- `NEIGHBORHOOD_SIZE`: número de vizinhos gerados por iteração.

O código testa sucessivamente valores incrementais, registrando o valor da função objetivo para cada configuração. A cada nova execução, se houver melhora, o parâmetro é mantido; caso contrário, o processo é interrompido após três tentativas sem ganho. O trecho a seguir apresenta a lógica para avaliação dos parâmetros:

```
def avaliar_parametro(nome_param, valores, fixos, instancias):
    ...
    while sem_melhora < MAX_NO_IMPROVE:
        nova_execucao = valores[-1] + INCREMENT
        ...
        for nome_inst, inst in instancias:
            config = { ... }
            sol = tabu_search(inst, ..., **config)
            ...
```

Listing 4: Ajuste iterativo dos parâmetros

Ao final do processo, os resultados são exportados para um arquivo Excel contendo, para cada combinação de parâmetros, a média da função objetivo, tempo de execução e tamanho das soluções. O trecho abaixo mostra a exportação final.

5. Análise de Resultados Paramétricos

Para avaliar o impacto dos parâmetros da Busca Tabu na qualidade das soluções, foi realizada uma série de experimentos sobre 10 instâncias distintas. Cada execução teve como objetivo maximizar a razão entre o total de unidades atendidas e o número de corredores utilizados, comparando esse valor com o **melhor objetivo teórico conhecido** da respectiva instância.

Os resultados de cada configuração estão registrados na planilha ajuste_parametros_resultado.xlsx, a qual também está disponível no repositório do projeto.

5.1. Resultados Paramétricos por Instância

A Tabela 1 apresenta os dados principais coletados nos experimentos, incluindo o valor do objetivo obtido, tempo de execução e o melhor valor teórico conhecido para cada instância.

Tabela 1: Desempenho do algoritmo por instância.

Instância	TABU_TENURE	MAX_ITERATIONS	NEIGHBORHOOD_SIZE	Objetivo	Melhor Objetivo
instance_0001.txt	115	270	60	15.00	15.00
instance_0002.txt	25	100	20	2.00	2.00
instance_0003.txt	115	270	60	11.67	12.00
instance_0004.txt	25	100	20	3.50	3.50
instance_0005.txt	115	250	20	41.34	177.88
instance_0006.txt	115	270	20	20.44	691.00
instance_0007.txt	115	270	60	14.23	392.25
instance_0008.txt	115	270	40	35.95	162.94
instance_0009.txt	25	100	20	4.42	4.42
instance_0010.txt	115	240	20	8.36	16.79

Observa-se que em instâncias simples (como instance_0001.txt e instance_0002.txt) o algoritmo foi capaz de atingir o objetivo ótimo em tempo desprezível, mesmo com parametrizações modestas.

Em contrapartida, nas instâncias mais complexas (instance_0005.txt até instance_0010.txt), a diferença entre o valor obtido e o melhor objetivo conhecido torna-se significativa, refletindo a complexidade do espaço de busca. A média de tempo de execução para essas instâncias também é substancialmente maior, atingindo cerca de 4 a 5 minutos por execução.

5.2. Melhor Configuração Paramétrica

Com base nos resultados gerais, a configuração que apresentou o melhor equilíbrio entre qualidade das soluções e tempo computacional foi:

- `TABU_TENURE` = 115
- `MAX_ITERATIONS` = 250
- `NEIGHBORHOOD_SIZE` = 20

Essa combinação foi utilizada para as execuções finais do algoritmo, permitindo um controle mais efetivo da lista tabu e um número viável de iterações e vizinhos por ciclo.

5.3. Testes com Parâmetros Definidos

Após a identificação da melhor configuração paramétrica, o algoritmo foi testado em um novo conjunto de instâncias, com os parâmetros fixados.

Esses testes têm como objetivo avaliar a capacidade da abordagem em generalizar seu desempenho em diferentes cenários. As instâncias analisadas são de nível intermediário a alto, com variação significativa no número de pedidos e corredores. A Tabela 2 apresenta os resultados obtidos:

Tabela 2: Resultados obtidos nas instâncias instance_0015.txt a instance_0020.txt com parâmetros definidos.

Instância	Tempo (s)	Objetivo	Nº Pedidos	Nº Corredores	Melhor Objetivo
instance_0015.txt	301,81	4,05	325	110	149,33
instance_0016.txt	52,33	12,50	375	30	85,00
instance_0017.txt	21,46	11,67	175	15	36,50
instance_0018.txt	75,90	15,86	667	58	117,20
instance_0019.txt	52,75	15,59	336	32	202,00
instance_0020.txt	0,00	5,00	5	2	5,00

O algoritmo obteve soluções viáveis em todas as instâncias, respeitando os limites de propostos. Em instance_0020.txt, foi atingido o valor ideal de 5, indicando sucesso absoluto em uma instância pequena.

Nas demais instâncias, observa-se que o valor do objetivo alcançado representa uma fração significativamente menot que o melhor objetivo conhecido. Com isso, ainda há espaço para melhorias, principalmente em casos de alta complexidade como instance_0015.txt, onde a razão foi significativamente inferior.

Esses resultados reforçam que, mesmo com parâmetros fixos, o algoritmo apresenta desempenho robusto e generalizável a diferentes padrões de carga e estoque.

6. Conclusão

Neste trabalho, foi desenvolvida uma abordagem baseada na metaheurística Busca Tabu para resolver o problema da seleção de pedidos e corredores em um ambiente logístico, com restrições de capacidade e suprimento. O algoritmo incorporou mecanismos clássicos como lista tabu, critério de aspiração, oscilação estratégica, controle de estagnação e pós-processamento.

Apesar da sofisticação da abordagem, os resultados obtidos em instâncias mais complexas não foram satisfatórios. Em várias situações, a solução encontrada representou uma fração modesta da melhor solução conhecida. Isso indica que a estratégia utilizada, embora eficaz em instâncias pequenas, perde eficiência conforme o espaço de busca se torna mais denso e irregular.

Entre os principais fatores que podem ter comprometido o desempenho estão o tamanho limitado da vizinhança explorada, a dificuldade de encontrar boas soluções iniciais e a falta de mecanismos adaptativos durante a execução.

7. Trabalhos Futuros:

- Investigar heurísticas construtivas mais robustas para geração da solução inicial;
- Explorar vizinhanças mais estruturadas (por exemplo, baseadas em agrupamento de pedidos);
- Implementar variantes híbridas com outras metaheurísticas, como GRASP ou ILS;
- Realizar testes com instâncias maiores e diferentes perfis de demanda, para validar a escalabilidade da abordagem.

Em síntese, o trabalho apresentou uma contribuição relevante ao aplicar a Busca Tabu a um problema de otimização logística não trivial, estabelecendo uma base para aprimoramentos futuros e comparações com métodos alternativos.

Todo o código-fonte, incluindo as implementações da Busca Tabu, scripts de ajuste de parâmetros e testes, está publicamente disponível no repositório GitHub: `https://github.com/leonardogaertner/TabuSearchSBPO`.

Referências

Blum, C. e Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.

Coleman, T. F. e Kelly, J. T. C. (2005). A practical introduction to optimization with matlab. *SIAM Review*, 47(3):434–436.

Glover, F. (1989). Tabu search–part i. *ORSA Journal on Computing*, 1(3):190–206.

Ho, Y. C. e Pepyne, D. L. (2002). Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications*, 115(3):549–570. URL https://doi.org/10.1023/A:1021251113462.

Wolpert, D. H. e Macready, W. G. (1997). No-free-lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82. URL https://doi.org/10.1109/4235.585893.