UMA IMPLEMENTAÇÃO DO MÉTODO PATAKI NA SOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE

Mateus de Assis Chacon Lima
Limeira-SP, Brasil
m271415@g.unicamp.br

Mario Mendoza Villalba
Limeira-SP, Brasil
m247931@dac.unicamp.br

Danielle Vasconcelos Gomes
Limeira-SP, Brasil
d245465@dac.unicamp.br

Felipe Freitas Viveiros Ribeiro
Limeira-SP, Brasil
f225667@g.unicamp.br

ABSTRACT

"The present work aims to implement the strategy proposed by Pataki in 2003 to solve the Traveling Salesman Problem (TSP). This strategy is tested on three datasets of different dimensions. This study compares the application of the MTZ model with two different strategies for implementing Pataki's algorithm (which combines MTZ with the sub-route formulation), comparing their results both in terms of response quality (GAP) and computational time."

Keywords: Traveling salesman problem, sub-tours, MTZ formulation.

RESUMO

O presente trabalho busca implementar a estratégia proposta por Pataki em 2003 para resolver o Problema do Caixeiro Viajante (TSP). Essa estratégia foi testada em três conjuntos de dados com diferentes dimensões. Este estudo compara a aplicação do modelo MTZ com duas estratégias diferentes para a implementação do algoritmo de Pataki, que combina o MTZ com a formulação de sub-rotas. A comparação é realizada com base nos resultados, tanto em termos de qualidade da resposta (GAP) quanto em tempo computacional.

Palavras-chave: Caixero viajante, sub rotas, formulação MTZ

1 INTRODUÇÃO

O problema do Caixeiro Viajante (TSP) é um dos problemas de otimização combinatória mais conhecidos [1]. Está entre os tópicos mais desafiadores da pesquisa operacional e consiste em determinar uma rota ótima que passe por cada nó apenas uma vez [2].

O destaque a esse tipo de problema encontra-se em sua complexidade, visto que o número de possíveis rotas cresce exponencialmente com o aumento das cidades, tornando a exploração completa da solução impraticável para instâncias de tamanho razoável. Isso incitou o desenvolvimento de abordagens algorítmicas que contornem a grande quantidade de restrições necessárias inicialmente.

Nesse contexto, o presente projeto se propôs a abordar o problema do CV através do Método MTZ (Miller, Tucker e Zemlin) [3] e do Modelo proposto por Pataki em 2003 [4]. A escolha do Método MTZ se deu pelo seu foco na eliminação de subciclos, evitando soluções inválidas. Por sua vez, o Modelo de Pataki introduz reformulações matemáticas inovadoras, visando aprimorar a eficiência do processo de resolução.

Para avaliar a eficácia dessas abordagens, foram apresentadas três instâncias (ver anexo 10.2) de crescente complexidade, apresentadas posteriormente. A análise dos resultados contempla a distância total percorrida nas rotas encontradas, assim como o tempo de execução das abordagens para cada instância. Além disso, a qualidade das soluções será comparada com resultados ótimos conhecidos, quando disponíveis, para avaliar a proximidade das soluções propostas ao valor mínimo teórico.

3. EXPOSIÇÃO DO PROBLEMA

O projeto abordou o desafio do Problema do Caixeiro Viajante em três instâncias específicas - bays29, brazil58 e si535 - com o objetivo de encontrar suas soluções ótimas ou próximas a ótima. Para tanto, foram empregados o Método MTZ (Miller, Tucker e Zemlin) e o método desenvolvido por Pataki em 2003, visando a comparação e análise dos resultados obtidos por ambas abordagens.

Cada instância apresenta uma configuração distinta, variando em relação ao número de cidades envolvidas e à complexidade das rotas a serem percorridas. O desafio central consistiu em determinar a rota mais curta que permitisse ao caixeiro viajante visitar todas as cidades uma única vez, antes de retornar à cidade de origem.

O Método MTZ utiliza uma matriz de distâncias entre as cidades para calcular a rota ótima. Adicionalmente, emprega variáveis auxiliares para evitar a formação de subciclos, garantindo a obtenção de uma solução coerente.

O Modelo proposto por Pataki em 2003 introduziu inovações na representação matemática do Problema do Caixeiro Viajante. Através de reformulações e relaxações, o modelo busca explorar a estrutura do problema de maneira mais eficaz, possibilitando a

obtenção de soluções de alta qualidade em um intervalo de tempo menor.

4. MODELOS

O modelo TSP, proposto por [5] em 1954, consta da seguinte formulação básica:

Variáveis:

$x_{ij} = \{1,\ se\ a\ rota\ ij\ é\ percorrida\ 0,\ se\ não$

Parâmetros:

C_{ij}: a distância do nó i ao nó j

Função objetivo:

$$MIN\ Z = \sum_{i \in I} \sum_{j \in J} C_{ij} x_{ij}$$

Restrições:

$$(1) \quad \sum_{i \in I} x_{ij} = 1,\ \forall j \in J$$

$$(2) \quad \sum_{j \in J} x_{ij} = 1,\ \forall i \in I$$

$$(3) \quad 0 \leq x_i \leq 1, \qquad x_i \in Z$$

4.1. Formulação MTZ

Proposta por [3] em 1960, adiciona ao modelo TSP as seguintes restrições, para evitar as sub rotas:

$$(4) \quad u_1 = 1,$$

$$(5) \quad 2 \leq u_i \leq n\ \forall i \neq 1$$

$$(6) \quad ui - uj + 1 \leq (n-1)(1 - xij)\ \forall i \neq 1, \forall j \neq 1$$

4.2. Formulação de sub rotas

Esta formulação, como se apresenta em [6] emprega a seguinte restrição no modelo TSP para excluir os subconjuntos que criam sub rotas:

$$(4) \sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad (S\_V,\ |S| > 1)$$

5. Algoritmo Pataki

O método Pataki combina as formulações anteriores, procurando uma maior eficiência computacional, aproveitado as vantagens de cada um dos modelos, a seguir se enuncia a estrutura do algoritmo:

1	Deixe a formulação do IP consistir apenas das restrições de atribuição, e $k = 1$.
2	**Enquanto** $k \leq maxrounds$ (2a) resolva o IP sobre a formulação atual. Suponha que a solução ótima consiste em r subtours $\{S1, \ldots, Sr\}$ (2b) se $r = 1$, PARE; a solução é ótima para o Problema do Caixeiro Viajante (TSP). Caso contrário, adicione à formulação no máximo 1000 restrições de subtour, nas quais S é a união de vários conjuntos S_i e $\|S\| \leq n/2$. Defina $k = k + 1$. **Fim do Enquanto**
3	Adicione as restrições de arco à formulação e resolva o IP até a otimalidade.

Pseudocódigo do método Pataki

6. EXPERIMENTO E RESULTADOS

A seguir se apresentam os critérios e a lógica da implementação do experimento

6.1. Critérios

Para o experimento computacional, foram utilizados os seguintes critérios:

6.1.1. Regra para leitura de uma sub rota

O processo se inicia com a representação da solução obtida pelo solver em coordenadas de origem e destino. Junto a isso, há um conjunto de verificação de cidades que ainda não foram visitadas e uma lista vazia destinada a guardar as sub-rotas que serão geradas. O procedimento percorre as seguintes etapas para a criação dessas sub-rotas:

- Selecione uma cidade não visitada como ponto de partida de uma nova sub-rota.
- Constrói uma sub-rota adicionando cidades adjacentes conforme as coordenadas fornecidas.
- Cada nova cidade é adicionada à sub-rota se suas coordenadas de chegada coincidirem com as coordenadas de saída da cidade anterior na sub-rota.
- A cidade recém-adicionada é removida da lista de verificação de cidades não visitadas.
- O processo continua até que não seja mais possível adicionar cidades à sub-rota.
- A sub-rota construída é armazenada em uma lista de sub-rotas.

- O método continua a construir novas sub-rotas até que todas as cidades do conjunto de verificação sejam visitadas.

6.1.2. Regras para inserção de restrições de eliminação de sub rotas

Para a inserção das restrições de eliminação de sub rotas se implementaram duas estratégias, descritas a seguir:

6.1.2.1. Estratégia 1:

A ideia central é criar uma restrição para cada sub-rota identificada durante o processo. Cada sub-rota representa um conjunto ordenado de cidades percorridas em sequência. Ao adicionar uma restrição correspondente a cada sub-rota no modelo, estamos essencialmente indicando ao solucionador quais combinações de cidades são válidas. Basicamente, o processo consiste em empregar a lista de sub-rotas obtida por meio do método que as gera, e então, aplicar o procedimento de inclusão das sub-rotas conforme expresso na equação 4.

6.1.2.2. Estratégia 2:

A estratégia essencial concentra-se em introduzir uma permutação do vetor de sub-rota como parte do processo. Cada vetor de sub-rota encapsula uma sequência ordenada de cidades que serão percorridas pelo caixeiro-viajante. Em termos concretos, essa abordagem consiste em utilizar os vetores de sub-rota obtidos previamente. Através da aplicação da permutação desses vetores, buscamos reorganizar a ordem das cidades, mantendo a integridade das sub-rotas.

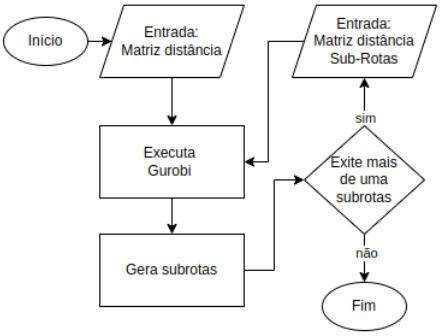
A permutação é realizada de modo a explorar diferentes combinações de cidades dentro das sub-rotas. Isso acrescenta uma camada de flexibilidade ao modelo, permitindo a avaliação de diferentes sequências de cidades. No entanto, devido à considerável quantidade de permutações que um vetor pode apresentar, será estabelecido um limite a fim de evitar potenciais problemas de memória. Esse limite será determinado pelo tamanho do vetor da sub-rota.

6.1.3. Critérios de parada

Na abordagem à resolução do problema do caixeiro-viajante, incorporamos dois critérios de parada fundamentais: um limite de tempo de uma hora e a identificação da solução ótima mediante um único subconjunto de rotas. Em outras palavras, quando o método que gera sub-rotas resulta em apenas um subconjunto, consideramos que alcançamos uma solução viável ou ideal para o problema

6.2. Estrutura do algoritmo implementado

Em resumo, o código é um loop que executa um processo iterativo de otimização. Em cada iteração, ele tenta melhorar a solução usando o Gurobi e as coordenadas fornecidas. Se encontrar uma solução com apenas uma sub-rota, ele retorna os resultados. Caso contrário, continua a otimização, adicionando as novas sub-rotas geradas à lista existente.



Fluxograma 1: Estrutura do algoritmo

7. SOLUÇÃO

A seguir, na Tabela 1, é visto os respectivos valores da função objetivo para cada data set. O link com o arquivo que contém as rotas fornecidas pelas estratégias implementadas para o experimento e o código no software Gurobi são apresentados no anexo 10.3.

	Bays29	**Brazil58**	**Si535**
Modelo MTZ	2020.0	25395.0	Não solução factível
Pataki, Estratégia 1	2020.0	25395.0	48449.999
Pataki, Estratégia 2	2020.0	25395.0	48449.999

Tabela 1: Quadro comparativo quanto às soluções ótimas encontradas.

Nota-se que todas as implementações – MTZ e Estratégias 1 e 2 de Pataki - alcançaram a solução ótima de 2020 para a instância bays29 e de 25395 para a

instância brazil58. Isso demonstra a eficácia de todas as abordagens para instâncias de tamanho moderado.

Percebe-se que o MTZ não conseguiu convergir em uma solução ótima para a instância si535. No entanto, ambas as estratégias de Pataki conseguiram encontrar uma solução próxima à ótima, que seria no valor de 48267. Isso sugere que, enquanto o MTZ pode encontrar dificuldades em lidar com instâncias de maior complexidade, as estratégias de Pataki foram mais robustas e bem-sucedidas na sua resolução.

Sequenciando a análise, na Tabela 2, os valores para o tempo computacional e o GAP obtido mediante as diferentes estratégias empregadas no presente estudo são apresentados. Para a execução do experimento, foi usado o solver Gurobi, versão 10.0.2; no anexo 10.1 são ampliadas as especificações técnicas de hardware e software.

Data set 1 (Bays29)		
Estratégia	**GAP**	**Tempo**
Modelo MTZ	0	1s
Pataki, Estratégia 1	0	0s, *3 iterações*
Pataki, Estratégia 2	0	0s, *3 iterações*
Data set 2 (Brazil58)		
Estratégia	**GAP**	**Tempo**
Modelo MTZ	0	255s
Pataki, Estratégia 1	0	1s, *6 iterações*
Pataki, Estratégia 2	0	2s, *6 iterações*
Data set 3 (si535)		
Estratégia	**GAP**	**Tempo**
Modelo MTZ	Não solução factível	2766s *(Memória esgotada)*
Pataki, Estratégia 1	4.12e-05	1368s *19 iterações*
Pataki, Estratégia 2	4.12e-05	1340s *20 iterações*

Tabela 2: Quadro comparativo quanto ao GAP e Tempo Computacional de cada estratégia.

Nota-se que, para a instância bays29, todas as estratégias apresentaram um gap de 0, chegando então a solução ótima. Além disso, o tempo computacional para o MTZ foi de 1 segundo, enquanto as Estratégias de Pataki conseguiram alcançar a solução ótima com tempos menores, em torno de 0 segundos.

Já no caso da instância brasil58, todas as estratégias também alcançaram um gap de 0, alcançando valor ótimo. Além disso, percebe-se que o MTZ exigiu um tempo computacional de 255 segundos, enquanto as Estratégias de Pataki alcançaram resultados comparáveis em tempos consideravelmente menores, na ordem de 1 a 2 segundos, já evidenciando uma maior eficácia se comparado ao MTZ.

Por fim, no caso da instância si535, o MTZ não conseguiu encontrar uma solução, resultando em um gap infactível. Além disso, o tempo computacional foi significativamente maior e levou à exaustão de memória após 2766 segundos. As Estratégias de Pataki, por outro lado, conseguiram atingir um gap considerado baixo, evidenciando a proximidade das soluções encontradas em relação à solução ótima. Apesar de ser perceptível que possuem tempos computacionais substanciais, eles foram consideravelmente menores em comparação ao MTZ.

8. CONCLUSÕES

Em conclusão, a análise dos resultados evidencia as capacidades e limitações das diferentes abordagens para resolver o Problema do Caixeiro Viajante em instâncias variadas. Enquanto o MTZ se mostrou menos eficaz para instâncias maiores, as Estratégias de Pataki demonstraram consistência ao encontrar soluções ótimas em cenários desafiadores, oferecendo uma alternativa poderosa para a resolução de instâncias complexas.

Ao observar as soluções ótimas encontradas para as instâncias bays29 e brazil58, tornou-se evidente que todas as estratégias demonstraram capacidade em produzir soluções ótimas. Para essas instâncias de tamanho moderado, as diferenças entre as abordagens foram sutis, refletindo sua competência na resolução de cenários menos complexos.

Entretanto, a instância si535 mostra o método MTZ como incapaz de convergir para uma solução, enquanto as Estratégias de Pataki conseguiram fazê-lo com um gap extremamente baixo. Isso sugere que, à medida que a complexidade aumenta, a robustez e a eficácia das Estratégias de Pataki emergem como vantagens notáveis. Considerando os tempos computacionais, as Estratégias de Pataki também se destacaram. Para as instâncias abordadas, as soluções foram encontradas em tempos consideravelmente menores do que o Método MTZ, demonstrando maior eficiência.

9. BIBLIOGRAFIA

[1] A. L. C. Ottoni, E. G. Nepomuceno, M. S. d. Oliveira, and D. C. R. d. Oliveira, "Reinforcement learning for the traveling salesman problem with refueling," *Complex Intell. Syst.*, vol. 8, no. 3, pp. 2001–2015, 2022, doi: 10.1007/s40747-021-00444-4.

[2] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *Eur. J. Oper. Res.*, vol. 59, no. 2, pp. 231–247, 1992, doi: 10.1016/0377-2217(92)90138-Y.

[3] C. E. Miller, R. A. Zemlin, and A. W. Tucker, "Integer Programming Formulation of Traveling Salesman Problems," *J. ACM*, vol. 7, no. 4, pp. 326–329, 1960, doi: 10.1145/321043.321046.

[4] G. Pataki, "Formulations Using the Traveling Salesman Problem," *SIAM Rev.*, vol. 45, no. 1, pp. 116–123, 2003.

[5] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a Large-Scale Traveling-Salesman Problem," *J. Oper. Res. Soc. Am.*, vol. 2, no. 4, 1954, doi: 10.1287/opre.2.4.393.

[6] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, "The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.," *J. Oper. Res. Soc.*, vol. 37, no. 5, 1986, doi: 10.2307/2582681.

10. ANEXOS

10.1	**Especificações técnicas do hardware e software empregados no experimento computacional**

```
SET PARAMETER USERNAME
ACADEMIC LICENSE - FOR NON-COMMERCIAL USE ONLY - EXPIRES 2024-08-14
SET PARAMETER TIMELIMIT TO VALUE 3600
GUROBI OPTIMIZER VERSION 10.0.2 BUILD V10.0.2RC0 (LINUX64)

CPU MODEL: INTEL(R) CORE(TM) I5-8250U CPU @ 1.60GHZ, INSTRUCTION SET [SSE2|AVX|AVX2]
THREAD COUNT: 4 PHYSICAL CORES, 8 LOGICAL PROCESSORS, USING UP TO 8 THREADS
```

10.2	**Instancias, resultados, Código do experimento.**

https://acortar.link/t0e8hL