LI Simpósio Brasileiro de Pesquisa Operacional
Limeira–SP, 02 a 06 de setembro de 2019

Uma heurística *Large Neighborhood Search* para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas

Armando H. Pereira, Sebastián A. Urrutia
Departamento de Ciência da Computação - Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627, Pampulha, Belo Horizonte, MG, CEP 31270-901, Brasil
{ahp,surrutia}@dcc.ufmg.br

RESUMO

Nesse trabalho apresentamos uma heurística *Large Neighborhood Search* (LNS) para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. No problema, um único veículo deve atender a um conjunto de requisições. Cada requisição é definida por uma localização de coleta, onde um item deve ser carregado no veículo, e uma localização de entrega, onde o item carregado deve ser entregue. Os itens são armazenados em pilhas de capacidade limitada dentro do veículo. Cada pilha segue a política *Last-In-First-Out* (LIFO). O objetivo do problema é encontrar uma rota que atenda todas as requisições e minimize a distância percorrida. Propomos novos operadores e melhorias para uma heurística LNS da literatura. Os experimentos computacionais mostram que a nova heurística é capaz de obter limites primais mais apertados em menor tempo. Além disso, as melhores soluções viáveis conhecidas para várias instâncias são melhoradas.

PALAVRAS CHAVE. Caixeiro Viajante, Restrições de Carregamento, Large Neighborhood Search, Coleta e Entrega.

Roteamento, Metaheurísticas, Restrições de Carregamento

ABSTRACT

In this work we present a Large Neighborhood Search heuristic (LNS) for the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks. In the problem, a single vehicle must fulfill a set of requests. Each request is defined by a pickup location, where one item must be loaded in the vehicle, and a delivery location, where the loaded item must be delivered. The items are stored in stacks with limited capacity inside the vehicle. Each stack follows the Last-In-First-Out (LIFO) policy. The objective of the problem is to find a route fulfilling all requests and minimizing the traveled distance. We propose new operators and improvements to a LNS heuristic of the literature. Computational experiments show that the new heuristic is able to obtain tighter primal bounds in a shorter time. In addition, the best feasible solutions known to various instances are improved.

KEYWORDS. Traveling Salesman, Loading Constraints, Large Neighborhood Search, Pickup and Delivery.

Routing, Metaheuristics, Loading Constraints

https://proceedings.science/p/106901?lang=pt-br

1. Introdução

Este trabalho trata do Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas (*Pickup and Delivery Traveling Salesman Problem with Multiple Stacks* - PDTSPMS). Nesse problema, um único veículo deve atender a um conjunto de requisições de clientes. Cada requisição é definida por duas localizações, a localização de coleta, onde um item deve ser carregado no veículo, e a localização de entrega, onde o item carregado deve ser entregue. Os itens são armazenados em pilhas de capacidade limitada dentro do veículo. Cada pilha deve seguir a política *Last-In-First-Out* (LIFO), isto é, somente o último item carregado em cada pilha está disponível para entrega. O objetivo do problema é encontrar uma rota para o veículo que atenda todas as requisições e que minimize a distância percorrida.

O PDTSPMS encontra aplicações onde a coleta e a entrega dos itens só podem ser realizadas pela traseira do veículo (e.g. carro furgão e caminhão cegonheira). Somente o último item estar disponível para entrega evita o problema de realocar os itens que estão bloqueando o caminho da entrega dentro do veículo. Além disso, reposicionar os itens fora do depósito é oneroso, uma vez que os itens podem ser grandes, pesados, frágeis ou perigosos. Dentro desse contexto, é fundamental que o itinerário do veículo seja elaborado para evitar contratempos. Os custos associados à operação dos veículos e à armazenagem dos itens formam uma importante componente dos gastos em sistemas de distribuição, dessa forma, uma pequena diminuição de custos pode resultar em uma economia significativa ao final do processo logístico. Consequentemente, o problema fornece uma fórmula para uma empresa transportadora reduzir os custos com sua frota e mão-de-obra.

[Ladany e Mehrez, 1984] fizeram a primeira contribuição para o Problema do Caixeiro Viajante com Coleta e Entrega onde a política LIFO é imposta. Os autores tratam uma aplicação real do problema, que envolve uma empresa de fretes em Israel. [Levitin e Abezgaouz, 2003] abordam uma aplicação real do problema em um armazém. Os itens são colocados em paletes, e cada novo palete é carregado no topo do lote anterior por um veículo guiado automatizado. Para evitar o uso excessivo de espaço e tempo ao rearranjar os paletes, os itens são transportados dentro do armazém de acordo com a política LIFO.

O PDTSPMS generaliza o *Pickup and Delivery Traveling Salesman Problem with LIFO loading constraints* (PDTSPL). O PDTSPL é uma versão mais simples onde o veículo contém somente uma pilha de capacidade ilimitada. O PDTSPL foi estudado inicialmente por [Carrabs et al., 2007b], os autores desenvolvem operadores de busca local e uma heurística *Variable Neighborhood Search* (VNS). [Carrabs et al., 2007a] propuseram um algoritmo exato *branch-and-bound* para o problema. [Cordeau et al., 2010] apresentaram três formulações de programação inteira para o PDTSPL, duas para o caso capacitado e uma para o não capacitado. Desigualdades válidas também são propostas e subsequentemente usadas em um algoritmo exato *branch-and-cut*. Um estudo poliedral e um algoritmo *branch-and-cut* são apresentados em [Dumitrescu et al., 2010]. [Li et al., 2011] sugeriram representar soluções do PDTSPL como árvores ao invés de listas. Eles desenvolvem uma heurística VNS para o problema e, além de proporem novos operadores, adaptam os operadores propostos por [Carrabs et al., 2007b].

Outra variante do PDTSPMS, introduzida no trabalho de [Petersen e Madsen, 2009], é o *Double Traveling Salesman Problem with Multiple Stacks* (DTSPMS). No DTSPMS todas as coletas devem ser executadas antes que qualquer entrega possa ser feita. [Petersen e Madsen, 2009] propuseram um modelo matemático e meta-heurísticas para o problema. [Ángel Felipe et al., 2009] propuseram quatro operadores de vizinhança que são usados em uma heurística VNS. A heurística anterior é melhorada no trabalho de [Ángel Felipe et al., 2011]. [Øvstebø et al., 2011] abordaram um DTSPMS no transporte de veículos feitos por navios. Os autores implementaram um método heurís-

https://proceedings.science/p/106901?lang=pt-br

tico baseado em busca tabu para o problema. Uma heurística LNS desenvolvida para o PDTSPMS é aplicada no DTSPMS por [Côté et al., 2012b].

O primeiro algoritmo exato para o PDTSPMS foi proposto por [Côté et al., 2012a]. Eles forneceram três formulações de programação inteira e desigualdades válidas que são incorporadas no algoritmo *branch-and-cut* apresentado. [Sampaio e Urrutia, 2017] introduziram uma formulação baseada em arcos para o PDTSPMS junto com um algoritmo *branch-and-cut*. As abordagens propostas por [Côté et al., 2012a] e [Sampaio e Urrutia, 2017] têm resultados computacionais similares. [Pereira e Urrutia, 2018] propuseram novas formulações e algoritmos exatos para o PDTSPMS baseando-se nas ideias dos dois trabalhos anteriores. Os novos algoritmos exatos propostos foram capazes de resolver várias instâncias não resolvidas da literatura, além de reduzir os tempos para as instâncias já resolvidas previamente.

Apesar da atenção recebida na área de algoritmos exatos, poucos trabalhos relacionados à heurísticas foram propostos para o PDTSPMS. Supomos que isso seja parcialmente devido a dificuldade de propor operadores de busca local que mantenham a viabilidade do problema. Como explicamos na Seção 3.2, o problema possui várias nuances e restrições para manter a viabilidade de uma solução. No melhor do nosso conhecimento, o único trabalho envolvendo heurísticas para o PDTSPMS foi realizado em [Côté et al., 2012b]. Os autores propuseram uma heurística LNS para o problema. Ao término da execução da heurística um algoritmo de programação dinâmica é aplicado na tentativa de melhorar a solução obtida. Nesse trabalho tomamos a heurística em questão como ponto de partida e propomos novos operadores e mecanismos de busca local. Através dos refinamentos anteriores eliminamos a necessidade do algoritmo de programação dinâmica, diminuindo o tempo de execução sem prejuízo à qualidade das soluções. Os resultados mostram que a nova heurística é capaz de obter soluções melhores com um tempo de execução menor.

Descrevemos formalmente e formulamos o problema na Seção 2. Na Seção 3 descrevemos o funcionamento dos operadores e da heurística LNS. Na Seção 4 os experimentos e os resultados da heurística são avaliados e discutidos. Na Seção 5 expomos nossas conclusões e direções futuras de estudos.

2. Descrição formal

Seja $G = (V, A)$ um grafo direcionado e completo. Particionamos o conjunto de vértices $V = P \cup D \cup \{0, 2n + 1\}$ em três subconjuntos disjuntos, $P = \{1, \ldots, n\}$ é o conjunto de pontos de coleta, $D = \{n+1, \ldots, 2n\}$ é o conjunto de pontes de entrega, e 0 e $2n+1$, respectivamente, os depósitos inicial e final. Definimos o conjunto de arcos como $A = \{\, (i, j) \mid i, j \in V, i \neq j \,\}$. Um custo $c_{ij} \geq 0$ é associado ao uso de cada arco $(i, j) \in A$. Um conjunto de n requisições deve ser atendido. Cada requisição tem uma coleta $i \in P$ associada com uma entrega $n+i \in D$, e impõe que o item coletado na localização i deve ser entregue na localização $n + i$. Cada localização de coleta $i \in P$ tem um item associado de tamanho $d_i \geq 0$. Consideramos que o item associado à localização $n + i$ tem tamanho $-d_i$. Os depósitos possuem itens com tamanhos $d_0 = d_{2n+1} = 0$. Um único veículo com um conjunto de pilhas $M = \{1, \ldots, |M|\}$, cada uma com capacidade $Q > 0$, é usado para atender as requisições. Todas as pilhas do veículo seguem a política LIFO, ela determina que todo item é carregado no topo de uma pilha, e só pode ser descarregado se também estiver no topo. O objetivo do PDTSPMS é satisfazer todas as requisições com custo mínimo, começando e terminando, respectivamente, nos depósitos 0 e $2n + 1$, satisfazendo as restrições de precedência, capacidade e a política LIFO.

Para modelar o problema, apresentamos a formulação de fluxo proposta em [Pereira e Urrutia, 2018]. Com o intuito de expor a formulação, definimos as seguintes variáveis:

- $\{x_{ij} \in \mathbb{B} : (i, j) \in A\}$, $x_{ij} = 1$ se o arco $(i, j) \in A$ é usado, caso contrário $x_{ij} = 0$.

- $\{z_i^k \in \mathbb{B} : i \in V, k \in M\}$, $z_i^k = 1$ se o item $i \in V$ é carregado na pilha $k \in M$, caso contrário, $z_i^k = 0$.
- $\{f_{ij}^k \in \mathbb{R}_+ : (i,j) \in A, k \in M\}$, f_{ij}^k é o valor da carga do veículo na pilha k no arco $(i,j) \in A$.
- $\{r_{ij} \in \mathbb{B} : i \neq j \in V\}$, $r_{ij} = 1$ se o veículo visita i antes de j, 0, caso contrário.

Além disso, definimos a função $\gamma(i) = i$ para todo $i \in P$, e $\gamma(n+i) = i$ para todo $n+i \in D$. O PDTSPMS pode ser formulado da seguinte maneira:

$$\min \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1}$$

$$s.t. \quad \sum_{j\in V} x_{ij} = 1, \qquad i \in P \cup D \cup \{0\} \tag{2}$$

$$\sum_{j\in V} x_{ji} = 1, \qquad i \in P \cup D \cup \{2n+1\} \tag{3}$$

$$\sum_{k\in M} z_i^k = 1, \qquad i \in P \tag{4}$$

$$\sum_{(i,j)\in A} f_{ij}^k - \sum_{(j,i)\in A} f_{ji}^k = d_i z_{\gamma(i)}^k, \qquad i \in P \cup D, k \in M \tag{5}$$

$$f_{ij}^k \leq Q x_{ij}, \qquad (i,j) \in A, k \in M \tag{6}$$

$$r_{ij} + r_{ji} = 1, \qquad i,j \in P \cup D \tag{7}$$

$$r_{ij} \geq x_{ij}, \qquad i,j \in P \cup D \tag{8}$$

$$r_{ij} \geq r_{iv} + x_{vj} - 1, \qquad i,j,v \in P \cup D \tag{9}$$

$$r_{n+i,i} = 0, \qquad i \in P \tag{10}$$

$$z_i^k + z_j^k + r_{ij} + r_{j,n+i} + r_{n+i,n+j} \leq 4, \qquad i,j \in P, k \in M \tag{11}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in A \tag{12}$$

$$r_{ij} \in \{0,1\}, \qquad i,j \in P \cup D, i \neq j \tag{13}$$

$$z_i^k \in \{0,1\}, \qquad i \in P, k \in M \tag{14}$$

$$f_{ij}^k \in \mathbb{R}_+, \qquad (i,j) \in A, k \in M \tag{15}$$

A função objetivo (1) minimiza o custo total da rota. O veículo sai e entra em cada nó uma única vez (2)-(3).Cada item deve ser carregado em uma única pilha (4). As restrições (5) estabelecem as restrições de conservação de fluxo na rota, cada coleta aumenta o fluxo e cada entrega diminui o fluxo. A carga em cada pilha não deve exceder a capacidade da pilha (6). As restrições (7)-(9) determinam a ordem relativa de visitação entre as localizações na rota. As restrições (10) impõem a precedência entre as localizações de coleta e entrega. Finalmente, as restrições (11) estabelecem a política LIFO na rota.

3. *Large Neighborhood Search*

Abordagens clássicas da literatura para problemas de roteamento envolvem movimentos que modificam a solução de maneira moderada, por exemplo, o operador 2-opt para o Problema do

Caixeiro Viajante. Uma heurística LNS funciona de forma mais drástica. Em cada iteração uma solução é destruída e posteriormente reconstruída. Isso é feito através da remoção de requisições da rota e da sua reinserção em uma posição que, possivelmente, proporcione melhoria. Indicamos o trabalho de Hansen e Mladenović [2005] para o leitor interessado em mais informações sobre LNS. A seguir explicamos os operadores de remoção e os operadores de inserção.

3.1. Operadores de remoção

Os operadores removem q requisições da rota atual. A remoção de uma requisição i implica em deletar o nó de coleta i da rota, assim como deletar o nó de entrega $n + i$ da mesma. Vale ressaltar que uma rota viável permanecerá viável com a remoção dos q itens, uma vez que removemos tanto a coleta quanto a entrega de uma requisição. Os operadores descritos nas Seções 3.1.1, 3.1.2, e 3.1.3, são operadores propostos em [Côté et al., 2012b]. O operador na Seção 3.1.4 é uma contribuição desse trabalho.

3.1.1. Remoção aleatória

O funcionamento desse operador é bem simples, o operador sorteia aleatoriamente q requisições para serem removidas da rota atual.

3.1.2. Remoção baseada em sequência

Esse operador remove uma sequência de nós consecutivos da rota. Inicialmente, uma requisição i é sorteada aleatoriamente, em seguida, $q - 1$ requisições em volta de i são removidas. Primeiro consideramos a requisição predecessora de i, e em seguida, a requisição sucessora de i. Se a requisição predecessora for o depósito inicial, então as requisições restantes consideram somente as requisições sucessoras, o raciocínio oposto é aplicado caso seja o depósito final.

3.1.3. Remoção baseada em distância

Iniciando com uma requisição escolhida aleatoriamente, a remoção das próximas requisições, baseando-se na distância, são induzidas probabilisticamente a serem requisições próximas as que já foram removidas. O parâmetro d controla a intensidade da escolha, valores altos para d induzem a remoção de requisições próximas as requisições já removidas. Seguindo a abordagem em [Côté et al., 2012b] o valor de d foi definido como 6. Outros valores para d foram testados, entretanto 6 permaneceu fornecendo os melhores resultados. Exibimos um pseudocódigo do operador no Algoritmo 1. No pseudocódigo, denotamos por S a solução atual.

Algoritmo 1: Remoção baseada em distância

```
1  i ← Aleatório(S);
2  L ← {i};
3  while |L| < q do
4  |   i ← Aleatório(L);
5  |   Temp ← ∅;
6  |   for cada requisição j ∈ S do
7  |   |   b_j ← c_ij + c_{n+i,n+j};
8  |   |   Temp ← Temp ∪ {j};
9  |   end
10 |   Ordene Temp em ordem crescente a partir dos valores b_j;
11 |   q ← Aleatório(0, 1);
12 |   Selecione o nó j na posição ⌈|Temp| * q^d⌉ de Temp;
13 |   L ← L ∪ {j};
14 |   S ← S \ {j, n + j};
15 end
16 return L;
```

3.1.4. Remoção baseada em custo

O operador remove as q requisições com os maiores custos de serem visitadas na solução atual. Sejam $pre(u)$ e $suc(u)$, respectivamente, o predecessor e o sucessor do nó u na rota. Enquanto q requisições não tiverem sido removidas da rota, selecionamos a requisição i com o maior custo de acordo com a expressão (16), removemos i da rota atual, e continuamos iterando até que q requisições sejam removidas.

$$c_{pre(i),i} + c_{i,suc(i)} + c_{pre(n+i),n+i} + c_{n+i,suc(n+i)} - c_{pre(i),suc(i)} - c_{pre(n+i),suc(n+i)} \tag{16}$$

3.2. Operadores de inserção

Após a destruição parcial da rota com os operadores de remoção, precisamos reconstruí-la com os operadores de inserção. Diferente de outros problemas onde todas as posições da rota são viáveis para a inserção de um nó, como por exemplo, no Problema do Caixeiro Viajante, o PDTSPMS impõe várias restrições na estrutura de um rota. Ao inserir um nó na rota, precisamos garantir que as restrições de precedência, política LIFO, e capacidade, sejam respeitadas. Para lidar com a precedência ao inserir uma requisição na rota, basta que a entrega $n+i$ seja inserida em uma posição depois da posição da coleta i. Entretanto, para os outros aspectos não é tão simples.

Assuma que uma requisição i seja colocada na pilha k, definimos um bloco $B_k(i, n+i)$ como o caminho da coleta i até a entrega $n+i$ na rota atual. O bloco $B_k(i, n+i)$ é composto se contém pelo menos um bloco $B_k(j, n+j)$ entre i e $n+i$, e simples se não existe nenhum bloco $B_k(j, n+j)$. Duas requisições i e j na pilha k respeitam a política LIFO se os blocos $B_k(i, n+i)$ e $B_k(j, n+j)$ não possuem nenhum vértice em comum, ou se um dos blocos está contido completamente no outro. Caso contrário, os blocos se coincidem parcialmente e a política LIFO é violada. Sem perda de generalidade, assuma que o nó i foi visitado antes de j. Nesse caso j será carregado acima de i na pilha k, entretanto, como os blocos possuem sobreposição, $n+i$ será entregue antes de $n+j$, apesar de j estar acima de i na pilha k, consequentemente, a política é violada.

Após a inserção da coleta i, as posições para inserir $n+i$ que ainda satisfazem a política LIFO passam a ser restritas. Claramente, a posição imediatamente após i é viável para $n+i$. Além disso, se i e o vértice atual não estão na mesma pilha, então $n+i$ pode ser inserido após o vértice em questão. Em seguida, as posições são avaliadas de acordo com os seguintes critérios:

- Se o veículo visita uma entrega $n+j$ tendo visitado sua coleta correspondente j depois de i, então $n+i$ não pode ser inserido entre j e $n+j$
- Se o veículo visita uma entrega $n+j$ sem ter visitado sua coleta correspondente j depois de i, então $n+i$ não pode ser inserido após este $n+j$

Para manter a capacidade viável procedemos de duas formas, de acordo com o nó ser uma coleta ou entrega. Se a carga da pilha k na posição atual mais a demanda da coleta a ser inserida forem maiores que a capacidade Q, então o nó não pode ser inserido nessa posição. Se for uma coleta, a busca continua pela rota atual em busca de novas posições viáveis. Caso contrário, se for uma entrega, a busca para pois a entrega não pode ser inserida além dessa posição, pois a partir desse ponto a capacidade da pilha será excedida. Observe que a posição da entrega na rota depende da posição da coleta fixada previamente, dessa forma, quando procuramos a posição para inserir a entrega, a demanda da coleta está no veículo. Ressaltamos que [Côté et al., 2012b] não discutem como manter a capacidade viável durante a execução dos operadores de inserção.

Posto a discussão anterior, agora podemos descrever os operadores de inserção. Os operadores nas Seções 3.2.1 e 3.2.1 foram propostos em [Côté et al., 2012b]. O operador na Seção 3.2.3 é uma contribuição desse trabalho.

3.2.1. Requisição de menor custo

Esse operador insere uma requisição, isto é, as localizações i e $n+i$, nas posições que resultam no menor custo na rota atual. Formalmente, o custo para inserir uma requisição i é definido de acordo com a expressão (16). Dentro do arcabouço da heurística LNS, quando esse operador é sorteado para reinserir os nós removidos, em cada iteração é inserido o nó que incorre no menor custo entre todos os nós disponíveis, e assim sucessivamente até que todos os nós tenham sido inseridos.

3.2.2. Inserção baseada em arrependimento

Assuma que o custo quando a requisição i é colocada na pilha $l \in M$ seja δ_{il}, e que o custo mínimo aconteça quando a requisição é colocada na pilha $k \in M$, dado por δ^*_{ik}. Então o arrependimento a_i de uma requisição i é dado por:

$$a_i = \sum_{l=1}^{m} (\delta_{il} - \delta^*_{ik}) \tag{17}$$

A requisição escolhida para ser inserida é aquela com o maior arrependimento. Essa requisição é colocada na melhor posição e na melhor pilha. A ideia desse operador é diminuir o efeito de escolhas ruins que possam ser realizadas pelos outros operadores.

3.2.3. Inserção aleatória

Diferente dos outros operadores de inserção que usam uma métrica de custo para decidir qual nó inserir primeiro. Quando esse operador é sorteado, os nós removidos da rota são reinseridos em uma ordem aleatória na rota. A posição onde o nó será inserido na rota é decidida baseada no custo.

3.3. Algoritmo

O esquema geral da heurística LNS proposta é descrito a seguir no Algoritmo 2. Denotamos por S e S^*, respectivamente, a solução atual e a melhor solução obtida. Além disso, indicamos por f a função objetivo.

A solução inicial é obtida com o operador baseado na requisição de menor custo. O critério para aceitar piorar uma solução na linha 13 é baseado na meta-heurística *Simulated Annealing*. Definimos a temperatura inicial T com o valor de $1.05f(S)$, onde S é a solução inicial, o fator de redução da temperatura foi definido como 0.99975. Escolhemos aleatoriamente no intervalo $[min(10, 0.45n), min(35, 0.75n)]$ a quantidade q de nós para remover em cada iteração. O número máximo de iterações é definido como $50,000$. Além dos operadores 3.1.4 e 3.2.3 propostos nesse trabalho, também são contribuições originais os mecanismos de busca local em cada iteração da heurística.

A busca local na linha 7 funciona da seguinte forma. Inicialmente, percorremos a rota buscando trocar duas requisições i e j adjacentes carregadas na mesma pilha. O nó i troca de posição com o nó j, e o nó $n+i$ troca de posição com o nó $n+j$. Note que, se o veículo visita em seguida duas coletas i e j carregadas na mesma pilha, então as requisições i e j podem ser trocadas de posição desde que a capacidade ainda continue viável. A troca só pode ocorrer se no subintervalo de $n+j$ até $n+i$ a carga na pilha k não aumentar, consequentemente, quem é descarregado primeiro depois da troca tem que ter uma demanda maior, isto é $d_i \geq d_j$. Caso a carga da pilha aumente no subintervalo, poderíamos checar o subintervalo para garantir a viabilidade da

https://proceedings.science/p/106901?lang=pt-br

Algoritmo 2: Large Neighborhood Search

```
1  Crie uma solução inicial S;
2  S* ← S;
3  for i ← 0 to max_iteracoes do
4  |  Selecione um número q de nós para remover;
5  |  Selecione aleatoriamente um operador de remoção e um operador de inserção;
6  |  Aplique os operadores em S para obter S';
7  |  Realize uma busca local em S' procurando trocar vizinhos adjacentes;
8  |  Realize uma busca local em S' procurando mover blocos na rota;
9  |  if f(S') < f(S) then
10 |  |  S ← S';
11 |  |  if f(S') < f(S*) then
12 |  |  |  S* ← S';
13 |  if f(S') ≥ f(S) then
14 |  |  Escolha um número aleatório w uniformemente de [0,1];
15 |  |  if w < e^{-(f(S')-f(S))/T} then
16 |  |  |  S ← S';
17 |  Reduza temperatura T;
18 end
19 return S;
```

capacidade, entretanto, a complexidade computacional do procedimento aumentaria, deixando de ser linear. Além da troca anterior, se uma localização v é visitada logo após uma requisição u, e ambas as requisições estão em pilhas diferentes, então podemos trocar u e v de posições.

Se um bloco $B_k(j, n+j)$ está contido em um bloco $B_k(i, n+i)$, então trocá-los não remove a viabilidade do problema. Além disso, o bloco $B_k(i, n+i)$ contendo somente i e $n+j$ também pode ser movido ao longo da rota, com a devida atenção à capacidade, e a mesma ainda permanecerá viável, a busca na linha 8 tenta melhorar a rota através desse fato, movendo uma requisição ao longo da rota e checando se o custo é melhorado com tal modificação.

4. Experimentos

Os algoritmos desse trabalho foram implementados em C++ e compilados com o compilador gcc $4.8.2$ com a *flag* -O3 ativada. Os experimentos foram realizados em um i7-4790K 4.00GHz com um total de 16GB de memória RAM disponível rodando o sistema operacional Ubuntu 16.04 LTS. O código-fonte de [Côté et al., 2012b] foi gentilmente compartilhado conosco. Ele foi compilado e executado no mesmo ambiente computacional.

Para avaliar a eficácia da heurística proposta, usamos o conjunto de instâncias de *benchmark* proposto por [Côté et al., 2012b]. Os autores as geraram com base nas instâncias do PDTSPL em [Carrabs et al., 2007a,b; Cordeau et al., 2010]. Foram geradas duas classes de famílias. Cada classe tem um total de 54 instâncias, divididas igualmente em 9 famílias. Na primeira classe, C1, a demanda de cada requisição é 1, o número de pilhas é um número aleatório entre 2 e 4, e a capacidade de cada pilha é um número aleatório entre 1 e 3. Na segunda classe, C2, a demanda de cada requisição é um número aleatório entre 1 e 10, o número de pilhas é um número aleatório entre 2 e 4, e a capacidade é um número aleatório entre 10 e 15.

Cada heurística foi executada um total de 10 vezes para cada instância. Nas tabelas 1 e 2 são reportados para cada instância o valor médio da solução encontrada (Solução$_\mu$), o valor da melhor solução encontrada (Melhor Solução), e o tempo médio de CPU em segundos (Tempo$_\mu$). Para cada instância fornecemos o nome (Nome), número de requisições (n), e, se conhecido, o ótimo (Opt). O total de vértices de uma instância é dado por $2n+2$, devido aos depósitos inicial e final. As instâncias que não possuem o ótimo conhecido possuem n.a. (não atribuído).

Instância			Côté et al. [2012b]			LNS proposta		
Nome	n	Opt	Solução$_\mu$	Melhor Solução	Tempo$_\mu$	Solução$_\mu$	Melhor Solução	Tempo$_\mu$
a280	11	449	450,1	449	4.17	449	449	0.48
	13	477	514,6	488	4.68	468	**468**	0.69
	15	542	605,4	582	5.70	542	**542**	0.81
	17	624	656	633	7.12	624	**624**	0.96
	19	669	704,3	678	8.82	669	**669**	1.14
	21	739	775,9	759	10.36	741,1	**739**	1.38
att532	11	4177	4177	4177	11.93	4177	4177	0.63
	13	4937	4937	4937	15.34	4937	4937	0.76
	15	5151	5151	5151	18.96	5151	5151	1.00
	17	5294	5294	5294	20.32	5294	5294	1.19
	19	5587	5587	5587	27.00	5587	5587	1.47
	21	9266	9266	9266	33.03	9266	9266	1.77
brd14051	11	4396	4396	4396	4.85	4396,2	4396	0.49
	13	4439	4441,4	4439	5.69	4440,5	4439	0.64
	15	n.a.	4819	4807	5.76	4818,7	**4797**	0.83
	17	n.a.	4952,3	4945	6.56	4927,3	**4922**	0.99
	19	n.a.	6693,1	6536	9.59	6596,8	**6479**	1.30
	21	n.a.	6882,4	**6733**	11.83	6894,4	6794	1.41
d15112	11	74,603	74603	74603	6.92	75418,9	74603	0.62
	13	80,690	80987,5	80690	8.68	81812,7	80690	0.81
	15	89,754	89754	**89754**	9.69	91645,5	90099	1.06
	17	96,804	96977,3	96804	11.34	98514,1	96804	1.29
	19	n.a.	103635	103356	14.43	104158	103476	1.59
	21	n.a.	109097	108407	17.68	109800	108081	1.73
d18512	11	4280	4280	4280	6.40	4283,5	4280	0.49
	13	4301	4301	4301	7.97	4302,2	4301	0.64
	15	4638	4638	**4638**	10.05	4647,3	4644	0.83
	17	4741	4741	**4741**	11.15	4811,5	4781	1.01
	19	n.a.	4917	**4917**	12.07	5003,6	4992	1.34
	21	n.a.	5100	**5100**	15.79	5214,7	5123	1.41
fnl4461	11	1889	1889	1889	8.29	1889	1889	0.67
	13	2088	2088	2088	8.63	2088	2088	0.86
	15	2356	2356	2356	10.91	2356	2356	1.15
	17	2517	2517	2517	12.08	2517	2517	1.35
	19	n.a.	2933	2933	14.49	2933	2933	1.72
	21	n.a.	3561,7	3561	19.49	3561	3561	1.96
nrw1379	11	2690	2690	2690	7.96	2690	2690	0.61
	13	n.a.	3061	3061	8.57	3061	3061	0.79
	15	n.a.	3119,9	3117	9.58	3117,7	3117	0.99
	17	n.a.	3218,9	3197	11.94	3197	3197	1.19
	19	n.a.	3492	3439	13.04	3438,1	**3430**	1.47
	21	n.a.	3777,1	3691	16.35	3741,3	**3651**	1.77
pr1002	11	13,718	13718	13718	4.66	13718	13718	0.50
	13	15,436	15436	15436	5.73	15436	15436	0.63
	15	16,268	16268	16268	6.89	16268	16268	0.79
	17	17,601	17601	17601	8.80	17601	17601	1.02
	19	18,673	18679	18673	10.65	18685,4	18673	1.13
	21	20,150	20171	20159	14.60	20158,1	20150	1.31
ts225	11	22,000	22000	22000	5.10	22000	22000	0.49
	13	29,395	29437,2	29395	5.77	29395	29395	0.65
	15	32,541	32541	32541	6.87	32541	32541	0.80
	17	36,405	36405	36405	8.78	36486,6	36405	0.95
	19	40395	40395	40395	10.93	40395	40395	1.14
	21	42878	42990,7	42878	14.07	42878	42878	1.42

Tabela 1: Resultados computacionais para o conjunto de instâncias C1.

Instância			Côté et al. [2012b]			LNS proposta		
Nome	n	Opt	Solução$_\mu$	Melhor Solução	Tempo$_\mu$	Solução$_\mu$	Melhor Solução	Tempo$_\mu$
a280	11	455	455	455	5.74	455	455	0.46
	13	479	490,8	479	6.65	479	479	0.61
	15	553	590,5	553	7.92	553	553	0.78
	17	635	669,4	639	10.97	635	**635**	0.95
	19	677	716,5	712	14.90	677	**677**	1.23
	21	753	793	793	18.35	753	**753**	1.33
att532	11	4190	4190	4190	6.68	4190	4190	0.49
	13	5033	5033	5033	8.98	5034,8	5033	0.63
	15	5665	5666,8	5665	10.21	5674,6	5665	0.81
	17	n.a.	5915,7	5865	12.85	5897,3	5865	0.96
	19	n.a.	6193	6184	16.54	6255,8	6184	1.17
	21	n.a.	10008,3	9998	20.24	10168,9	9998	1.37
brd14051	11	4386	4386	4386	7.88	4386	4386	0.59
	13	4458	4458,4	4458	11.10	4458	4458	0.76
	15	n.a.	4796,2	**4795**	13.50	4817,9	4796	1.01
	17	n.a.	4891	4890	17.08	4889	**4889**	1.21
	19	n.a.	6275,6	6274	20.11	6274	6274	1.49
	21	n.a.	6321,6	6320	26.54	6335,9	6320	1.71
d15112	11	73,872	73872	73872	9.83	73872	73872	0.62
	13	81,657	81657	81657	12.73	81718,8	81657	0.85
	15	91,799	91799	91799	15.01	92133,4	91799	1.05
	17	n.a.	97063,7	97040	19.27	97489,2	97040	1.24
	19	n.a.	99729	99729	25.45	99729	99729	1.50
	21	n.a.	104890	104213	31.70	105581	104213	1.75
d18512	11	4341	4341	4341	6.83	4341	4341	0.50
	13	4572	4572	4572	8.30	4580	4572	0.64
	15	4893	4893	4893	9.97	4893,5	4893	0.82
	17	n.a.	5100,7	**5096**	12.17	5180,6	5179	0.98
	19	n.a.	5360,2	**5359**	14.16	5379,6	5363	1.22
	21	n.a.	5768	5768	18.45	5826,7	**5726**	1.50
fnl4461	11	1889	1883	1883	10.79	1883	1883	0.61
	13	2088	2088	2088	10.96	2088	2088	0.79
	15	2262	2263,5	**2262**	13.88	2267	2267	1.03
	17	2428	2433,9	**2428**	18.26	2454	2454	1.24
	19	n.a.	2635,5	2634	23.85	2634	2634	1.55
	21	n.a.	2773	2773	30.48	2785,6	2773	1.78
nrw1379	11	2690	2690	2690	10.39	2690	2690	0.70
	13	n.a.	3055	3055	12.03	3055	3055	0.88
	15	n.a.	3112,3	3108	15.71	3108	3108	1.12
	17	n.a.	3197	3197	18.10	3197	3197	1.36
	19	n.a.	3421,2	3421	20.30	3431,8	3421	1.64
	21	n.a.	3776,3	3769	25.45	3784,6	3769	1.95
pr1002	11	13,527	13527	13527	11.42	13527	13527	0.60
	13	15,221	15221	15221	15.86	15221	15221	0.77
	15	15,676	15676	15676	19.38	15830,8	15676	1.01
	17	17,009	17009	17009	24.33	17112,2	17009	1.20
	19	18,136	18136	18136	29.54	18164,8	18136	1.51
	21	19,613	19613	**19613**	37.23	19785	19785	1.70
ts225	11	22,000	22000	22000	7.04	22000	22000	0.49
	13	34,000	34000	34000	8.63	34000	34000	0.65
	15	37,703	37703	37703	10.05	37703	37703	0.83
	17	n.a.	41703	41703	13.66	41703	41703	1.08
	19	n.a.	45703	45703	17.25	45703	45703	1.19
	21	n.a.	49097	49097	22.60	49218,2	49097	1.42

Tabela 2: Resultados computacionais para o conjunto de instâncias C2.

https://proceedings.science/p/106901?lang=pt-br

A Tabela 1 exibe os resultados para as instâncias na classe C1. Os tempos de execução da heurística LNS proposta são bastante inferiores aos resultados da heurística proposta em [Côté et al., 2012b]. Em nenhuma instância foi gasto mais do que 2 segundos, enquanto em algumas instâncias [Côté et al., 2012b] gastou mais de vinte segundos. A partir dos resultados na Tabela 1, observa-se que em várias famílias de instâncias nossa heurística obteve as melhores soluções, e em algumas instâncias não resolvidas melhoramos a melhor solução conhecida. A melhoria desses limites superiores também pode ser usada para auxiliar algoritmos exatos existentes para o problema. Além disso, devido à sua velocidade, ela pode ser utilizada em algoritmos exatos que possuem o PDTSPMS como seu problema de *pricing* [Cherkesly et al., 2016].

Os resultados para o conjunto de instâncias C2 são dispostos na Tabela 2. A partir da tabela, observa-se que o baixo tempo de execução da heurística LNS proposta permanece na classe C2. Tanto na classe C1 quanto na classe C2 a heurística é capaz de obter resultados melhores para a família de instâncias a280. Ao comparar a média da soluções obtidas, nota-se que as heurísticas possuem resultados próximos, entretanto, devido ao baixo tempo de execução, a heurística LNS proposta se torna atrativa.

Um dos motivos do alto tempo de execução da heurística de [Côté et al., 2012b] é o algoritmo de programação dinâmica executado ao término da heurística, que enumera toda as soluções possíveis a partir da evolução das configurações das pilhas do veículo. Em nossa abordagem eliminamos a programação dinâmica com mecanismos de buscas locais em cada iteração. Durante cada iteração tentamos melhorar a solução com o uso dos operadores de vizinhança baseados em blocos.

5. Conclusão

Esse artigo abordou o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. Propusemos novos operadores e mecanismos de busca local para uma heurística *Large Neighborhood Search* da literatura. Explicamos como a heurística mantém a viabilidade e apresentamos novos operadores de vizinhança para realizar buscas locais. Os resultados computacionais mostram que a heurística obtém resultados em menor tempo de execução, e para várias instância obtém soluções melhores. Estudos futuros incluem estudar novos operadores de vizinhança para a heurística, incluindo operadores baseados em blocos de tamanho maior. Além disso, pretendemos aplicar os operadores de vizinhança propostos para problemas correlatos, como o Problema do Caixeiro Viajante Duplo e a versão do PDTSPMS onde o veículo possui uma única pilha de capacidade ilimitada.

Referências

Ángel Felipe, Ortuño, M. T., e Tirado, G. (2009). The double traveling salesman problem with multiple stacks: A variable neighborhood search approach. *Computers & Operations Research*, 36(11):2983 – 2993.

Ángel Felipe, Ortuño, M. T., e Tirado, G. (2011). Using intermediate infeasible solutions to approach vehicle routing problems with precedence and loading constraints. *European Journal of Operational Research*, 211(1):66 – 75.

Carrabs, F., Cerulli, R., e Cordeau, J.-F. (2007a). An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with lifo or fifo loading. *INFOR: Information Systems and Operational Research*, 45(4):223–238.

Carrabs, F., Cordeau, J.-F., e Laporte, G. (2007b). Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4): 618–632.

Cherkesly, M., Desaulniers, G., Irnich, S., e Laporte, G. (2016). Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and multiple stacks. *European Journal of Operational Research*, 250(3):782 – 793.

Cordeau, J.-F., Iori, M., Laporte, G., e Salazar González, J. J. (2010). A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading. *Networks*, 55(1):46–59.

Côté, J.-F., Archetti, C., Speranza, M. G., Gendreau, M., e Potvin, J.-Y. (2012a). A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(4):212–226.

Côté, J.-F., Gendreau, M., e Potvin, J.-Y. (2012b). Large neighborhood search for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(1):19–30.

Dumitrescu, I., Ropke, S., Cordeau, J.-F., e Laporte, G. (2010). The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269–305.

Hansen, P. e Mladenović, N. (2005). *Variable Neighborhood Search*, p. 211–238. Springer US, Boston, MA.

Ladany, S. P. e Mehrez, A. (1984). Optimal routing of a single vehicle with loading and unloading constraints. *Transportation Planning and Technology*, 8(4):301–306.

Levitin, G. e Abezgaouz, R. (2003). Optimal routing of multiple-load agv subject to lifo loading constraints. *Computers & Operations Research*, 30(3):397–410.

Li, Y., Lim, A., Oon, W.-C., Qin, H., e Tu, D. (2011). The tree representation for the pickup and delivery traveling salesman problem with lifo loading. *European Journal of Operational Research*, 212(3):482 – 496.

Pereira, A. H. e Urrutia, S. (2018). Formulations and algorithms for the pickup and delivery traveling salesman problem with multiple stacks. *Computers & Operations Research*, 93:1 – 14.

Petersen, H. L. e Madsen, O. B. (2009). The double travelling salesman problem with multiple stacks – formulation and heuristic solution approaches. *European Journal of Operational Research*, 198(1):139 – 147.

Sampaio, A. H. e Urrutia, S. (2017). New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 24(1-2):77–98.

Øvstebø, B. O., Hvattum, L. M., e Fagerholt, K. (2011). Routing and scheduling of roro ships with stowage constraints. *Transportation Research Part C: Emerging Technologies*, 19(6):1225 – 1242.