UMA HEURÍSTICA MULTI-START LOCAL SEARCH PARA O PROBLEMA DO CAIXEIRO VIAJANTE MÚLTIPLO COM SELEÇÃO DE HOTÉIS

Marques Moreira de Sousa
Instituto de Computação - Universidade Federal Fluminense
Avenida General Milton Tavares de Souza, s/n - Boa Viagem - CEP 24210-330, Niterói (RJ), Brasil
msousa@ic.uff.br

Luiz Satoru Ochi
Instituto de Computação - Universidade Federal Fluminense
Avenida General Milton Tavares de Souza, s/n - Boa Viagem - CEP 24210-330, Niterói (RJ), Brasil
satoru@ic.uff.br

Simone de Lima Martins
Instituto de Computação - Universidade Federal Fluminense
Avenida General Milton Tavares de Souza, s/n - Boa Viagem - CEP 24210-330, Niterói (RJ), Brasil
simone@ic.uff.br

RESUMO

Este trabalho aborda o problema do caixeiro viajante múltiplo com seleção de hotéis (PCVM-SH), uma variante do problema do caixeiro viajante múltiplo e do problema do caixeiro viajante com seleção de hotéis. No PCVM-SH, é necessário definir rotas para os caixeiros que precisam visitar um número pré-definido de clientes em diferentes localidades, considerando que cada jornada de trabalho possui um tempo limite para ser realizada. Por esta razão, caixeiros podem necessitar de mais de uma jornada de trabalho para concluir as visitas. Se o tempo limite for atingido, um hotel deve ser utilizado para aguardar até o início da próxima jornada. O objetivo do PCVM-SH é minimizar o tempo total viajado por todos os caixeiros. É apresentado uma busca local *multi-start* para resolver este problema. Os resultados mostram que a heurística proposta é capaz de encontrar soluções qualitativamente próximas aquelas encontradas na literatura em um baixo custo computacional.

PALAVRAS CHAVE. Problema do Caixeiro Viajante Múltiplo com Seleção de Hotéis, Metaheurísticas, Busca Local Multi-Start.
Tópicos (Metaheurísticas, Otimização Combinatória, Logística e Transportes)

ABSTRACT

This paper deals with the multiple traveling salesperson problem with hotel selection (MTSPHS) which generalizes the well-known and classical multiple traveling salesperson problem and the traveling salesperson problem with hotel selection. The MTSPHS consists in determining a route for some salesperson who needs to visit a predefined number of customers at different locations (cities) considering that that each work day is time limited. For this reason, the salespersons may demand more than a single day to perform its demand. If the time limit is achieved, a hotel must be selected from the set of available for the salesperson to spend the night there. The goal is to minimize the total travel time. We present a Multi-Start Local Search to solve the MTSPHS. The results show that the proposed heuristic is able to find solutions close to those found in literature in a small computational time.

KEYWORDS. Multiple Traveling Salesperson Problem with Hotel Selection. Metaheuristics. Multi-Start Local Search.
Paper topics (Metaheuristics, Combinatorial Optimization, Logistics and Transports)

1. Introdução

Problemas em que é necessário realizar a definição de rotas, vêm recebendo atenção considerável por pesquisadores de todo o mundo, principalmente nas últimas décadas. Estes problemas estão relacionados com a definição do melhor caminho a se seguir para visitar um conjunto pré definido de locais, de forma a gastar o mínimo possível de recursos.

Devido a complexidade inerente destes problemas e suas possíveis aplicações no mundo real, uma enorme quantidade de variantes e métodos de resolução foram propostos ao longo das últimas décadas. Logo, o estudo de problemas de roteamento se tornou uma das mais importantes áreas no campo da Pesquisa Operacional. Dentre esses, está o problema do Caixeiro Viajante, um problema clássico de otimização combinatória (OC).

No Problema do Caixeiro Viajante (PCV) o objetivo é definir um ciclo hamiltoniano de custo mínimo em um grafo cujas arestas possuem um peso associado. O PCV é conhecidamente tido como um problema $\mathcal{NP}$-Difícil. Desde a definição deste problema, diversas outras variantes surgiram devido sua alta aplicabilidade a problemas práticos. São algumas delas: Problema do Caixeiro Viajante Múltiplo (PCVM) [Bektas, 2006], O PCV com Seleção de Hotéis (PCVSH) [Vansteenwegen et al., 2012; Castro et al., 2013, 2014a; Sousa et al., 2015], o PCV com Múltiplas Janelas de Tempo e Seleção de Hotéis (PCVMJT-SH) [Baltz et al., 2014] e o PCV Múltiplo com seleção de Hotéis (PCVM-SH) [Castro et al., 2014b].

Ao longo deste trabalho, dois termos são utilizados para descrever a diferença entre uma jornada de trabalho e o total trabalhado por um caixeiro. O termo viagem é usado para indicar uma sequência de visita a clientes que inicia e termina em um dos hotéis disponíveis (intermediário ou extremo). Já o termo rota, especifica o conjunto de viagens conectadas que um dado caixeiro realiza.

De forma similar ao PCV, no PCVM o objetivo é alcançar um caminho de custo mínimo que visite todos os clientes exatamente uma única vez, utilizando múltiplos caixeiros. No PCVSH, o objetivo é visitar todos os clientes no menor número de dias e minimizar o tempo total viajado, realizando paradas em hotéis nos casos em que não seja possível atender a todos os clientes em uma única jornada de trabalho. Da combinação destas duas variantes, originou-se o PCVM-SH que considera um número máximo de jornadas a serem realizadas por cada um dos múltiplos caixeiros e tem como objetivo encontrar um conjunto de rotas, cada uma iniciando e finalizando em um local comum (hotel ou depósito), de forma que, os clientes sejam visitados uma única vez e o tempo total necessário para percorrer todas as rotas seja minimizado.

Dentre as aplicações práticas do PCVM-SH vale citar o problema em que caixeiros precisam entregar uma grande quantidade de encomendas, problema de transportadores de cargas perecíveis que precisam atender a um conjunto grande de clientes geograficamente dispersos e o problema de entregas de mercadorias realizado por VANTS (Veículos Aéreos Não Tripulados) que precisam ser recarregados periodicamente.

Ao herdar características relativas a estrutura de dois problemas que são $\mathcal{NP}$-Difíceis, notavelmente, a complexidade do PCVM-SH é no mínimo equivalente à de seus ascendentes. Mesmo considerando uma jornada de trabalho infinita e o mínimo possível de caixeiros, o que em tese facilitaria a resolução do problema, a complexidade seria equivalente à apresentada pelo PCV [Castro et al., 2014b].

É possível identificar que a dificuldade de resolução do PCVM-SH é proporcional a quantidade de recursos com que o problema está lidando. Por exemplo, número de clientes que precisam ser visitados, a quantidade de caixeiros e a quantidade de hotéis disponíveis. Estes recursos sem dúvida, interferem diretamente na qualidade da solução e no tempo computacional necessário.

Resolver o PCVM-SH de forma ótima é uma tarefa extremamente árdua. Logo, a utilização de métodos exatos fica limitada na maioria das vezes pelo tamanho da instância. Uma alternativa interessante é o uso de métodos heurísticos com a finalidade de obter soluções viáveis de alta qualidade em um tempo computacional aceitável.

Devido ao PCVM-SH ter sido apresentado recentemente na literatura, existe apenas um trabalho que utiliza uma abordagem heurística para tratar o problema. Essa metodologia [Castro et al., 2014b] é baseada no framework *Iterated Local Search* e configura o atual estado da arte para o problema em questão.

Este trabalho propõe a aplicação de uma abordagem heurística para tratar o PCVM-SH. A estratégia é testada para um grande conjunto de instâncias disponíveis na literatura e os resultados observados são comparados com a heurística que corresponde ao estado da arte para este problema.

O restante da estrutura deste trabalho é organizada como segue. Na Seção 2, é apresentada uma definição formal do problema. A Seção 3 apresenta a heurística proposta para tratar o PCVM-SH. Os resultados computacionais obtidos pela aplicação da heurística proposta são discutidos na Seção 4, e finalmente, na Seção 5 são apresentadas as conclusões acerca do trabalho, bem como, os trabalhos futuros a serem desenvolvidos envolvendo o problema abordado.

2. Descrição do Problema

Seja um conjunto de $s+1$ hotéis $H = \{0, ..., s\}$ e o conjunto de n clientes $C = \{s+1, ..., s+n\}$. O PCVM-SH é definido em um grafo direcionado $G = (V, A)$, onde $V = H \cup C$ é o conjunto de vértices no grafo e $A = \{(i, j)|i, j \in V, i \neq j\}$ é o conjunto das possíveis arestas. Para cada cliente que deve ser visitado, está associado um tempo de visita v_i, $i \in C$, sendo que para cada hotel $i \in H$, o tempo de visita é igual a zero ($v_i = 0$). Complementarmente, o tempo de viagem associado a cada aresta $(i, j) \in A$ é conhecido previamente.

Uma viagem d é limitada por um tempo máximo de percurso (jornada de trabalho) L. Cada rota r associada a um caixeiro p é limitada por um número máximo de viagens D. Dado m caixeiros, que serão associados à P rotas, cada um deve iniciar e finalizar sua rota no hotel h_i, $i = 0$ (depósito). O objetivo é encontrar m rotas que iniciem e finalizem no hotel $i = 0$, e que sejam limitadas por D jornadas de trabalho de maneira que o tempo total percorrido para visitar todos os clientes, incluindo o tempo de visita associado a cada um, seja o mínimo possível.

3. Algoritmo Proposto

A heurística proposta para resolução do PCVM-SH neste trabalho é uma combinação de técnicas clássicas utilizadas na área de otimização. A primeira técnica consiste na escolha de soluções de partida diversas ao longo da execução, chamada de *Multi-Start*, que garante a diversificação da solução provendo um mecanismo que permite alcançar diferentes ótimos locais, e possivelmente, o ótimo global. Porém, a utilização apenas de um mecanismo de diversificação não garante bons resultados, sendo necessário utilizar uma estratégia de intensificação de busca por soluções melhores. Neste caso, a heurística em questão (MLS) utiliza um procedimento de busca local capaz de melhorar iterativamente uma solução corrente. Abordagens similares foram empregadas com sucesso em trabalhos que lidam com problemas de definições de rotas, bem como roteamento de veículos [Penna et al., 2013; Subramanian et al., 2013; Michallet et al., 2014].

O algoritmo proposto (Algoritmo 1) que implementa a heurística MLS constrói uma solução inicial que possui propriedade determinística (linha 3). Internamente, a cada iteração é escolhido um dos três possíveis construtores disponíveis para gerar uma nova solução estocástica (linha 7) e uma nova solução é criada utilizando o construtor definido no passo anterior (linha 8). A solução construída é otimizada utilizando um procedimento de busca local (linha 10). Se, e somente se, a aplicação da busca local resulte em uma solução melhor que a solução global (S_BEST), esta será substituída pela solução corrente (linhas 11 - 13).

3.1. Construção da Solução

A fase de construção da solução utiliza 4 heurísticas distintas para gerar soluções capazes de abranger diferentes características de soluções ideais para cada tipo de instância do problema. Para isto, estas heurísticas dividem-se em 3 heurísticas puramente estocásticas (i.e. geram soluções distintas a cada iteração) e uma solução determinística, i.e. gera sempre a mesma solução, caso seja executada várias vezes. Ambos os métodos construtivos são procedimentos simples e já utilizados com sucesso em outros trabalhos [Penna et al., 2013; Castro et al., 2014b; Sousa et al., 2015].

Algoritmo 1: Multi-Start_LS($iter\_max$, α)

```
1  início
2      S_BEST ← ∅;
3      ConstruirSolução(S_BEST, InserMaisBarata);
4      i ← 0;
5      enquanto (i < iter_max) faça
6          S1 ← ∅;
7          Tipo ← rand(InserSequencial, InserParalela, InserMaisBarataLCR);
8          ConstruirSolução(S1, Tipo, α);
9          RVND(S1);
10         se (S1 melhor que S_BEST) então
11             S_BEST ← S1;
12         fim
13         i++;
14     fim
15     retorne S_BEST;
16 fim
```

Devido a característica de seleção dos hotéis intermediários a cada rota, todos os métodos construtivos, após a geração da solução, possuem um procedimento baseado na heurística de Lin e Kernighan [1973] que otimiza cada uma destas rotas que compõem a solução utilizando a implementação de Applegate et al. [2006], e posteriormente, divide-as em viagens viáveis (i.e. que satisfazem o critério de tempo máximo para uma viagem) [Sousa et al., 2015]. Este método de divisão é baseado no Algoritmo de Dijkstra [Dijkstra, 1959] e garante que as viagens serão viáveis em relação ao tamanho, porém pode gerar um número de viagens que excede o máximo permitido para uma dada rota. Neste caso, é necessário realizar a aplicação de um método que realoque os clientes que fazem parte da viagem excedente, remanejando para as demais rotas/viagens. Estes clientes são realocados para posições em que representem o menor acréscimo para composição do tempo total a ser percorrido por aquela solução.

O primeiro método construtivo a ser utilizado pelo Algoritmo 1 (linha 2) é o método de Inserção Mais Barata (ISM). Este método inicialmente insere para cada uma das rotas o cliente que menos incrementa o tempo total viajado da solução. Com isso, garante que ao menos um cliente será atendido por cada caixeiro. Os demais clientes são inseridos um a um, na posição em que representam o menor acréscimo no tempo total viajado. Neste caso, é realizada uma busca exaustiva pela melhor posição, considerando todas as rotas.

Os demais métodos construtivos são utilizados no Algoritmo 1 (linha 8) para produzir soluções distintas a cada iteração do *Multi-Start*. O método Inserção Mais Barata LRC (Lista Restrita de Candidatos), diverge do primeiro, pois utiliza uma Lista de Clientes (LC) para armazenar os clientes em ordem ascendente, quanto à sua contribuição para otimizar a solução, ou seja, do melhor a ser inserido até o pior. Após ser inserido um cliente em cada rota, os remanescentes são inseridos considerando a LRC (Equação 1) e apenas uma porção será considerada para escolha de forma aleatória a cada iteração. Esta porção é definida de acordo com o valor da contribuição e do parâmetro α.

$$LRC = \{c \in LC | valor(c) \leq min(valor(LC)) + \alpha[max(valor(LC)) - min(valor(LC))]\} \quad (1)$$

Objetivando aumentar a diversidade das soluções, dois métodos baseados no trabalho de Penna et al. [2013] foram utilizados. Os métodos construtivos de Inserção Paralela (IP) e Sequencial

(IS) possuem similaridades com o método de inserção mais barata, porém possuem internamente outro nível de aleatoriedade que garante, por exemplo, que haja um equilíbrio entre rotas em relação ao tempo total necessário para voltar do último cliente para o hotel de partida. De forma simplificada, na IP, a cada iteração é inserido um cliente em uma rota. Na IS, as rotas possuem tamanhos similares, pois o número de clientes é rateado entre os caixeiros e cada rota é preenchida por completo antes de iniciar a inserção de sua sucessora.

3.2. Busca Local

Apesar da heurística proposta apresentar 4 formas distintas para gerar soluções, esta característica não é suficiente para garantir que soluções boas ou até mesmo ótimas, sejam encontradas. Foi utilizada uma estratégia baseada em *Variable Neigborhood Descent* (VND) [Mladenović e Hansen, 1997], onde diferente da proposta original, a ordem de aplicação das estruturas de vizinhança (ver Subseção 3.2.1) é definida aleatoriamente antes de realizar a busca local. Esta estratégia de definição aleatória da ordem de aplicação foi utilizada em Penna et al. [2013] e Sousa et al. [2015], onde provou ser eficiente. Uma vez que a técnica de escolha aleatória traz bons resultados para o TSPHS e problemas similares, justifica-se então seu uso para a variante tratada por este trabalho. Ao aplicar as estruturas de vizinhança sempre na mesma ordem, a heurística tende a subutilizar as últimas estruturas que são testadas, podendo dificultar a saída de espaços de solução tidos como ótimos locais. Ao inserir a aleatoriedade, o potencial das estruturas de vizinhança é melhor explorado e com isso ocorre um aumento nas chances de obtenção de melhores resultados.

No Algoritmo 2 é exposto o funcionamento do procedimento de busca local RVND. Enquanto a solução atual puder ser melhorada pelas estruturas de vizinhança que realizam trocas de clientes entre rotas distintas (linhas 2 - 17), o procedimento RVND é reexecutado. Para cada rota definida na solução corrente (linhas 4 - 16), é especificada a ordem de aplicação das estruturas de vizinhança (linha 5). Para cada estrutura, é verificado se algum dos movimentos otimizará a solução atual (linha 8), caso afirmativo, a melhor solução é atualizada e o procedimento é reiniciado aplicando a primeira estrutura definida pela ordem de aplicação (linhas 10 - 11), caso contrário, a próxima estrutura na ordem é selecionada (linha 13).

Algoritmo 2: RVND(SOL_Y)

```
1  início
2      repita
3          SOL_Y1 ← SOL_Y;
4          para cada rota em SOL_Y' faça
5              DF[ ] ← Define_Ordem();
6              k ← 0;
7              enquanto (k < numero_Vizinhancas) faça
8                  Busca(N_DF[k], SOL_Y');
9                  se (SOL_Y' melhor que SOL_Y) então
10                     SOL_Y ← SOL_Y';
11                     k ← 0;
12                 senão
13                     k ← k + 1;
14                 fim
15             fim
16         fim
17     enquanto (Shift(SOL_Y) == melhora)||(Swap(SOL_Y) == melhora));
18     retorne SOL_Y;
19 fim
```

Complementarmente, o Algoritmo 3 é o responsável por realizar a avaliação de cada movimento passível de ser aplicado (linha 4). A função de avaliação de uma dada solução é apresentada pela Equação 2 e consiste na soma do tempo total necessário para atender a todos os clientes, penalizando aquelas viagens que excederem o tempo máximo permitido L. Caso o melhor movimento contribua para a melhora da solução, essa solução é atualizada e otimizada por uma estrutura de vizinhança intra viagem (linha 6).

$$F(y) = \sum_{p=1}^{P}\sum_{d=1}^{D}(tempo_{pd}) + \omega \sum_{p=1}^{P}\sum_{d=1}^{D} max(0, tempo_{pd} - L) \quad (2)$$

Algoritmo 3: Busca(N_k, SOL_Y)

```
1  begin
2      SOL_Y' ← SOL_Y
3      repita
4          SOL_Y'' ← argmin_{S∈N_k(SOL_Y')} F(S)
5          se (SOL_Y'' melhor que SOL_Y') então
6              SOL_Y' ← otimizar SOL_Y'' com OR-OPT
7          fim
8      enquanto melhoras forem encontradas;
9      retorne SOL_Y'
10 end
```

Internamente ao processo de busca local, foram utilizadas nove estruturas de vizinhança. Ambas amplamente utilizadas com sucesso em diversas variantes do PCV [Castro et al., 2013, 2014a] e também do Problema de Roteamento de Veículos [Penna et al., 2013], outro problema clássico de otimização. As vizinhanças podem ser classificadas de acordo com o escopo de sua aplicação: (i) vizinhanças que atuam manipulando clientes e (ii) vizinhanças específicas para trabalhar com hotéis. Ainda podem ser classificadas considerando a localidade de aplicação: (i) internamente a uma viagem (intra viagem); (ii) entre viagens distintas (inter viagens) e (iii) entre rotas distintas (inter rota).

No momento de aplicação de cada estrutura, todas as trocas possíveis, sejam elas de clientes ou de hotéis são analisadas. Para que a busca seja guiada efetivamente para uma solução vizinha, esta deve ser melhor que a solução corrente, considerando sempre a solução que mais contribua com a melhora da solução atual (*i.e. best improvement*).

3.2.1. Estruturas de Vizinhança Intra Rota

Dentre as nove estruturas de vizinhança utilizadas, sete são aplicadas apenas internamente a uma rota específica de cada vez. Podem ser focadas em operações sobre clientes: intra viagem (*2OPT* e *OrOPT*) e inter viagem (*Relocate*, *Exchange*) ou sobre hotéis (*InsertHotel*, *RemoveHotel* ou *ChangeHotel*). As estruturas que compõem a heurística proposta, são:

- ***2OPT*** [Croes, 1958] - realiza movimentos dentro de uma viagem de forma que dois arcos distintos que conectam clientes são removidos e reconectados de forma diferente, invertendo a ordem de visita dos clientes internos aos arcos quebrados.

- ***OrOpt*** [Or, 1976] - busca realocar uma quantidade k de clientes, onde $k = \{3, 2, 1\}$, dentro de cada viagem. Esta estrutura irá aplicar, caso melhore a solução corrente, aquele movimento que produza a melhor solução dentre todos os movimentos testados.

- ***Relocate*** [Laporte et al., 2000] - realoca uma quantidade k de clientes, onde $k = \{3, 2, 1\}$, entre viagens distintas. A modificação da solução corrente somente será efetivada se a realocação resultar em uma melhora da mesma.

- ***Exchange*** [Laporte et al., 2000] - troca uma quantidade k de clientes, tendo $k = \{3, 2, 1\}$, entre viagens distintas. De forma análoga ao *Relocate*, a solução corrente somente será alterada após a confirmação de que a aplicação da vizinhança resulte em melhora na solução atual.

- ***InsertHotel*** [Castro et al., 2014b] - insere um hotel intermediário na rota, caso esta inserção contribua positivamente para reduzir a inviabilidade da solução, ou seja, reduza ou até mesmo anule a quantia que excede o tempo limite para se percorrer uma viagem.

- ***RemoveHotel*** [Castro et al., 2014b] - remove um hotel intermediário caso esta remoção contribua minimizando o tempo total necessário para percorrer a rota. Esta estrutura também tem o objetivo de retirar hotéis repetidos que podem aparecer em sequência na solução após a aplicação de outras estruturas como o *Relocate* ou *Shift* (detalhado a seguir).

- ***ChangeHotel*** [Castro et al., 2014a] - verifica se cada um dos hotéis intermediários podem ser substituídos por algum outro hotel disponível, de forma a, gerar uma solução com tempo total percorrido menor que aquele apresentado pela solução atual.

3.2.2. Estruturas de Vizinhança Inter Rota

Os métodos construtivos dificilmente são capazes de alocar cada cliente na rota considerada ideal para garantir o melhor valor possível para a solução (em alguns casos o ótimo). A aleatoriedade presente nestes métodos não garante por si só, que os clientes serão alocados nas suas respectivas rotas ideais.

Diante desta necessidade de movimentar clientes entre rotas, foram utilizadas duas estruturas de vizinhança que exploram diferentes formas de realocar os clientes entre duas rotas distintas. São elas:

- ***Shift*** [Penna et al., 2013] - estrutura que atua de forma semelhante ao *Relocate*, porém neste caso, a realocação de clientes é realizada entre rotas. Considera um conjunto k de clientes para serem realocados, testando $k = \{3, 2, 1\}$. Para cada um dos movimentos possíveis utilizando os valores de k, apenas aquele que melhor otimiza a solução atual, será aplicado. Caso não haja melhoria, nenhum movimento será aplicado a solução corrente.

- ***Swap*** [Penna et al., 2013] - estrutura similar ao *Exchange*, porém a troca é realizada entre rotas e não entre viagens como ocorre nesta estrutura de vizinha comparada. Esta estrutura envolve um número maior de possibilidades de trocas de clientes, uma vez que, testa movimentos trocando (3, 3), (3, 2), (2, 2), (2, 1) e (1, 1) clientes. De forma similar as estruturas anteriores, apenas o movimento que guie a solução corrente em direção a maior melhora, será aplicado. Também não é permitido a aplicação de um movimento que não melhore a solução.

4. Resultados Computacionais

Nesta seção são apresentados os resultados obtidos pela heurística proposta utilizando um grupo de instâncias geradas de forma aleatória. A descrição do processo de criação das instâncias é apresentado em Castro et al. [2014b]. Basicamente, as instâncias são divididas em dois subgrupos: clusterizadas, que possuem como característica o agrupamento dos clientes por regiões no espaço considerado, e randomizadas, que não apresentam formas de agrupamento, sendo a escolha do local dos clientes realizada aleatoriamente.

Para geração das instâncias foram utilizados diferentes níveis de parâmetros. Para cada combinação de parâmetros apresentados na Tabela 1 e para cada um dos tipos de instância (clusterizadas e randomizadas), foram criadas 5 instâncias distintas totalizando 900 instâncias. Assim como adotado no trabalho de Castro et al. [2014b], os resultados que são apresentados no decorrer desta seção correspondem a média de 10 execuções da heurística proposta para cada instância (com exceção da Tabela 3 que utiliza a melhor solução entre as 10 execuções).

Tabela 1: Parâmetros utilizados para gerar as instâncias.

Parâmetro	Valores
Número de clientes (n)	50 – 75 – 100 – 200 - 300
Número de caixeiros (m)	2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 - 10
Número máximo de viagens	5 – 10

A estratégia proposta foi codificada em C++ e o compilador g++ 4.8.2 foi utilizado. Os testes foram executados em um computador com arquitetura Intel i7-870 2.93GHz e 8GB de memória RAM. O sistema operacional utilizado foi o Ubuntu 14.04 64-bit, sendo que todos os testes foram executados utilizando uma única *thread*. Os parâmetros utilizados para execução da heurística foram definidos empiricamente de acordo com exaustivos testes computacionais realizados com diferentes configurações. Desta forma, os valores para cada um dos parâmetros segue: $iter_{max}$ = 200 e α = 0,2.

Para avaliação dos resultados obtidos, foi utilizada a métrica de diferença percentual (Equação 3) entre a média da solução encontrada pela abordagem MLS e a solução considerada ótima (definida no momento da criação das instâncias). Esta métrica também foi utilizada para apresentar os resultados encontrados pela metaheurísta de Castro et al. [2014b], bastando apenas substituir o resultado de acordo com a heurística utilizada. A equação que define o Gap (em porcentagem) é a seguinte:

$$Gap = 100 \times \frac{Tempo(MLS) - Tempo(otimo)}{Tempo(otimo)}. \quad (3)$$

As Tabelas 2, 3 e 4 apresentam respectivamente as médias de Gap entre a execução da heurística MLS e a solução ótima. As Tabelas 5 e 6 expõem os tempos computacionais médios demandados.

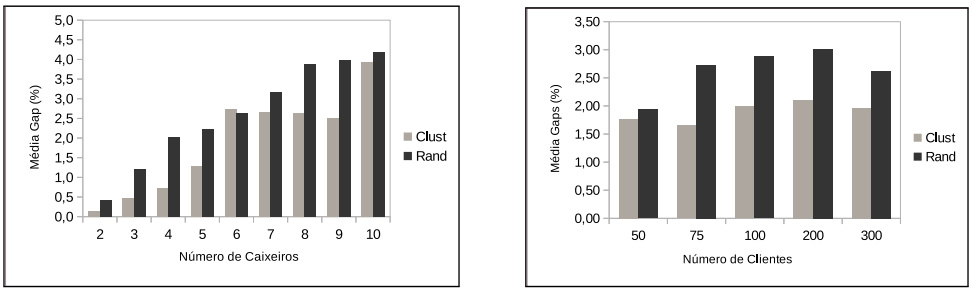
Nas tabelas seguintes, a primeira coluna contém a quantidade de clientes relativa a cada uma das instâncias testadas. Da segunda até a décima coluna são apresentados, na primeira linha a quantidade de caixeiros que cada instância possui e nas demais linhas, são apresentados os Gaps médios (Tabelas 2, 3 e 4). Ainda considerando estas colunas, para a Tabela 5 e Tabela 6 o tempo computacional médio é apresentado. A última coluna apresenta a média das médias dos Gaps/tempos computacionais alcançados para instâncias que compartilham uma mesma quantidade de clientes. Adicionalmente, na última linha de cada tabela é apresentada a média das médias para Gap/tempo computacional agrupando os valores pela quantidade de caixeiros disponíveis em cada instância.

Tabela 2 contém a média dos Gaps produzidos pela heurística MLS comparando-se com o valor da solução ótima. Verifica-se que a média varia entre 0,0% (n = 50, m = 2) e 4,7% (n = 200, m = 10). Figura 1(a) apresenta a média dos Gaps agrupados pelo número de caixeiros e pelo tipo de instância. Quanto menos caixeiros disponíveis, mais fácil se torna a resolução do problema e apenas para um grupo de instâncias (6 caixeiros) o tipo clusterizado é mais difícil de ser resolvido quando comparado ao tipo de instâncias randomizado. Figura 1(b) ilustra a média dos Gaps agrupadas pelo tipo da instância e pelo número de clientes. É possível verificar que a média do Gap tende a

aumentar com o aumento do número de clientes e também que as instâncias do tipo randomizadas são inerentemente mais difíceis de serem resolvidas.

Tabela 2: Média dos Gaps (%) encontrados pela abordagem tendo como referência a média entre execuções.

n/m	2	3	4	5	6	7	8	9	10	Total
50	0,0	0,2	0,5	1,1	1,9	2,6	3,1	3,0	3,7	1,8
75	0,2	0,5	1,0	1,4	2,3	2,1	3,1	3,7	4,4	2,1
100	0,1	0,6	1,2	1,6	2,8	2,9	3,5	3,4	4,5	2,3
200	0,1	0,9	1,4	1,6	2,3	3,3	3,3	3,6	4,7	2,3
300	0,4	0,9	1,6	1,9	2,7	3,2	2,8	2,5	3,0	2,1
Total	0,2	0,6	1,1	1,6	2,4	2,8	3,2	3,3	4,1	2,1



(a) Média Gaps por caixeiro e tipo de instância (b) Média Gaps por clientes e tipo de instância

Figura 1: Média Gaps agrupadas de diferentes formas

A Tabela 3 traz informações similares às apresentadas na Tabela 2, porém neste caso os resultados médios de Gap apresentados são definidos considerando-se a melhor solução encontrada para cada instância, ao invés, de considerar a média das 10 execuções. A média dos Gaps neste teste apresenta em relação as soluções ótimas, uma porcentagem mínima de 0,0% (n = {50, 75, 200}, m = {2, 3, 4}) e máxima de 2,9% (n = 100, m = 10). Estes resultados são próximos aos encontrados pela heurística estado da arte [Castro et al., 2014b].

Tabela 3: Média Gaps (%) utilizando como referência a melhor solução encontrada.

n/m	2	3	4	5	6	7	8	9	10	Total
50	0,0	0,0	0,0	0,4	0,6	1,6	1,7	1,6	1,7	0,8
75	0,0	0,1	0,5	0,6	1,1	1,2	1,7	2,5	2,3	1,1
100	0,1	0,1	0,2	0,2	1,4	0,7	1,8	1,9	2,9	1,0
200	0,0	0,3	0,3	0,5	0,7	1,4	1,3	1,4	2,4	0,9
300	0,1	0,1	0,4	0,6	0,8	1,3	1,3	1,2	1,4	0,8
Total	0,0	0,1	0,3	0,5	0,9	1,2	1,5	1,7	2,1	0,9

Os resultados encontrados por Castro et al. [2014b] (Tabela 4) demonstram que a média do Gap varia entre 0,0% (n = 50, m = 2) e 1,4% (n = 300, m = 6).

Tabela 4: Média dos Gaps (%) encontrados por Castro et al. [2014b] usando a média das execuções.

n/m	2	3	4	5	6	7	8	9	10	Total
50	0,0	0,1	0,3	0,2	0,1	0,2	0,4	0,1	0,2	0,2
75	0,2	0,2	0,2	0,3	0,2	0,2	0,2	0,3	0,3	0,2
100	0,6	0,1	0,2	0,5	0,4	0,2	0,4	0,4	0,4	0,3
200	0,7	0,6	0,5	0,5	0,8	0,8	0,8	0,6	0,9	0,7
300	1,2	0,9	1,0	1,1	1,4	1,3	1,1	1,1	1,1	1,1
Total	0,5	0,4	0,4	0,5	0,6	0,5	0,6	0,5	0,6	0,5

A Figura 2 ilustra a porcentagem de soluções ótimas que são encontradas de acordo com o número de caixeiros disponíveis. É possível identificar a correlação das informações entre a Figura 1 e a Figura 2, pois na primeira é possível verificar que as instâncias randomizadas são mais difíceis que as clusterizadas. Na segunda figura, esta dificuldade se reflete na porcentagem de soluções ótimas encontradas, sendo maior para as instâncias clusterizadas.

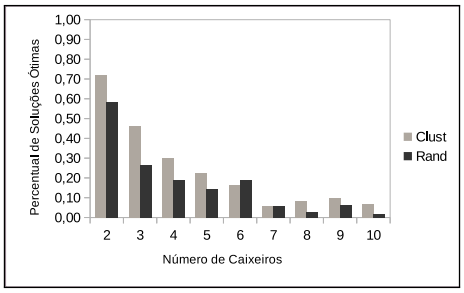


Figura 2: Percentual de soluções ótimas encontradas de acordo com o número de caixeiros

A Tabela 5 apresenta os tempos computacionais médios necessários para a execução das instâncias. De forma análoga ao formato de apresentação das médias do Gap, aqui são apresentadas as médias do tempo computacional agrupadas por quantidade de caixeiros e número de clientes presentes nas instâncias. Considerando que foram apresentados os tempos computacionais médios e que o valor máximo foi de 27,6 segundos para instâncias com n = 300, este valor é aceitável ao propósito de uma heurística. Ou seja, a heurística proposta alcança resultados de Gap médios próximos ao estado da arte da literatura, demandando um pequeno tempo computacional.

Tabela 5: Média de tempo computacional para a heurística proposta (em segundos).

n/m	2	3	4	5	6	7	8	9	10	Total
50	0,6	0,6	0,6	0,6	0,6	0,7	0,7	0,8	0,8	0,7
75	1,4	1,2	1,2	1,3	1,3	1,4	1,3	1,5	1,6	1,3
100	2,3	2,1	2,0	2,1	2,1	2,2	2,3	2,4	2,3	2,2
200	7,9	8,1	8,4	8,2	9,0	9,3	9,6	10,3	10,5	9,0
300	18,8	20,6	20,8	20,1	22,4	22,9	24,3	26,1	27,6	22,6
Total	6,2	6,5	6,6	6,5	7,1	7,3	7,7	8,2	8,6	7,2

Para efeito comparativo, na Tabela 6 são apresentados os tempos computacionais médios demandados pela heurística estado da arte para o PCVM-SH. O tempo computacional máximo para este caso é de 33,7 segundos na média, para instâncias com n = 300. Comparando a abordagem proposta ao estado da arte, a primeira possui um total médio de tempo computacional igual a 7,2 segundos, enquanto a segunda 10,3. Em termos de porcentagem, a heurística estado da arte utiliza cerca de 30% mais tempo computacional.

Tabela 6: Média de tempo computacional para a abordagem de Castro et al. [2014b] (em segundos).

n/m	2	3	4	5	6	7	8	9	10	Total
50	0,8	0,9	1,2	1,1	1,2	1,3	1,2	1,1	1,2	1,1
75	1,8	1,9	1,9	2,0	2,0	2,0	2,1	2,3	2,2	2,0
100	2,8	2,8	3,3	3,5	3,6	3,4	3,5	3,8	3,9	3,4
200	14,4	14,0	13,0	12,9	13,3	13,3	12,6	12,3	13,1	13,2
300	33,4	33,4	30,5	32,4	31,1	33,7	30,3	29,8	31,4	31,8
Total	10,7	10,6	10,0	10,4	10,2	10,7	9,9	9,9	10,4	10,3

5. Conclusões

Neste trabalho foi abordado o Problema do Caixeiro Viajante Múltiplo com Seleção de Hotéis. Este problema, surgiu recentemente na literatura e é uma variante do clássico Problema

do Caixeiro Viajante Múltiplo combinado com o Problema do Caixeiro Viajante com Seleção de Hotéis. A literatura pertinente ao problema apesar de limitada, traz a modelagem matemática do problema e uma heurística baseada em *Iterated Local Search*. Foi proposta uma heurística que possui um mecanismo de busca local embarcado a uma estratégia *Multi-Start*. Os resultados computacionais obtidos comprovam a superioridade da heurística existente na literatura, em relação a qualidade da solução. Quanto ao tempo computacional a heurística proposta utiliza cerca de 30% menos tempo.

Os resultados encontrados para as médias realizadas com apenas a melhor solução dentre as 10 execuções, demonstram que a proposta deste trabalho consegue resultados próximos aos contidos na literatura. Apesar das limitações da heurística, há indícios suficientes de que pode ser aperfeiçoada após a realização de um estudo aprofundado da contribuição de cada passo da heurística, e também utilizando técnicas clássicas de otimização com o intuito de fornecer maior competitividade.

Como pesquisas futuras é interessante testar outras heurísticas e metaheurísticas clássicas para o PCVM-SH. Utilizar estruturas de dados mais eficientes para reduzir o tempo computacional, pode influenciar positivamente na quantidade de vezes em que o procedimento de busca local será executado. Bons resultados podem ainda ser encontrados utilizando procedimentos de perturbação da solução em nível de hotéis e clientes e modelando matematicamente o problema com o intuito de hibridizá-lo por meio da combinação do método exato com a heurística.

Agradecimentos

Os autores gentilmente agradecem o fornecimento de recursos computacionais por parte do Laboratório de Inteligência Computacional (LABIC - UFF) da Universidade Federal Fluminense e o apoio fornecido pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Referências

Applegate, D. L., Bixby, R. E., Chvatal, V., e Cook, W. J. (2006). *The traveling salesman problem: a computational study*. Princeton University Press.

Baltz, A., Ouali, M. E., Jäger, G., Sauerland, V., e Srivastav, A. (2014). Exact and heuristic algorithms for the travelling salesman problem with multiple time windows and hotel selection. *Journal of the Operational Research Society*, 66:615 – 626.

Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209 – 219.

Castro, M., Sörensen, K., Vansteenwegen, P., e Goos, P. (2014a). A fast metaheuristic for the travelling salesperson problem with hotel selection. *4OR*, p. 1 – 20.

Castro, M., Sörensen, K., Goos, P., e Vansteenwegen, P. (2014b). The multiple travelling salesperson problem with hotel selection. Technical report, University of Antwerp.

Castro, M., Sörensen, K., Vansteenwegen, P., e Goos, P. (2013). A memetic algorithm for the travelling salesperson problem with hotel selection. *Computers & Operations Research*, 40(7): 1716 – 1728.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6 (6):791 – 812.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Laporte, G., Gendreau, M., Potvin, J. Y., e Semet (2000). Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, 7(4-5):285 – 300.

Lin, S. e Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498 – 516.

Michallet, J., Prins, C., Amodeo, L., Yalaoui, F., e Vitry, G. (2014). Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. *Computers & operations research*, 41:196 – 207.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100.

Or, I. (1976). *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Northwestern University, Evanston, Illinois.

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.

Sousa, M. M., Ochi, L. S., Coelho, I. M., e Goncalves, L. B. (2015). A variable neighborhood search heuristic for the traveling salesman problem with hotel selection. In *Computing Conference (CLEI), 2015 Latin American*, p. 1 – 12. IEEE.

Subramanian, A., Uchoa, E., e Ochi, L. S. (2013). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519 – 2531.

Vansteenwegen, P., Souffriau, W., e Sörensen, K. (2012). The travelling salesperson problem with hotel selection. *Journal of the Operational Research Society*, 63(2):207 – 217.