Sequential Ordering Problem Utilizando a Metaheurística de Busca Tabu

Augusto Bennemann[1], **Fabrício M. Mazzola**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)

{abennemann,fmmazzola}@inf.ufrgs.br

1. Introdução

O trabalho apresentado visa estudar um problema NP-completo, desenvolver sua formulação matemática em programação inteira e implementar uma meta-heurística a ser aplicada sobre esse problema. Primeiramente, é feito um estudo sobre o problema *Sequential Ordering Problem*, abordado nesse trabalho. Esse problema consiste em uma variação do *Problema do Caixeiro Viajante Assimétrico*. Na Seção 2, é apresentada a descrição detalhada do problema e é desenvolvida sua formulação matemática em programação inteira. Em seguida, na Seção 3 é abordada em detalhes a *Busca Tabu*, meta-heurística utilizada na resolução do problema.

Na Seção 4, é apresentada uma avaliação comparativa da resolução do problema entre os métodos Programação Linear e Meta-Heurística. Por fim, na Seção 5 são apresentadas as conclusões e considerações finais acerca do trabalho desenvolvido.

2. Problema

O problema a ser resolvido neste trabalho é o *Sequential Ordering Problem (SOP)*. Esse problema é uma variação do *Problema do Caixeiro Viajante Assimétrico (ATSP)*. O objetivo é encontrar o caminho de menor custo, respeitando restrições de precedência entre os vértices definidos na instância. Esse caminho deve visitar todos os vértices do grafo exatamente uma vez, partindo do vértice 1 e chegando ao vértice N.

A formulação matemática desenvolvida para esse problema, em programação inteira, foi baseada nos estudos de [Letchford and Salazar-González 2016, Balas et al. 1995] e é mostrada a seguir:

Conjuntos

V conjunto de vértices (1 a n)

Parâmetros

c_{ij} $i \in V, j \in V$ custo do vértice *i* ao vértice *j*

p_{ij} $i \in V, j \in V$ $\begin{cases} 1, & \text{se vértice } i \text{ precede o vértice } j \\ 0, & \text{caso contrário} \end{cases}$

Variáveis

x_{ij} $i \in V, j \in V$ $\begin{cases} 1, & \text{se arco } (i,j) \text{ presente na solução} \\ 0, & \text{caso contrário} \end{cases}$

o_i $i \in V$ ordem de visitação $[0,n-1]$

Função objetivo

$$\min \quad \sum_{i \in V} \sum_{j \in V - \{i\}} x_{ij} c_{ij} \tag{1}$$

Restrições

$$\sum_{j \in V - \{i\}} x_{ij} = 1, \ \forall i \in V \setminus \{n\} \tag{2}$$

$$\sum_{i \in V - \{j\}} x_{ij} = 1, \ \forall j \in V \setminus \{1\} \tag{3}$$

$$x_{n,j} = 0, \ \forall j \in V. \tag{4}$$

$$x_{i,1} = 0, \ \forall i \in V. \tag{5}$$

$$o_i + 1 \leq o_j, \ \forall i, j \in V, \ i \neq j, \ p_{ij} = 1 \tag{6}$$

$$o_i + 1 \leq o_j - M * (x_{i,j} - 1), \ \forall i, j \in V, \ i \neq j. \tag{7}$$

$$x_{i,j} \in \{0, 1\} \tag{8}$$

$$o_i \in R^+ \tag{9}$$

O problema é representado através de um grafo direcionado G = $\langle$ V, E $\rangle$ definido sobre um conjunto V de n vértices e um conjunto E de $n * n$ arcos direcionados. Cada aresta possui um custo $c_{i,j}$ associado, que representa o tempo de viagem entre dois vértices. Além disso, o grafo possui um vertice de origem $v_0 \in V$, um vértice final $v_n \in V$ e pares de precedência (v_i, v_j), v_i, $v_j \in V$. Cada par de precedência (v_i, v_j) indica que um vértice v_j pode ser visitado se, e somente se, o vértice v_i já tenha sido visitado. Os pares de precedência representam restrições para o problema e influenciar diretamente na ordem em que os vértices serão visitados.

Por exemplo, considerando um grafo direcionado com quatro vértices que representam cidades do Rio Grande do Sul (Porto Alegre, Viamão, Canoas e Gravataí). Um par de precedência (Viamão, Canoas) indica que Canoas somente poderá ser visitada depois que a cidade de Viamão tenha sido frequentada. Consequentemente, esse par de precedência influenciará diretamente na ordem que as cidades serão visitadas, indicando que Canoas nunca poderá ser visitada antes de Viamão.

O conjunto de vértices V foi o único conjunto utilizado na formulação do problema. Essa decisão foi feita pois o grafo G foi modelado como um grafo direcionado completo, em que as arestas inválidas possuem pesos logicamente infinitos (representado por valores inteiros muito altos). Dois parâmetros foram utilizados na modelagem do problema. O c_{ij} armazena o custo do arco entre os vértices i e j, sendo esse valor ≥ 0. O outro parâmetro, p_{ij}, associa se um vértice i precede o vértice j, isso é, se j somente pode ser visitado após já se ter passado pelo i.

Duas variáveis de decisão foram utilizadas na formulação apresentada. A primeira variável ($x_{i,j}$) pertence ao conjunto dos números binários e representa se o arco (i, j) está ou não presente na solução ótima do problema. Já a segunda variável (o_i) associa a cada vértice i a sua posição (a partir de 0) no caminho encontrado. A saber, para o primeiro vértice, o_i = 0; para o segundo o_{i+1} = 1; e assim por diante.

A função objetivo (1) minimiza o custo total, que é a soma dos custos associados a cada um dos arcos utilizados. A restrição (2) garante que na solução exista somente um arco sainte de cada vértice, exceto no último, para o qual a restrição (4) garante que não exista qualquer nó sainte. Similarmente, a restrição (3) garante que na solução exista somente um arco entrante em cada vértice, exceto no primeiro, para o qual a restrição (5) garante que não exista qualquer nó entrante.

As restrições (6) e (7) são utilizadas para garantir a ordem das visitas. A restrição (6) garante que a ordem de precedência dos vértices, fornecida na instância do problema, é obedecida. Já a restrição (7) serve para garantir a ordem de visitação. Por exemplo, se a aresta (i,j) é usada, então i obrigatoriamente deve ser visitado antes de j. Caso contrário, i não necessariamente precisa preceder j. Por fim, as restrições (8) e (9) definem o domínio das variáveis de decisão, explicados anteriormente.

3. Algoritmo proposto

O algoritmo utilizado consiste em, primeiramente, gerar uma solução inicial para a instância recebida. Em seguida, a meta-heurística Busca Tabu é utilizada para tentar otimizar (no caso, minimizar) o valor de função objetivo.

O principal objetivo da Busca Tabu é evitar que partes recentemente visitadas sejam retornadas, criando ciclos na solução. Para evitar esses ciclos, é utilizada uma lista Tabu para armazenar movimentos feitos até o presente momento. Essa lista serve como uma memória de curto prazo dos movimentos recentes dentro da área pesquisável.

As informações a respeito do grafo são armazenadas em uma matriz de adjacência de tamanho NxN, na qual cada elemento (i,j) armazena o custo associado ao arco entre os nós i e j. O valor armazenado também pode ser -1, no caso de o vértice j dever preceder o vértice i. Ademais, são utilizados vetores de inteiros para armazenar os caminhos e um vetor de pares ordenados para a lista tabu.

A solução factível inicial é gerada de modo guloso. Partindo-se do primeiro vértice, escolhe-se a aresta de menor custo cujo vértice conectado (i) ainda não foi visitado e (ii) todos vértices que o precedem já foram visitados. Esse controle de precedência é realizado por meio de um vetor onde são armazenados os vértices que devem esperar algum precedente ser visitado. Uma vez escolhido o segundo vértice do caminho, o processo é repetido, a partir desse vértices para se escolher o terceiro elemento. E assim por diante, até que o caminho alcance o vértice final.

Depois, utilizando esse solução factível inicial, o algoritmo começa a procura por um caminho ótimo. Para isso, uma vizinhança é selecionada através de uma busca local 2-opt. Essa busca troca o vértice destino de duas arestas do conjunto E entre dois vértices. Inicialmente, é calculado o custo 2-opt para cada combinação de vizinhos de um vértice i para um vértice j. Após, é escolhida a combinação de menor custo e realiza a troca (best improvement) Coloca na lista tabu o movimento realizado

O algoritmo implementado possui três parâmetros distintos: (i) tamanho da lista Tabu; (ii) probabilidade de aceitar soluções tabu (critério de aspiração) e (iii) diversificar o espaço de busca caso não encontre uma solução que reduza o custo. A fim de obter os melhores resultados do algoritmo, foi feita uma avaliação variando os três diferentes parâmetros para se atingir o melhor valor possível para cada um.

O primeiro parâmetro foi avaliado utilizando três valores diferentes: N/2, N/4 e N/8. O valor de N representa o número de vértices no grafo. Pôde-se perceber que quando o tamanho da lista tabu era muito grande (e.g., N/2), a quantidade de movimentos considerados válidos era reduzida, pois eles demoravam mais para serem removidos da lista. Já quando a lista suportava poucos elementos (e.g., N/8), a lista Tabu possuía pouca memória dos seus movimentos anteriores, potencialmente causando movimentos repetitivos e entrando em *loops*. Pôde-se observar que os resultados utilizando uma lista pequena eram próximos da solução inicial. O tamanho de N/4 apresentou o melhor resultado dentre as três opções. Esse tamanho está de acordo com os estudos de [Tsubakitani and Evans 1998].

O critério de aspiração possibilita que pares de vértices presentes na lista Tabu possam ser considerados válidos, com uma determinada probabilidade, pois a solução obtida com eles pode ser melhor que as com outros vértices. Para esse parâmetro, foram comparados três probabilidades distintas: 50%, 20% e 10%. Foi observado que ao utilizar uma alta probabilidade de aceitar movimentos presentes na lista Tabu (e.g., 50%), a solução final permanecia muito próximo da solução factível inicial, pois a busca Tabu perdia parte bastante significativa de seu efeito. Já ao utilizar uma probabilidade muito baixa, foi possível observar que a solução final também permanecia ruim e próxima da inicial, pois desse jeito não é permitido um maior espaço para exploração da instância, potencialmente bloqueando a solução em ótimos locais. A porcentagem de 20% se mostrou a mais adequada, pois, ao mesmo tempo que permite exploração, ela evita que a busca Tabu perca seu efeito.

O terceiro parâmetro visa diversificar o espaço de busca, considerando válidas trocas entre vértices mais distantes, quando nenhuma movimentação possível gera uma solução melhor. Isso é feito comparando-se a distância entre os vértices i e j a serem trocados. Para esse parâmetro, foram avaliados três valores: 3, 5 e 6. A primeira opção apresentou melhor efeito.

Tabela 1. Variação dos parâmetros

Parâmetro	Variações
Tamanho lista Tabu	N/2, **N/4**, N/8
Critério de aspiração	50%, **20**%, 10%
Diversificar espaço de busca	**3**, 5, 6

Dentre os critérios de parada que podem ser utilizados para algoritmo adotado são: número de iterações sem encontrar solução melhor, proximidade de algum limitante, número de iterações, ou até uma combinação de várias critérios. O critério adotado é o número de iterações. Após 10 iterações sem melhoria do custo, o algoritmo é encerrado.

4. Avaliação

A avaliação foi feita obtendo uma execução do método em programação linear (e.g. GLPK) e dez execuções para a meta-heurística. Para o primeiro método, foi utilizado como critério de parada a duração de uma hora do experimento. Para cada execução da meta-heurística, foi criada e utilizada uma semente de aleatoriedade diferente.

A tabela 2 apresenta os resultados obtidos com limite de 10 iterações sem melhoria da solução, enquanto a tabela 3 apresenta os mesmos dados para 50 iterações e com o

Tabela 2. Tabela de resultados - 10 iterações

Instância	1	2	3	4	5	6	7
ESC07	2125	0	2700	2550.0	0.0	0.00	20.00
ESC12	1675	1.2	2034	1751.0	0.0	0.01	4.53
ESC25	1681	129.6	3360	3360.0	0.0	0.08	99.88
ESC47	3152	3600	3843	3553.0	0.0	0.85	175.85
ESC78	–	3600	22600	22120.0	430.0	6.74	21.33
ft70.1	–	3600	45255	44459.0	0.0	3.14	13.08
prob.100	–	3600	2921	2755.0	0.0	10.84	136.88
rbg109a	–	3600	1443	1330.4	4.8	122.46	28.16
rbg150a	–	3600	2168	2079.8	1.8	417.97	18.84
rbg174a	–	3600	2444	2295.7	0.9	1009.22	12.92

Tabela 3. Tabela de resultados - 50 iterações

Instância	1	2	3	4	5	6	7
ESC07	2125	0	2700	2550.0	0.0	0.01	20.00
ESC12	1675	1.2	2034	1743.4	22.8	0.06	4.08
ESC25	1681	129.6	3360	3179.8	143.1	0.65	89.16
ESC47	3152	3600	3843	3553.0	0.0	3.61	175.85
ESC78	–	3600	22600	21812.0	519.0	27.82	19.64
ft70.1	–	3600	45255	44459.0	0.0	12.26	13.08
prob.100	–	3600	2921	2755.0	0.0	42.15	136.88
rbg109a	–	3600	–	–	–	-	–
rbg150a	–	3600	–	–	–	-	–
rbg174a	–	3600	–	–	–	-	–

tempo limitado a 2 minutos. As instâncias estão representadas nas linhas e as colunas representam os seguintes valores:

1. Valor da melhor solução encontrada pelo GLPK com a formulação matemática;
2. Tempo de execução do GLPK (com limite de 1h);
3. Valor médio da solução inicial do algoritmo;
4. Valor médio da melhor solução encontrada pelo algoritmo;
5. Desvio padrão das melhores soluções encontradas pelo algoritmo;
6. Tempo de execução médio (em segundos) do algoritmo;
7. Desvio percentual médio das soluções obtidas pelo algoritmo em relação à melhor solução conhecida.

4.1. Analise dos resultados obtidos

Analisando-se os resultados obtidos, pode-se perceber que o aumento do número de interações produz uma melhoria bastante significativa de algumas soluções. No caso da instância ESC25, ao executar o algoritmo com 50 iterações foi possível alcançar a melhor solução conhecida. Entretanto, à medida que se aumenta o número de iterações, o tempo aumenta de forma exponencial. Isso impossibilitou que, dentro do limite de 2 minutos, a execução do algoritmo encerrasse para as instâncias rbg109a, rbg150a e rbg174a.

Ademais, é possível observar que, para as quatro menores instâncias, o GLPK conseguiu encontrar soluções melhores que as obtidas com a meta-heurística. No entanto, para o restante das instâncias, não foi possível encontrar uma solução dentro de um limite de tempo de 1h.

5. Conclusões

A resolução de problemas usando programação linear inteira retorna soluções ótimas, porém esse método se mostra inadequado para instâncias grandes. Em comparação, a utilização de meta-heurísticas permite encontrar soluções de forma muito mais eficiente e rápida, por meio de estratégias que exploram outros fatores, como a aleatoriedade, por exemplo. As soluções obtidas por esse método nem sempre são as melhores possíveis, porém compensam muito pela rapidez.

A busca tabu mostrou-se eficiente e simples de se implementar. Para vários casos, a solução obtida foi satisfatória. No caso das instâncias maiores, pesou bastante em seu desempenho o fato de o algoritmo testar todas as possíveis trocas de vértice, de forma exaustiva. Explorar um pouco mais a aleatoriedade, nesse caso, parece ser uma estratégia mais interessante.

Referências

Balas, E., Fischetti, M., and Pulleyblank, W. R. (1995). The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68(1):241–265.

Letchford, A. N. and Salazar-González, J.-J. (2016). Stronger multi-commodity flow formulations of the (capacitated) sequential ordering problem. *European Journal of Operational Research*, 251(1):74 – 84.

Tsubakitani, S. and Evans, J. R. (1998). Optimizing tabu list size for the traveling salesman problem. *Comput. Oper. Res.*, 25(2):91–97.