Orienteering Problem utilizando Iterated Local Search (ILS)

Carolina Lange Mello e Marcely Zanon Boito

December 12, 2017

1 Introdução

Neste relatório apresentamos um problema da classe NP-completo chamado *Orienteering Problem*, primeiramente apresentado em 1987 [Golden et al. 1987]. Nós apresentaremos a sua formulação, a sua resolução utilizando o *solver* GLPK [GLP] e iremos comparar esse resultado com o de uma implementação própria da meta-heurística chamada *Iterated Local Search* (ILS).

2 Problema: The Orienteering Problem

O Orienteering Problem é um problema similar ao *The generalized traveling salesman problem* [Golden et al. 1987], onde possuímos a restrição do primeiro e último vértice da rota criada serem o mesmo, gerando um ciclo. Neste problema, é definido um grafo completo não direcionado de N vértices. Cada um destes vértices possui um valor, aqui chamado de *score*. De forma similiar, cada aresta que liga dois vértices possui também um valor, aqui chamado de *cost*. Por último, temos um valor de custo máximo relativo às arestas que não deve ser estrapolado durante a construção da rota. Assim, o objetivo deste problema é encontrar uma rota que saia e retorne a um vértice inicial pré-definido, maximizando o score (soma dos valores dos vértices) e não ultrapassando o custo máximo (soma das arestas).

2.1 Formulação Matemática

Através da definição do problema, é facilmente dedutível que se trata de um problema de maximização em programação inteira (já que não podemos ter arestas fracionárias). Assim, para a formulação deste problema, estamos considerando os seguintes parâmetros e variáveis:

- **Parâmetros:** C custo máximo;
  $D_{i,j}$ custo da aresta que liga o vértice i ao vértice j;
  P_v pontuação do vértice v.
- **Variáveis:** $X_{i,j} \in 0, 1$ indica se a aresta (i, j) está presente na solução);
  $V_n \in 0, 1$ indica se o vértice n está na solução encontrada;
  $U_n \in \mathbb{N}^+$ indica a ordem que um vértice v pertencente à solução é visitado.

Considerando as nossas variáveis, a nossa função de maximização está descrita em (1). Em seguida, apresentamos o conjunto de restrições utilizados. Todas as restrições consideram N como sendo o número de vértices no grafo recebido.

$$\max \sum_{n \in N} p_n \cdot V_n \tag{1}$$

1. Criação de um caminho que inicia e termina em v0:

$$\sum_{j \in N, j \neq v0} x_{vo,j} = 1 \tag{2}$$

$$\sum_{i \in N, v0 \neq i} x_{i,vo} = 1 \tag{3}$$

2. Um vértice só pode ser visitado uma única vez (4-5) e criação de um fluxo entre os vértices (6):

$$\sum i \in N, i \neq v x_{i,n} \leq 1, \forall n \in N \cap v0 \tag{4}$$

$$\sum_{j \in N, j \neq n} x_{v,j} \leq 1, \forall n \in N \cap v0 \tag{5}$$

$$\sum_{j \in N} x_{n,j} - \sum_{j \in N} x_{j,n} = 0, \forall n \in N \tag{6}$$

3. Respeito ao custo máximo C:

$$\sum_{i,j \in N} x_{i,j} \cdot d_{i,j} \leq C \tag{7}$$

4. Restrição MTZ (Miller-Tucker-Zemlin): Essa restrição cria uma ordem entre os vértices visitados, rejeitando soluções que possuam rotas desconexa [Pataki 2003]s.

$$u_1 = 1 \tag{8}$$

$$2 \leq u_i \leq N, \forall i \in N[i \neq 1] \tag{9}$$

$$u_i - u_j + 1 \leq (N-1) \cdot (1 - x_{i,j}), \forall i, j \in N[i \neq 1][j \neq 1] \tag{10}$$

3 Meta-Heurística: Iterated Local Search

O algoritmo *Iterated Local Search* (ILS) [Lourenço et al. 2003] consiste em, a partir de uma solução inicial (s0) que pode ser gerada de forma aleatória ou gulosa, gerar várias soluções obtidas por uma heurística de busca local que são perturbadas com o objetivo de buscar soluções melhores. Essas soluções podem ser aceitas ou não dependendo do critédio utilizado, e o algoritmo fica em *loop* até que atinja um critério de parada e devolva uma solução.

3.1 Algoritmo implementado

Visão geral do funcionamento do algoritmo:

- **Solução inicial:** Nossa solução inicial é gerada de forma **aleatória**. Ela inicia sempre em v0 (vértice inicial) e escolhe aleatóriamente outros vértices para adicionar na solução inicial.
- **Busca Local:** A Busca Local implementada tenta adicionar vértices na solução perturbada, escolhidos de forma aleatória e que respeitem as restrições do problema, até chegar no critério de parada. Se o nodo escolhido não pode ser adicionado ele é ignorado.
- **Perturbação utilizada:** A perturbação utilizada foi a remoção aleatória de até 20% dos vértices da melhor solução encontrada até o momento, os quais são escolhidos aleatoriamente.

- **Critério de aceitação:** O algoritmo implementado aceita soluções melhores em 90% das vezes, permitindo em 10% dos casos soluções piores, o que tenta evitar a permanência em um máximo local.
- **Critério de parada da busca local:** 30 iterações, controladas por uma variável *maxImprov*.
- **Critério de parada o algoritmo:** 50.000 iterações, controladas por uma variável *maxIt*.

3.2 Estruturas de Dados

O algoritmo foi implementado em Java, e embora ele possua várias classes, durante as iterações, a manipulação é feita utilizando apenas objetos da classe Solution. Esses objetos, entre outras variáveis e métodos, possuem uma lista simples de inteiros, o vetor de adjacência da solução gerada.

Escolhemos efetuar toda manipulação (consulta, adição e remoção) de vértices através desse vetor de inteiros por dois motivos: intuitividade e eficiência. Em anexo 6 ilustramos a nossa organização de classes, que não será apresentada aqui por falta de espaço.

4 Experimentos e Análise

Um dos objetivos desse trabalho é conseguir compreender a diferença de solvers exatos (como o GLPK) e meta-heurísticas para aproximações da solução ótima. Assim, na seção 4.1 nós apresentamos os resultados da formulação apresentada em 2.1 executada no solver GLPK. Em seguida, em 4.2 nós apresentamos os resultados obtidos com a implementação da meta-heurística escolhida (ILS). Para os nossos experimentos, nós utilizamos as mesmas 10 instâncias, onde, para cada uma delas, o número representa a quantidade de vértices presentes.

4.1 GLPK

A tabela 4.1 apresenta os nossos resultados. Para cada instância, foi dado o tempo máximo de uma hora para a conclusão do algoritmo. Como pode ser verificado na coluna mais a direita, apenas duas instâncias não estouraram o tempo definido. Apenas 4 instâncias conseguiram o retorno de um valor

diferente de zero, e em apenas metade desses casos o valor encontrado foi o ótimo.

instance	**BKV**	**GLPK**	**gap(%)**	**time (seconds)**	**timeout?**
a8	190	190	0	0.1	N
a16	245	245	0	33	N
ber25	609	560	8.04	3,600	S
bier127	2,365	0	100	3,600	S
eil51	1,399	1,277	8.72	3,600	S
eil76	2,467	0	100	3,600	S
kroA200	6,123	0	100	3,600	S
lin105	2,986	0	100	3,600	S
pr152	3,905	0	100	3,600	S
rat99	2,908	0	100	3,600	S
average			61.83	2,883.31	

Table 1: Resultados utilizando a formulação apresentada em 2.1 implementada para o GLPK.

4.2 ILS

Nossos resultados podem ser verificados na tabela 4.2. Nenhuma solução encontrada foi ótima, verificável observando a coluna de gap.

Nós utilizamos número de iterações como critério de parada (50.000), e o tempo resultante (última coluna) ser baixo tem duas razões: implementação eficiente das estruturas de dados e o fato do nosso algoritmo ficar preso em um máximo local. Foram testados, para as duas menores instâncias (a8 e a16), números de iterações maiores que 50.000 entretanto, os valores encontrados foram **exatamente iguais**, o que evidencia uma característica já conhecida sobre busca local: sua tendência de ficar "preso" em um máximo/mínimo local.

5 Análise e Conclusão

Analisando as duas tabelas de resultados, podemos evidenciar algumas características das duas diferentes técnicas de solução. O GLPK com limite de tempo é muito próximo do exato, retornando o valor ótimo para instâncias

instance	BKV	best	avg	gap (%)	time (seconds)
a8	190	180	136	28.42	0.11
a16	245	190	140	42.85	0.15
ber25	609	318	237	61.08	0.22
bier127	2,365	479	396	83.25	2.26
eil51	1,399	518	403	71.19	0.61
eil76	2,467	547	492	80.05	1.14
kroA200	6,123	643	556	90.91	3.88
lin105	2,986	717	473	84.15	2.09
pr152	3,905	745	479	87.73	2.88
rat99	2,908	627	517	82.22	1.28
average				71.19	1.46

Table 2: Resultados utilizando o algoritmo apresentado em 3.1. Cada instância foi executada 10 vezes, utilizando seeds de 1 a 10.

pequenas e aproximado para a instância com 51 vértices (eil51). Entretanto, para problemas maiores, dispondo de apenas uma hora, não foi possível encontrar nenhuma solução[1]. Isso evidencia a maior desvantagem de técnicas exatas: o tempo necessário para retornar valores para instâncias maiores de um problema.

Por outro lado, a meta-heurística, mesmo retornando vários valores consideravelmente longe da solução ótima, sempre retorna uma primeira aproximação da solução. Por causa disso, a média de distância (gap) não é muito maior que a apresentada pelo GLPK.

Após a realização desse trabalho, tendo implementado o ILS e avaliado seu resultado, nós julgamos que heurísticas de buscas locais não são as mais adequadas para esse tipo de problema. Vemos uma degradação grande na qualidade do resultado conforme aumentamos o número de vértices no nosso problema, e por causa disso acreditamos que outras heurísticas seriam mais aconselhadas.

Por fim, concluímos que o GLPK é uma ótima ferramenta para encontrar a solução ótima de problemas, mas deve ser usado com parcimônia. Quando temos instâncias grandes e/ou necessitamos apenas de uma aproximação, outras técnicas não tão exatas podem ser utilizadas.

[1]Por curiosidade, executamos a quarta menor instância (eil76) com o tempo de 4 horas, mas mesmo assim o GLPK não conseguiu retornar uma primeira aproximação da solução.

References

[GLP] Glpk: Gnu linear programming kit. `https://www.gnu.org/software/glpk/glpk.html#documentation`.

[Golden et al. 1987] Golden, B. L., Levy, L., and Vohra, R. (1987). The orienteering problem. *Naval research logistics*, 34(3):307–318.

[Lourenço et al. 2003] Lourenço, H. R., Martin, O. C., and Stutzle, T. (2003). Iterated local search. *International series in operations research and management science*, pages 321–354.

[Pataki 2003] Pataki, G. (2003). Teaching integer programming formulations using the traveling salesman problem. *SIAM review*, 45(1):116–123.

6 Anexo