UM ALGORITMO SIMPLES E EFICIENTE PARA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE GENERALIZADO

Jordi Alves Reinsma, Puca Huachi Vaz Penna, Marcone Jamilson Freitas Souza
Departamento de Computação - Universidade Federal de Ouro Preto (UFOP)
Campus Universitário, Morro do Cruzeiro, CEP 35400-000, Ouro Preto (MG), Brasil
jordi.reinsma@aluno.ufop.edu.br, puca@ufop.edu.br,
marcone@ufop.edu.br

RESUMO

Este trabalho tem seu foco no Problema do Caixeiro Viajante Generalizado, uma variante do Problema do Caixeiro Viajante clássico, onde as cidades são agrupadas em partições, sendo necessária a visita de apenas uma cidade em cada partição. Dada a dificuldade de resolvê-lo na otimalidade no caso geral, desenvolve-se neste trabalho um algoritmo híbrido, que combina as metaheurísticas *Multi-Start* e *Iterated Local Search* (ILS). O ILS é apoiado por duas buscas locais e um método de perturbação que aumenta, gradativamente, o nível de perturbação após um certo número de iterações sem melhora na solução corrente. Resultados demostram que o algoritmo é competitivo quando comparado com os melhores algoritmos da literatura.

PALAVRAS CHAVE. ***Problema do Caixeiro Viajante Generalizado, Iterated Local Search, Multi-Start.***

ABSTRACT

This paper deals the Generalized Traveling Salesman Problem, a variant of the classic Traveling Salesman Problem, where the cities are grouped in clusters, and it is necessary to visit only one city of each cluster. Given its difficulty in solving it optimally in the general case, a hybrid algorithm is developed on this study, combining the metaheuristics Multi-Start and Iterated Local Search (ILS). The ILS is supported by two local searches and one perturbation method, which increases its perturbation ratio after a fixed number of iterations, without improvements on the current solution. Results show the algorithm is competitive when compared with the best algorithms in the literature.

KEYWORDS. ***Generalized Traveling Salesman Problem, Iterated Local Search, Multi-Start.***

1. Introdução

O Problema do Caixeiro Viajante Generalizado (*Generalized Traveling Salesman Problem*, GTSP) é uma variação do clássico Problema do Caixeiro Viajante (*Traveling Salesman Problem*, TSP), ambos sendo problemas de minimização da distância de rotas cíclicas. A motivação para se estudar este problema está em sua dificuldade teórica e em sua relevância prática. Sua dificuldade encontra-se no fato de que o problema é da classe de complexidade $\mathcal{NP}$-Difícil - assim como o TSP, não possuindo, portanto, um algoritmo eficiente para sua resolução em tempo hábil. Sua relevância está na existência de diversas situações no mundo real cuja abstração é modelada como o GTSP. Exemplos de aplicações são: sequenciamento de visita a armazéns com múltiplos locais para estoque, sequenciamento de arquivos de computador, seleção de aeroportos, roteamento postal e design de redes em anel [Noon e Bean, 1991; Laporte et al., 1996].

Vários estudos foram feitos sobre o GTSP. Existem algoritmos exatos que o resolvem utilizando *branch-and-bound* e *branch-and-cut* [Fischetti et al., 1997], porém, devido à complexidade do problema, tais algoritmos possuem tempo de execução não muito eficientes ao analisar instâncias grandes, perdendo a praticidade.

Para contornar esta dificuldade, pesquisadores desenvolveram transformações do GTSP para o TSP [Ben-Arieh et al., 2003], cuja literatura sobre métodos exatos e heurísticas eficientes é vasta. Tais transformações conseguem produzir soluções ótimas para ambos os problemas, porém soluções sub-ótimas do TSP não preservam a sub-otimalidade para o GTSP. Este fato cria a necessidade da criação de heurísticas próprias para o GTSP.

Gutin e Karapetyan [2010] produziram um algoritmo memético para o GTSP, que combina a diversificação da solução do algoritmo genético com a intensificação da vizinhança proporcionada por buscas locais. Com enfoque maior na utilização de buscas locais diversificadas, este algoritmo produziu as melhores soluções encontradas atualmente. Seus resultados serão comparados com o algoritmo produzido neste trabalho para medição de performance. Jafarzadeh et al. [2017] desenvolveram um algoritmo memético com um único método de busca local, o método do vizinho mais próximo (*Nearest Neighbor*), reproduzindo as melhores soluções conhecidas, com menor tempo de execução para todas as instâncias analisadas.

Algoritmos populacionais são frequentemente usados para solucionar o TSP e suas variações. Para o GTSP, autores também utilizaram o sistema de colônia de formigas [Pintea et al., 2017; Jun-man e Yi, 2012] e a otimização por enxame de partículas [Shi et al., 2007]. Ambos os algoritmos produzem tempos de execução relativamente altos e com menor número de soluções ótimas encontradas em comparação a algoritmos meméticos.

Com a popularidade e eficiência de algoritmos populacionais já reconhecida, como alternativa neste trabalho é desenvolvido um algoritmo que combina as metaheurísticas *Multi-Start* (MS) e *Iterated Local Search* (ILS) para resolver o GTSP, tal como utilizado em Penna et al. [2013]. O ILS é acompanhado por duas buscas locais simples e um único método de perturbação da solução, com diferentes níveis de intensidade. Além disso, o algoritmo proposto somente amplifica a perturbação após uma certa quantidade de iterações sem melhora na solução corrente, permitindo que haja uma melhor exploração na vizinhança da solução corrente antes de se afastar dessa região. Os resultados obtidos nos testes realizados validam a aplicação do algoritmo proposto, ao serem produzidas soluções de excelente qualidade em baixo tempo de processamento.

O restante deste trabalho está organizado como segue. A Seção 2 apresenta uma caracterização formal do problema e sua relação com o TSP. A Seção 3 descreve, em detalhes, o algoritmo proposto. Na Seção 4, os resultados computacionais são detalhados e discutidos. Por fim, na Seção 5, são apresentadas as conclusões e perspectivas de trabalhos futuros.

2. Descrição do problema

Considere um grafo ponderado direcionado ou não-direcionado G possuindo n vértices e um conjunto de partições V_1, V_2, ... , V_{m-1}, V_m, $m \leq n$, de seus vértices, chamados de *clusters*. O Problema do Caixeiro Viajante Generalizado tem como objetivo encontrar o ciclo de peso mínimo que contém um único vértice de cada *cluster*. O problema é $\mathcal{NP}$-Difícil, considerando que o TSP é uma versão especial do GTSP, onde $|V_i| = 1$ para todo i, e, consequentemente, $m = n$.

Um exemplo de tal grafo encontra-se na Figura 1. Considere que o grafo é completo, suas arestas foram removidas para melhor visualização. Este grafo possui 10 vértices, numerados de 1 a 10, e 4 *clusters*, nomeados A, B, C e D.

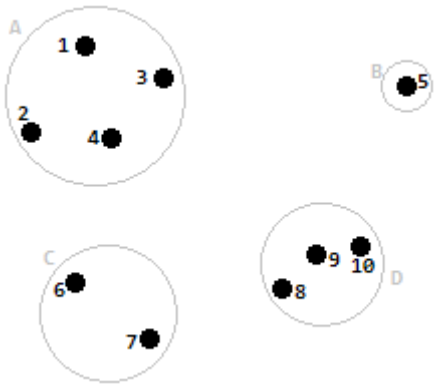

Figura 1: Exemplo de um grafo possuindo 4 partições e 10 vértices

Uma solução viável S do GTSP para este grafo é a escolha de um vértice qualquer de cada *cluster*, como mostra a Figura 2. O valor da solução é dada pela soma do peso das arestas que ligam os vértices.

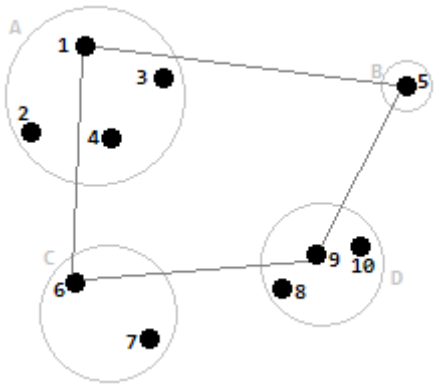

Figura 2: Exemplo de solução para o GTSP

3. Métodos desenvolvidos

Nesta seção são apresentadas as representações computacionais utilizadas neste trabalho para a resolução do problema, assim como os algoritmos desenvolvidos que aplicam tal representação para percorrer o espaço de soluções.

3.1. Representação da Instância

São utilizadas duas representações de instâncias para o GTSP. A primeira usa um vetor de listas que guarda a informação dos vértices contidos em cada *cluster*. Cada posição do vetor é um *cluster*, como mostra a Tabela 1, exemplificado pelo grafo apresentado na Figura 1 da Seção 2. A segunda representação é o inverso da primeira, ou seja, um vetor que guarda a informação sobre

Cluster	Vértices
A	1, 2, 3, 4
B	5
C	6, 7
D	8, 9, 10

Tabela 1: Vetor de *clusters*

Vértice	Cluster
1	A
2	A
3	A
4	A
5	B
6	C
7	C
8	D
9	D
10	D

Tabela 2: Vetor de vértices

qual *cluster* cada vértice é pertencente, demonstrado na Tabela 2. Estes dois vetores são igualmente úteis pois faz-se necessária a rápida busca por todos os vértices pertencentes a um *cluster* - no caso da primeira representação - e também a busca pelo *cluster* que contém um específico vértice, através da segunda. Ambos os modelos são utilizados nos métodos descritos nas próximas subseções. Para armazenar o peso das arestas do grafo, utiliza-se uma matriz de distâncias entre vértices, onde a célula da linha i com a coluna j representa o peso da aresta que liga o vértice i ao vértice j.

3.2. Representação da Solução

A solução possui representação semelhante ao TSP, representada por um vetor de vértices, em que a posição no vetor indica a ordem de visita aos vértices. Posições adjacentes, portanto, representam arestas utilizadas na solução, adicionando a aresta que liga o vértice da última posição do vetor ao vértice da primeira posição, concluindo a rota cíclica. A diferença para com o TSP está no tamanho do vetor. No GTSP, o tamanho da rota é determinada pelo número total de *clusters* da instância, onde só é permitido existir um único vértice de cada *cluster* ao mesmo tempo no vetor. A Tabela 3 mostra a representação da solução do grafo do exemplo da Figura 2 da Seção 2. É válido destacar que $[5, 6, 9, 1]$ e $[6, 9, 1, 5]$ são exemplos de permutações do vetor que representam a mesma solução, pois o formato cíclico do problema permite a escolha arbitrária do vértice a ocupar a primeira posição do vetor.

Ordem de visita	1	2	3	4
Vértice	1	5	6	9

Tabela 3: Vetor representativo da solução do GTSP

3.3. Avaliação da Solução

Por se tratar de um problema de minimização da distância de rotas, a avaliação de uma solução para o GTSP é caracterizada pela soma do peso das arestas que conectam os vértices componentes do vetor solução.

3.4. Construção da Solução Inicial

Como forma de produzir uma solução inicial, a heurística do vizinho mais próximo (*Nearest Neighbor*, NN) para o TSP é implementada, com modificações para tornar-se adequada ao problema generalizado. O algoritmo funciona como se segue: um vértice qualquer é escolhido aleatoriamente para ser o primeiro vértice a compor a solução, anotando o *cluster* a qual pertence como visitado. Em seguida, são analisados todos os vértices v pertencentes a *clusters* não visitados contidos na solução até então. A análise é dada pela soma do peso da aresta que liga o vértice v ao vértice inicial da rota, somado ao peso da aresta que liga v ao último vértice inserido na rota. Então,

o vértice que produz a soma com valor mínimo é escolhido para fazer parte da solução (ou seja, o vizinho mais próximo), sendo posicionado logo em seguida do último vértice adicionado. Esta análise de vértices é repetida até que todos os *clusters* tenham sido visitados. Esta heurística produz soluções consideravelmente melhores em relação à geração aleatória, com contrapeso de possuir complexidade de tempo $\Theta(nm)$, sendo n o número total de vértices do grafo e m a quantidade de *clusters*.

3.5. Buscas Locais

A partir da solução inicial construída pelo método NN, descrito na Seção 3.4, heurísticas de refinamento da solução são necessárias, a fim de procurar por ótimos locais, dado o fato que o ótimo global também é um ótimo local. Neste trabalho são utilizadas duas estruturas de vizinhança: a reordenação da rota através da troca de duas arestas – *2-Opt Swap*, e a substituição do vértice representativo do *cluster* na rota – *Cluster Swap*.

A primeira busca local implementada é a *Cluster Improvement*, método que utiliza a vizinhança do *Cluster Swap*. Neste, um vértice v incluso na solução é escolhido, encontrando o *cluster* c do qual pertence. Então, é calculada a distância de todos os vértices v_i inclusos em c para os vértices adjacentes a v no vetor solução. Considerando tais informações, o vértice v_i que produz a menor distância entre todos substitui o vértice v na solução. Este processo é realizado para cada vértice da solução, em ordem aleatória. Como a análise é feita considerando todos os vértices de todos os *clusters* uma única vez, a complexidade de tempo deste algoritmo é linear em relação ao número de vértices do grafo.

O segundo método produzido é resultado da aplicação da clássica heurística *2-Opt* para resolução do TSP. Para simular a troca de duas arestas da estrutura de vizinhança *2-Opt Swap*, são escolhidos 4 vértices do vetor da solução s: v_i, v_{i+1}, v_j e v_{j+1}. v_i e v_{i+1} são vértices adjacentes, assim como v_j e v_{j+1}. A vizinhança então é representada pela troca da posição dos vértices v_{i+1} e v_j na solução, e a inversão do sentido de todos os vértices entre v_{i+1} e v_j. Todas as $m \times (m-3)$ possibilidades de troca de posição são avaliadas, considerando m o tamanho do vetor solução. A cada troca, é calculada o valor da solução e, caso a troca produza uma rota de menor distância que a anterior, a troca é aceita, caso contrário, a troca é desfeita. O algoritmo reinicia todo o processo buscando melhorar a solução em uma de suas iterações, encerrando caso nenhuma melhoria seja encontrada.

3.6. *Multi-Start Smart Iterated Local Search*

Para guiar a utilização das buscas locais e evitar a parada prematura em ótimos locais, é aplicada uma combinação das metaheurísticas *Multi-Start* – MS [Martí et al., 2013] e Iterated Local Search – ILS [Lourenço et al., 2003]. A combinação dessas duas metaheurísticas, nomeado MS-ILS, tem produzido bons resultados, como pode ser visto nos estudos de Penna et al. [2013] e Nguyen et al. [2012]. Este trabalho, por sua vez, considera uma variação do ILS tradicional. Nessa técnica, nomeada MS-SILS, das iniciais em inglês de *Multi-Start Smart Iterated Local Search*, o SILS faz o papel de método de busca local da metaheurística MS. Este método é explicado abaixo.

Um aspecto importante do ILS é a utilização adequada de parâmetros para gerar perturbações que evitem a busca ficar presa em ótimos locais sem que seja perdida a qualidade das soluções. Produzir um equilíbrio entre tais aspectos é a especialidade da metaheurística, mas também seu aspecto mais difícil de lidar. Como forma de aumentar as chances de encontrar o nível de perturbação mais equilibrado, utiliza-se uma perturbação dinâmica, que somente é aumentada quando a busca não consegue gerar uma solução melhor após um certo número de iterações sem melhora, tal como em Souza et al. [2010], que utiliza o *General Variable Neighborhood Search* – GVNS [Hansen et al., 2010]. Por outro lado, sempre que uma nova solução melhor é gerada, a perturbação volta ao seu nível mais baixo. O método ILS com essas alterações é chamado de *Smart Iterated Local Search* (SILS).

Como método de perturbação da solução utilizado pelo SILS, é aplicada a técnica *Random Shuffle*. Considere p uma porcentagem referente à quantidade de vértices da solução que sofrem a perturbação. O algoritmo escolhe aleatoriamente uma posição $inicio$ do vetor da solução s para ser a posição inicial da perturbação, e produz a posição fim, calculada por $inicio + \lceil p \times m \rceil$. Entre $inicio$ e fim, todos os vértices da solução são trocados para posições aleatórias entre os dois valores, gerando a solução perturbada.

O Algoritmo 1 mostra o pseudocódigo do método MS-SILS proposto.

Algoritmo 1: *Multi-Start Smart Iterated Local Search* (MS-SILS)

```
1  Entrada: iter_maxMS, p_ini, p_incr, num_incr, iter_maxSILS
2  f(s*) ← ∞;
3  iter ← 0;
4  while iter < iter_maxMS do
5      iter ← iter + 1;
6      s ← NearestNeighbor();
7      s' ← ClusterImprovement(s);
8      s' ← 2-Opt(s');
9      s' ← ClusterImprovement(s');
10     s'' ← SILS(s', p_ini, p_incr, num_incr, iter_maxSILS);
11     if f(s'') < f(s*) then
12         s* ← s'';
13 Saída: s*
```

O Algoritmo 1 começa com a inicialização da melhor solução atual na linha 2, uma solução artificial de função objetivo infinitamente grande, para rápida substituição para uma solução válida adquirida pelos métodos aplicados em seguida. Este *Multi-Start* é caracterizado por execuções sequenciais de uma construção de uma nova solução independente (linha 6), buscas locais (linhas 7 a 9) e o método SILS (linha 10) de fuga de ótimos locais. Por fim, a melhor solução encontrada até então é armazenada. Este processo se repete $iter_{\max MS}$ vezes.

O Algoritmo 2 mostra o pseudocódigo do método SILS, linha 10 do Algoritmo 1. Esse algoritmo inicia na linha 2 com o nível de perturbação p fixado no valor dado por p_{ini}. Em seguida, é calculado na linha 3 o valor máximo de perturbação, $p_{\max}$. A seguir, entre as linhas 5 e 17, o algoritmo entra em um laço em que, a cada nível de perturbação p, tentar-se-á por até $iter_{\max}$ iterações, melhorar a solução corrente. Para tanto, é gerada uma solução s' vizinha da solução corrente s pela aplicação do procedimento *RandomShuffle*(s, p) descrito no início desta Seção. Essa solução vizinha s' é, então, submetida a uma sequência de três buscas locais consecutivas, sendo a primeira e terceira a *ClusterImprovement* e a segunda, a *2-Opt*. O resultado da aplicação dessas buscas locais é uma solução s'', que é avaliada na linha 12. Caso ela represente uma melhora em relação à solução corrente, então ela passa a ser a nova solução corrente (linha 13), o nível de perturbação volta ao seu nível mais baixo (linha 14) e o número de iterações em um mesmo nível de perturbação retorna ao seu nível mais baixo (linha 15). Após $iter_{\max SILS}$ iterações sem sucesso em um mesmo nível de perturbação, a perturbação é incrementada na linha 16 e o contador do número de iterações em um mesmo nível de perturbação volta ao seu valor mais baixo na linha 17.

Algoritmo 2: *Smart Iterated Local Search (SILS)*

Entrada: $s, p_{ini}, p_{incr}, num_{incr}, iter_{\max SILS}$
$p \leftarrow p_{ini}$;
$p_{max} \leftarrow p + (p_{incr} \times num_{incr})$;
$iter \leftarrow 0$;
while $p \leq p_{\max}$ **do**
    while $iter < iter_{\max SILS}$ **do**
        $iter \leftarrow iter + 1$;
        $s' \leftarrow$ *RandomShuffle*(s, p);
        $s'' \leftarrow$ *ClusterImprovement*(s');
        $s'' \leftarrow$ *2-Opt*(s'');
        $s'' \leftarrow$ *ClusterImprovement*(s'');
        if $f(s'') < f(s)$ **then**
            $s \leftarrow s''$;
            $p \leftarrow p_{ini}$;
            $iter \leftarrow 0$;
    $p \leftarrow p + p_{incr}$;
    $iter \leftarrow 0$;
Saída: s

4. Resultados computacionais

Para a análise do algoritmo desenvolvido, nomeado MS-SILS, foram utilizadas as instâncias disponibilizadas por Karapetyan [2012], possuindo também as melhores soluções de todas as instâncias. Esta biblioteca possui instâncias de até 217 *clusters* e 1084 vértices, misturando instâncias simétricas e assimétricas e instâncias que respeitam ou não a desigualdade triangular em relação ao peso das arestas. Este trabalho manteve o foco em resolução das instâncias simétricas.

O algoritmo foi desenvolvido na linguagem C++ e os experimentos executados em um computador com processador *Intel Core i7-3520M* 2.90GHz, com 16GB de memória RAM e sistema operacional *Windows 10*. Os parâmetros do MS-SILS utilizados para todos os testes foram: $iter_{\max MS} = 50$, $p_{ini} = 10\%$, $p_{incr} = 5\%$, $num_{incr} = 3$, $iter_{\max SILS} = 150$, valores considerados os melhores por meio de uma análise empírica numa bateria preliminar de testes.

Os resultados dos testes são apresentados nas Tabelas 4 e 5. Nessas tabelas, a primeira coluna indica a instância e a segunda, o valor da melhor solução da literatura, dada pelo algoritmo GK [Gutin e Karapetyan, 2010], executado em um computador com processador *AMD Athlon 64 X2* 3.0GHz. A terceira e quarta colunas mostram, respectivamente, o melhor valor e o valor médio encontrados em 10 execuções do algoritmo MS-SILS. A quinta e sexta colunas indicam o *Gap* da distância relativa da solução do MS-SILS para com o GK, obtido pelas equações (1) e (2):

$$Gap^{\star} = \frac{\text{MS-SILS}^{\star} - GK}{GK} \times 100 \tag{1}$$

$$Gap_m = \frac{\text{MS-SILS}_m - GK}{GK} \times 100 \tag{2}$$

Tabela 4: Resultados computacionais: qualidade das soluções

Instância	**GK**	**MS-SILS***	**MS-SILS$_m$**	**Gap* (%)**	**Gap$_m$ (%)**
3burma14	1805	1805	1805,00	0,00	0,00
4br17	31	31	31,00	0,00	0,00
4gr17	1309	1309	1309,00	0,00	0,00
4ulysses16	4539	4539	4539,00	0,00	0,00
5gr21	1740	1740	1740,00	0,00	0,00
5gr24	334	334	334,00	0,00	0,00
5ulysses22	5307	5307	5307,00	0,00	0,00
6bayg29	707	707	707,00	0,00	0,00
6bays29	822	822	822,00	0,00	0,00
6fri26	481	481	481,00	0,00	0,00
9dantzig42	417	417	417,00	0,00	0,00
10att48	5394	5394	5394,00	0,00	0,00
10gr48	1834	1834	1834,00	0,00	0,00
10hk48	6386	6386	6386,00	0,00	0,00
11berlin52	4040	4040	4040,00	0,00	0,00
11eil51	174	174	174,00	0,00	0,00
12brazil58	15332	15332	15332,00	0,00	0,00
14st70	316	316	316,00	0,00	0,00
16eil76	209	209	209,00	0,00	0,00
16pr76	64925	64925	64925,00	0,00	0,00
20gr96	29440	29440	29440,00	0,00	0,00
20kroA100	9711	9711	9711,00	0,00	0,00
20kroB100	10328	10328	10328,00	0,00	0,00
20kroC100	9554	9554	9554,00	0,00	0,00
20kroD100	9450	9450	9450,00	0,00	0,00
20kroE100	9523	9523	9523,00	0,00	0,00
20rat99	497	497	497,00	0,00	0,00
20rd100	3650	3650	3650,00	0,00	0,00
21eil101	249	249	249,00	0,00	0,00
21lin105	8213	8213	8213,00	0,00	0,00
22pr107	27898	27898	27898,00	0,00	0,00
24gr120	2769	2769	2769,00	0,00	0,00
25pr124	36605	36605	36605,00	0,00	0,00
26bier127	72418	72418	72418,00	0,00	0,00
26ch130	2828	2828	2828,00	0,00	0,00
28gr137	36417	36417	36417,00	0,00	0,00
28pr136	42570	42570	42570,00	0,00	0,00
29pr144	45886	45886	45886,00	0,00	0,00
30ch150	2750	2750	2750,00	0,00	0,00
30kroA150	11018	11018	11018,00	0,00	0,00
30kroB150	12196	12196	12196,00	0,00	0,00
31pr152	51576	51576	51576,00	0,00	0,00
32u159	22664	22664	22664,00	0,00	0,00
35si175	5564	5564	5564,00	0,00	0,00
36brg180	4420	4420	4420,00	0,00	0,00
39rat195	854	854	854,00	0,00	0,00

Tabela 5: Resultados computacionais: qualidade das soluções (cont.)

Instância	**GK**	**MS-SILS***	**MS-SILS$_m$**	**Gap* (%)**	**Gap$_m$ (%)**
40d198	10557	10557	10557,00	0,00	0,00
40kroA200	13406	13406	13406,00	0,00	0,00
40kroB200	13111	13111	13111,00	0,00	0,00
41gr202	23301	23301	23301,00	0,00	0,00
45ts225	68340	68340	68340,00	0,00	0,00
45tsp225	1612	1612	1612,00	0,00	0,00
46gr229	71972	71972	71972,00	0,00	0,00
46pr226	64007	64007	64007,00	0,00	0,00
53gil262	1013	1013	1013,00	0,00	0,00
53pr264	29549	29549	29549,00	0,00	0,00
56a280	1079	1079	1079,00	0,00	0,00
60pr299	22615	22615	22615,00	0,00	0,00
64lin318	20765	20765	20765,00	0,00	0,00
80rd400	6361	6361	6394,80	0,00	0,53
84fl417	9651	9651	9651,00	0,00	0,00
87gr431	101946	101946	101947,20	0,00	0,00
88pr439	60099	60099	60099,00	0,00	0,00
89pcb442	21657	21657	21670,60	0,00	0,06
99d493	20023	20046	20084,70	0,11	0,31
107ali535	128639	128639	128691,00	0,00	0,04
107att532	13464	13471	13478,40	0,05	0,11
107si535	13502	13502	13502,10	0,00	0,00
113pa561	1038	1039	1045,70	0,10	0,74
115rat575	2388	2420	2437,80	1,34	2,09
115u574	16689	16692	16754,50	0,02	0,39
131p654	27428	27428	27428,00	0,00	0,00
132d657	22498	22556	22667,20	0,26	0,75
134gr666	163028	163728	164394,90	0,43	0,84
145u724	17272	17449	17560,00	1,02	1,67
157rat783	3262	3321	3341,00	1,81	2,42
200dsj1000	9187884	9311640	9352324,30	1,35	1,79
201pr1002	114311	115464	115954,20	1,01	1,44
207si1032	22306	22433	22459,20	0,57	0,69
212u1060	106007	107317	107925,10	1,24	1,81
217vm1084	130704	131825	132349,30	0,86	1,26
Média				0,13	0,21

Os resultados obtidos foram bastante competitivos, ou seja, resultados idênticos aos melhores valores da literatura foram obtidos em 83% das instâncias, equivalente a 67 instâncias. Nas 14 instâncias restantes, o *Gap* é relativamente pequeno, com a média do $Gap^{\star}$ considerando apenas estas instâncias resultando em 0,73%, e 1,16% para a média do Gap_m.

A Tabela 6 mostra o tempo médio de dez execuções do algoritmo MS-SILS e do algoritmo referência, GK, para instâncias de 40 ou mais *clusters*. Como os algoritmos foram executados em máquinas com diferentes configurações, não é possível fazer uma comparação justa dos tempos computacionais dos mesmos. Contudo, pode-se verificar que o MS-SILS obteve um tempo computacional superior ao GK. Mesmo assim, o MS-SILS possui um tempo computacional competitivo. Por exemplo, na maior instância, contendo 1084 vértices e 217 *clusters*, uma única iteração do SILS necessita pouco mais de 2 segundos para finalizar, o que permite a utilização do *Multi-Start* sem tornar o algoritmo pouco responsivo e lento. A velocidade do SILS é justificada pela baixa complexidade dos métodos implementados e pela rápida convergência para soluções próximas do ótimo, guiadas pela perturbação dinâmica.

Tabela 6: Resultados computacionais: tempos médios de execução

Instância	**GK**	**MS-SILS**	**Instância**	**GK**	**MS-SILS**
40d198	0,14	1,66	40kroA200	0,14	1,58
40kroB200	0,16	1,87	41gr202	0,21	1,81
45ts225	0,24	1,99	45tsp225	0,19	1,90
46pr226	0,10	1,31	46gr229	0,25	2,35
53gil262	0,31	3,12	53pr264	0,24	2,65
56a280	0,38	3,78	60pr299	0,42	4,28
64lin318	0,45	4,62	80rd400	1,07	9,09
84fl417	0,73	7,73	87gr431	2,01	11,30
88pr439	1,48	11,67	89pcb442	1,72	11,14
99d493	4,17	14,79	107ali535	5,82	17,38
107att532	3,45	17,74	107si535	1,88	15,78
113pa561	3,22	18,50	115u574	3,76	21,87
115rat575	4,12	19,64	131p654	2,82	21,55
132d657	6,82	29,72	134gr666	14,46	32,39
145u724	11,61	38,50	157rat783	15,30	44,24
200dsj1000	50,14	86,77	201pr1002	34,83	87,13
207si1032	36,76	79,63	212u1060	44,76	102,81
217vm1084	59,82	101,48	**Média**	8,97	23,82

5. Conclusões

Este trabalho teve seu foco no Problema do Caixeiro Viajante Generalizado. Para resolvê-lo, foi proposto um algoritmo, nomeado MS-SILS, que combina as metaheurísticas *Multi-Start* (MS) e *Iterated Local Search* (ILS). O algoritmo MS-SILS usa dois métodos de busca local, baseados nas vizinhanças de reordenação da rota por meio da troca de duas arestas (*2-Opt Swap*) e a substituição do vértice representativo do *cluster* na rota (*Cluster Swap*).

O algoritmo proposto foi testado em instâncias clássicas da literatura e seu desempenho foi comparado a um dos melhores algoritmos da literatura, o GK. Os resultados mostraram que o MS-SILS é competitivo com o GK. Obtendo um *Gap* médio entre as melhores soluções do MS-SILS e as soluções do GK de apenas 0,13%.

Como trabalhos futuros propõe-se usar a ferramenta *IRACE* [López-Ibáñez e Cáceres, 2017] para melhor calibrar os parâmetros do algoritmo, testar a inclusão de novas buscas locais mais refinadas e diferentes métodos de perturbação, e também a inclusão da análise de outras instâncias menos populares da literatura.

Agradecimentos

Os autores agradecem à Universidade Federal de Ouro Preto (UFOP), Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) o apoio ao desenvolvimento deste trabalho.

Referências

Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., e Zverovitch, A. (2003). Transformations of generalized ATSP into ATSP. *Operations Research Letters*, 31(5):357–365.

Fischetti, M., Salazar González, J. J., e Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394.

Gutin, G. e Karapetyan, D. (2010). A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60.

Hansen, P., Mladenović, N., e Pérez, J. A. M. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407.

Jafarzadeh, H., Moradinasab, N., e Elyasi, M. (2017). An enhanced genetic algorithm for the generalized traveling salesman problem. *Engineering, Technology & Applied Science Research*, 7(6):2260–2265.

Jun-man, K. e Yi, Z. (2012). Application of an improved ant colony optimization on generalized traveling salesman problem. *Energy Procedia*, 17:319–325.

Karapetyan, D. (2012). GTSP instances library. URL `http://www.cs.nott.ac.uk/~pszdk/gtsp.html`.

Laporte, G., Asef-Vaziri, A., e Sriskandarajah, C. (1996). Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, 47(12):1461–1467.

Lourenço, H., Martin, O., e Stützle, T. (2003). *Handbook of metaheuristics*, chapter Iterated local search, p. 321–353. Kluwer, Dordrecht.

López-Ibáñez, M. e Cáceres, L. P. (2017). The irace package: Iterated race for automatic algorithm configuration. URL `http://iridia.ulb.ac.be/irace/`.

Martí, R., Resende, M. G., e Ribeiro, C. C. (2013). Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, 226(1):1–8.

Nguyen, V.-P., Prins, C., e Prodhon, C. (2012). A multi-start iterated local search with tabu list and path relinking for the two-echelon location-routing problem. *Engineering Applications of Artificial Intelligence*, 25(1):56–71.

Noon, C. E. e Bean, J. C. (1991). A lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, 39(4):623–632.

Penna, P. H. V., Subramanian, A., e Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19(2):201–232.

Pintea, C.-M., Pop, P. C., e Chira, C. (2017). The generalized traveling salesman problem solved with ant algorithms. *Complex Adaptive Systems Modeling*, 5(1):8. ISSN 2194-3206.

Shi, X. H., Liang, Y. C., Lee, H. P., Lu, C., e Wang, Q. (2007). Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, 103(5):169–176.

Souza, M. J. F., Coelho, I. M., Ribas, S., Santos, H. G., e Merschmann, L. H. C. (2010). A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, 207:1041–1051.