Uma heurística híbrida para o Problema do Caixeiro Viajante com Coleta e Entrega envolvendo um único tipo de produto

Bruno Cordeiro Paes, Anand Subramanian, Luiz Satoru Ochi
Instituto de Computação - Universidade Federal Fluminense
Rua Passo da Pátria 156 - Bloco E - 3º andar, São Domingos, CEP: 24210-240, Niterói - RJ
{bpaes, anand, satoru}@ic.uff.br

RESUMO

O Problema do Caixeiro Viajante com Coleta e Entrega envolvendo um único tipo de produto, conhecido na literatura como *One-Commodity Pickup and Delivery Traveling Salesman Problem* (1-PDTSP) é uma generalização do Problema do Caixeiro Viajante (PCV) onde clientes de entrega possuem uma demanda por uma dada quantidade de um único produto, enquanto clientes de coleta fornecem uma dada quantidade do mesmo produto. O objetivo é encontrar a rota de menor custo que satisfaça todas as restrições de demandas dos clientes e capacidade do veículo. Neste trabalho é proposto um algoritmo baseado nas metaheurísticas *Greedy Randomized Adaptative Search Procedure* (GRASP) e *Iterated Local Search* (ILS) que utiliza o método *Variable Neighborhood Descent* com ordem aleatória de vizinhanças (RVND), na fase de busca local. A heurística proposta foi testada em 100 instâncias disponíveis na literatura. Os resultados obtidos pelo algoritmo, em termos de qualidade das soluções, foram altamente competitivos, sendo capaz de encontrar 49 das 50 soluções ótimas conhecidas e melhorar todas as demais 50 soluções apontadas pela literatura.

PALAVRAS CHAVE. 1-PDTSP. Heurísticas Híbridas. *Iterated Local Search*. Metaheurísticas.

ABSTRACT

The One-Commodity Pickup and Delivery Traveling Salesman Problem (1-PDTSP) is a generalization of the well-known Traveling Salesman Problem (TSP) where each delivery customer has a demand for a given amount of a single product, while pickup customers provide a given quantity of the same product. The objective is to find the least-cost route that satisfies all the demand constraints as well as the vehicle capacity. In this work we propose an algorithm based on the metaheuristics Greedy Randomized Adaptive Search Procedure (GRASP) and Iterated Local Search (ILS) that uses the Variable Neighborhood Descent with Random Neighborhood Ordering (RVND) method in the local search phase. The proposed heuristic was tested in 100 benchmark instances available in the literature. The results found by the algorithm, in terms of solution quality, were highly competitive, being capable to find 49 of the 50 known optimal solutions and to improve all the remaining 50 solutions pointed out in the literature.

KEYWORDS. 1-PDTSP. Hybrid Heuristics. Iterated Local Search. Metaheuristics.

1. Introdução

O Problema do Caixeiro Viajante com Coleta e Entrega envolvendo um único tipo produto, conhecido na literatura como *One-Commodity Pickup and Delivery Traveling Salesman Problem* (1-PDTSP) é uma variante do Problema do Caixeiro Viajante (PCV). O 1-PDTSP foi proposto por Hernández-Pérez e Salazar-González (2004a) e a principal diferença em relação ao PCV é que cada cliente é classificado de acordo com o tipo de serviço exigido (coleta ou entrega). Cada cliente de entrega (*delivery customer*) possui uma demanda por uma dada quantidade de um determinado produto, enquanto clientes de coleta (*pickup customer*) fornecem uma dada quantidade deste mesmo produto. O 1-PDTSP consiste em encontrar a rota de menor custo que satisfaça todas as restrições de demandas dos clientes e não exceda a capacidade do veículo. Desta forma, o veículo sai de um ponto inicial, denominado de depósito, percorre todos os clientes e retorna ao ponto de partida. Uma importante característica deste problema é que o produto coletado em um determinado cliente de coleta pode ser fornecido para qualquer outro cliente de entrega.

Esta variante possui diversas aplicações reais. Dentre elas, tem-se uma aplicação no contexto de reposicionamento de estoque. Considere uma rede varejista localizada em diferentes regiões. Devido a natureza randômica das demandas, algumas dessas filiais podem ter um excesso de um dado produto no estoque enquanto outras necessitam do mesmo produto. Desta forma, a matriz pode decidir transferir uma dada quantidade do produto entre as filiais, objetivando obter um balanceamento entre seus estoques. Outra aplicação é na movimentação de quantias de dinheiro entre bancos, há somente um único tipo de produto envolvendo diferentes fontes e destinos.

O 1-PDTSP é $\mathcal{NP}$-difícil pois pode ser reduzido ao PCV quando a capacidade do veículo é suficientemente grande. Além disso, o problema de verificar a existência de uma solução viável é $\mathcal{NP}$-Completo, ou seja, encontrar uma rota viável pode ser considerada uma tarefa bastante complexa.

Hernández-Pérez e Salazar-González (2004a) desenvolveram um algoritmo *branch-and-cut* para resolver instâncias com até 60 clientes. Um conjunto de desigualdades para o 1-PDTSP e uma nova versão do algoritmo *branch-and-cut* foram sugeridos por Hernández-Pérez e Salazar-González (2007). Hernández-Pérez e Salazar-González (2004b) propuseram duas heurísticas onde a primeira é baseada em um algoritmo guloso com um critério de k-otimalidade e a segunda é fundamentada no procedimento exato desenvolvido em Hernández-Pérez e Salazar-González (2004a). Os mesmos autores propuseram um algoritmo que combina as metaheurísticas GRASP e VND (Hernández-Pérez e Salazar-González, 2009).

Recentemente, Zhao *et al.* (2009) apresentaram um Algoritmo Genético (AG) composto por uma nova heurística construtiva para gerar a população inicial e um procedimento de busca local para acelerar a convergência do algoritmo. Os testes computacionais mostraram que o AG consegue, na maioria das instâncias, melhorar os resultados da literatura.

Este trabalho propõe um algoritmo baseado nas metaheurísticas *Greedy Randomized Adaptative Search Procedure* (GRASP) e *Iterated Local Search* (ILS), além de utilizar o *Variable Neighborhood Descent* com ordem aleatória de vizinhanças (RVND) como método de busca local.

O restante do trabalho está estruturado da seguinte maneira. A Seção 2 apresenta a definição do 1-PDTSP e sua formulação matemática. A Seção 3 descreve o algoritmo proposto. A Seção 4 contém os resultados obtidos. A Seção 5 apresenta as considerações finais deste trabalho.

2. Definição do Problema

Seja $G = (V, A)$ um grafo completo e orientado, onde o conjunto de vértices $V = \{1, \ldots, n\}$ é o conjunto de clientes, e $A = \{(i, j) : i, j \in V\}$ é o conjunto de arcos. Os vértices $i \in V$ representam os clientes, cada um com uma demanda q_i associada, sendo $q_i < 0$ para clientes de entrega e $q_i > 0$ para clientes de coleta. Para cada par de localizações $\{i, j\}$, a distância (ou custo) c_{ij} de viagem é fornecido. Considere Q como sendo a capacidade

do veículo, onde tipicamente $Q \leq max\{\sum_{i \in V:q_i>0} q_i, -\sum_{i \in V:q_i<0} q_i\}$. Além disso, qualquer cliente pode ser considerado um depósito que fornece ou absorve uma quantidade de produtos, como apresentado na eq. (1).

$$q_1 = -\sum_{i=2}^{n} q_i. \tag{1}$$

Assim sendo, a conservação do fluxo entre os clientes é satisfeita, como pode ser observado na eq.(2).

$$\sum_{\forall i \in V:q_i>0} q_i - \sum_{\forall i \in V:q_i<0} q_i = 0. \tag{2}$$

Seja x_{ij} uma variável binária que indica se o arco $(i,j) \in A$ faz parte da solução ($x_{ij} = 1$) ou não ($x_{ij} = 0$) e f_{ij} uma variável não-negativa que representa o fluxo do arco $(i,j) \in A$. A seguir é apresentada uma formulação matemática para o 1-PDTSP, proposta por Hernández-Pérez e Salazar-González (2004a).

$$Min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{3}$$

sujeito a:

$$\sum_{j \in V, j \neq i} x_{ij} = 1 \qquad \forall j \in V \tag{4}$$

$$\sum_{i \in V, i \neq j} x_{ij} = 1 \qquad \forall i \in V \tag{5}$$

$$\sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geqslant 1 \qquad \forall S \subset V \tag{6}$$

$$\sum_{j \in V, j \neq i} f_{ij} - \sum_{j \in V, j \neq i} f_{ji} = q_i \qquad \forall i \in V \tag{7}$$

$$0 \leqslant f_{ij} \leqslant Q x_{ij} \qquad \forall (i,j) \in A \tag{8}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{9}$$

A função objetivo (3) minimiza a soma do custo de viagem. As restrições (4) e (5) indicam que cada cliente dever ser visitado exatamente uma vez. Restrições (6) são as desigualdades que proíbem a formação de rotas desconexas da origem. As restrições (7) garantem a conservação do fluxo. Restrições (8) e (9) definem o domínio das variáveis.

3. Algoritmo Proposto

Esta seção apresenta o algoritmo proposto para o 1-PDTSP, composto pelas componentes das metaheurísticas GRASP (Resende e Ribeiro, 2003) e ILS (Lourenço et al., 2002), bem como o procedimento RVND. Como pode ser observado no Alg. 1, inicialmente um pré-processamento é realizado com o intuito de tentar reduzir a dimensão do espaço de busca (linha 3). Para tanto, utiliza-se uma Lista Reduzida de Vizinhos (LRV) de cada cliente associada a cada vizinhança que será tratada em detalhes na Subseção 3.4. A heurística *multi-start* executa $MaxIter$ iterações (linhas 4 a 28), onde cada solução inicial é gerada por um método baseado na fase de construção do GRASP (linha 5). Em seguida, o laço principal do ILS (linhas 8 a 23) procura melhorar esta

solução inicial usando um procedimento RVND (linha 9) na fase de busca local combinado com um mecanismo de pertubação (linha 14). Caso não tenha sido possível gerar uma solução viável durante a fase de pertubação ou o número máximo de pertubações consecutivas sem melhora ($MaxIterILS$) tenha sido atingido, executa-se novamente o procedimento RVND com a Lista Reduzida de Vizinhos Complementar ($\overline{\text{LRV}}$), onde LRV $\cap$ $\overline{\text{LRV}} = \emptyset$ e LRV $\cup$ $\overline{\text{LRV}}$ = LV, onde LV é a Lista Completa de Vizinhos. Em caso de melhora reinicia-se o contador de pertubações consecutivas ($iterILS$) (linhas 17 a 19).

Algoritmo 1 GILS-RVND

1: Procedimento GILS-RVND($MaxIter, MaxIterILS$)
2: $f^* \leftarrow \infty$;
3: LRV $\leftarrow$ Preprocessamento();
4: **for** $k \leftarrow 1, ..., MaxIter$ **do**
5: $s \leftarrow$ GeraSolucaoInicial(seed);
6: $s' \leftarrow s$;
7: $iterILS \leftarrow 0$;
8: **while** $iterILS < MaxIterILS$ **do**
9: $s \leftarrow$ RVND($N(.)$, LRV, $f(.)$, r, s) {r = n^0 de vizinhanças}
10: **if** $f(s) < f(s')$ **then**
11: $s' \leftarrow s$;
12: $iterILS \leftarrow 0$;
13: **end if**;
14: $s \leftarrow$ Perturba(s');
15: **if** ($s = s'$ ou $iterILS = MaxIterILS - 1$) **then**
16: $s \leftarrow$ RVND($N(.)$, $\overline{LRV}$, $f(.)$, r, s')
17: **if** $f(s) < f(s')$ **then**
18: $s' \leftarrow s$;
19: $iterILS \leftarrow 0$;
20: **end if**;
21: **end if**;
22: $iterILS \leftarrow iterILS + 1$;
23: **end while**;
24: **if** $f(s') < f^*$ **then**
25: $s^* \leftarrow s'$;
26: $f^* \leftarrow f(s')$;
27: **end if**;
28: **end for**;
29: return s^*;
30: end GILS-RVND.

3.1 Procedimento Construtivo

O procedimento construtivo empregado para gerar uma solução viável envolve uma abordagem gulosa randomizada baseada na fase construtiva do GRASP. O pseudocódigo do procedimento construtivo é descrito no Alg. 2. Primeiramente inicializam-se a solução s a ser construída e a Lista dos Candidatos (LC) (linhas 2 e 3). No laço principal (linhas 4 a 10), todos os clientes pertencentes a LC são avaliados de acordo com um critério de inserção mais barata modificado (linha 5), onde somente os candidatos com $g(e) \in [g^{min}, g^{min} + \alpha(g^{max} - g^{min})]$ farão

parte da Lista Restrita de Candidatos (LRC) (linha 6). Após esta etapa, um novo cliente é selecionado aleatoriamente da LRC e adicionado a solução s. O laço principal termina quando todos os clientes tiverem sido inseridos ou nenhum cliente puder ser adicionado a solução parcial sem que as restrições de capacidade sejam violadas. Caso não tenha sido possível gerar uma solução completa viável, o processo de construção é reiniciado (linhas 11 a 13).

O custo de inserção de um dado cliente $e \in$ LC (linha 5) na solução parcial s é avaliado de acordo com a eq. (10), onde $g(e)$ representa o custo de inserção. O valor de $g(e)$ é composto da soma de duas parcelas. A primeira é o custo de inserção de um cliente e entre todos os pares adjacentes de clientes i e j, enquanto a segunda corresponde a uma bonificação para favorecer a inserção de clientes que possuam maior demanda. A diferença entre a carga do veículo após visitar o cliente i ($\mathcal{L}_i$) e a demanda do cliente e (d_e) é ponderado por um fator γ, escolhido aleatoriamente a partir do seguinte intervalo $\{0.00, 0.05, 0.01, \ldots, 1.70\}$. Este intervalo foi definido através de experimentos preliminares.

$$g(e) = (c_{ie} + c_{ej} - c_{ij}) + \gamma((\mathcal{L}_i) - |d_e|) \tag{10}$$

Algoritmo 2 GeraSolucaoInicial

1: Procedimento GeraSolucaoInicial(seed)
2: $s \leftarrow \emptyset$;
3: Inicializar lista de candidatos (LC);
4: **while** LC $\neq \emptyset$ e no mínimo um cliente $e \in$ LC pode ser adicionado a s **do**
5: Avalie o custo de inserção $g(e)$ de $e \in$ LC;
6: LRC $\leftarrow g(e) \in [g^{min}, g^{min} + \alpha(g^{max} - g^{min})]$
7: selecione randomicamente $e \in$ LRC;
8: $s \leftarrow s \cup \{e\}$;
9: Atualize LC;
10: **end while**;
11: **if** (LC > 0) **then**
12: **Go to** line 2;
13: **end if**;
14: return s;
15: end GeraSolucaoInicial.

3.2 Busca Local

A fase de busca local é realizada por um método baseado no procedimento *Variable Neighborhood Descent* (VND) (Mladenović e Hansen, 1997), utilizando ordem aleatória da estrutura de vizinhança (RVND). Basicamente, o RVND consiste em selecionar de maneira aleatória a ordem de execução das estruturas pertencentes ao conjunto $N = \{N^1, N^2, N^3, \ldots, N^{(\eta)}\}$ de vizinhanças. Testes preliminares mostraram que esta abordagem, em média, encontra melhores resultados quando comparados com a versão com ordem determinística.

O algoritmo proposto possui um conjunto de cinco estruturas de vizinhanças amplamente exploradas na literatura, a saber:

- **Or-*opt*** – $N^{(1)}$ – Um único cliente é removido e inserido em uma outra posição da rota.
- **Or-*opt2*** – $N^{(2)}$ – Dois clientes adjacentes são removidos e inseridos em outra posição da rota.

- **Or-*opt3*** – $N^{(3)}$ –Três clientes adjacentes são removidos e inseridos em outra posição da rota.
- **2-*opt*** – $N^{(4)}$ – Dois arcos não adjacentes são removidos e outros dois são adicionados para formar uma nova rota.
- ***Swap*** – $N^{(5)}$ – Permutação entre dois clientes.

Cada estrutura de vizinhança é examinada exaustivamente e somente movimentos viáveis são admitidos, ou seja, aqueles que não violam as restrições de capacidade do veículo. Assim sendo, em caso de melhora, deve-se verificar se a nova solução é viável ou não. Para todas as estruturas de vizinhança a carga é verificada a partir da primeira posição modificada da rota. A complexidade desta checagem é da ordem de $\mathcal{O}(n)$.

O pseudocódigo do procedimento RVND é descrito no Alg 3. Inicialmente, a Lista de Vizinhanças (LN) é inicializada com as estruturas de vizinhanças (linha 2). No laço das linhas 3 a 13, uma estrutura de vizinhança $N^{(\eta)}$ é selecionada aleatoriamente de LN (linha 4) e então o melhor vizinho encontrado por esta estrutura é armazenado em s' (linha 5). Em caso de melhora, preenche-se a LN com todas as estruturas de vizinhanças (linhas 6 a 10). Caso contrário, $N^{(\eta)}$ é removida de LN (linha 11). O procedimento termina quando a LN estiver vazia.

Algoritmo 3 RVND

1: Procedimento RVND$(N(.), f(.), r, s)$
2: Inicializar a Lista de Vizinhanças (LN);
3: **while** LN $\neq 0$ **do**
4: Escolha a vizinhança $N^{(\eta)} \in$ LN randomicamente;
5: Encontre o melhor vizinho s' of $s \in N^{(\eta)}$;
6: **if** $f(s^{'}) < f(s)$ **then**
7: $s \leftarrow s^{'}$;
8: $f(s) \leftarrow f(s^{'})$;
9: Atualiza LN;
10: **else**
11: Remova $N^{(\eta)}$ de LN;
12: **end if**;
13: **end while**;
14: return s;
15: end RVND.

3.3 Mecanismos de Pertubação

O procedimento *double-bridge*, proposto por Martin et al. (1991), foi originalmente desenvolvido para o TSP e consiste na remoção de quatro arcos de uma determinada rota e na inserção de outras quatro de tal maneira a gerar uma nova rota. Este movimento também pode ser visto como uma permutação entre dois seguimentos disjuntos de rotas. O pseudocódigo do procedimento Perturba() é descrito no Alg. 4. Inicialmente, considere $MaxSize$ e $MaxTrial$ como sendo, respectivamente, o número máximo de clientes consecutivos e o número máximo de tentativas consecutivas de pertubar s' (linhas 2 e 3). Experimentos preliminares indicaram que: (i) para instâncias onde $n \leq 60$, dificilmente foi possível gerar soluções viáveis adotando $MaxSize > 2$, enquanto para $n > 60$ com $MaxSize > 5$, verificou-se um alto índice de inviabilidade nas soluções perturbadas; (ii) para instâncias maiores ($n > 60$) adotou-se $MaxTrial = n$, enquanto para as demais instâncias considerou-se $MaxTrial = 4N$, pois, nestes casos, n tentativas demonstraram ser insuficientes para gerar soluções viáveis (linhas 4 a 10). O processo de pertubação propriamente dito é executado enquanto $MaxSize > 2$ e s for inviável (linhas 11 a 21). A pertubação *double-bridge* em si é realizada $MaxTrial$ vezes até que uma solução viável seja gerada (linhas 13 a 17).

Algoritmo 4 Perturba

1: Procedimento Perturba(s')
2: Considere $MaxSize$ como sendo número máximo de clientes consecutivos;
3: Considere $MaxTrial$ como sendo o número máximo de tentativas consecutivas de pertubar s';
4: **if** ($n > 60$) **then**
5: $MaxSize \leftarrow 5$;
6: $MaxTrial \leftarrow n$;
7: **else**
8: $MaxSize \leftarrow 2$;
9: $MaxTrial \leftarrow 4n$;
10: **end if**;
11: **while** ($MaxSize \geq 2$ e s for inviável) **do**
12: $CounterTrial \leftarrow 0$;
13: **while** (s não for inviável ou $CounterTrial \neq MaxTrial$) **do**
14: Selecione aleatoriamente 2 segmentos de rotas disjuntos A e B de s' contendo até $MaxSize$ clientes consecutivos;
15: $s \leftarrow$ Permute os segmentos A e B de s';
16: $CounterTrial \leftarrow CounterTrial + 1$;
17: **end while**
18: **if** (s não for viável) **then**
19: $MaxSize \leftarrow MaxSize - 1$;
20: **end if**;
21: **end while**;
22: return s;
23: end Perturba.

3.4 Fase de Pré-processamento

Nesta fase utiliza-se um procedimento para tentar eliminar movimentos que possivelmente nunca farão parte de uma solução viável, além de tentar reduzir de forma heurística e exata o espaço de busca das vizinhanças. Este processo é realizado com o intuito de diminuir o esforço computacional durante a fase de busca local.

O seguinte critério exato de eliminação de arcos que nunca farão parte de uma solução viável foi adotado: Dado os clientes i e $j \in A$, se $q_i + q_j > Q$ ou $q_i + q_j < 0$ então o arco (i, j) é eliminado do grafo G. Desta forma, as vizinhanças $N^{(1)}, N^{(2)}, N^{(3)}$ sequer avaliarão movimentos envolvendo arcos inválidos.

Com relação as vizinhanças $N^{(4)}, N^{(5)}$ adotou-se o seguinte critério exato de redução do espaço de busca. Dado os clientes i e $j \in A$, se ($q_i \leq 0$ e $q_j \leq Q + q_i$) ou ($q_i > 0$ e $q_j \geq q_i - Q$), então o movimento envolvendo estes clientes não será considerado para avaliação.

Visando reduzir ainda mais o espaço de busca das vizinhanças utilizou-se um critério heurístico que leva em consideração o custo entre os clientes i e j, Para cada cliente $i \in V$ calcula-se o custo médio $AvgCusto_i$ entre o mesmo em relação aos demais clientes $j \in V, i \neq j$. Se $c_{i,j} > AvgCusto_i$, então o arco (i, j) não é considerado como sendo promissor durante a fase de busca local.

Para cada par $(i, \eta), i \in V, \eta \in N$ tem-se uma lista reduzida de vizinhos LRV_i^η geradas a partir dos critérios de redução. A LRV é composta pela união destas listas reduzidas, isto é, $\text{LRV} = \bigcup_{i \in V, \eta \in N} \{\text{LRV}_i^\eta\}$.

O pseudocódigo do pré-processamento é descrito no Alg 5. Inicialmente, para cada cliente $i \in V$ armazena-se a média de suas distâncias em relação a todo cliente $j \in V$, $i \neq j$, e inicializam-se LRV_i^η e $\overline{\text{LRV}}_i^\eta$, $i \in V, \eta \in N$ (linhas 2 a 8). As listas reduzidas (LRV_i^η) e complementares ($\overline{\text{LRV}}_i^\eta$) são preenchidas de acordo com critérios de

redução supracitados (linhas 9 a 21). A complexidade da fase de pré-processamento é da ordem de $\mathcal{O}(|N|n^2)$.

Algoritmo 5 Preprocessamento

1: Procedimento Preprocessamento()
2: **for** cada cliente $i \in V$ **do**
3: $AvgCusto[i] \leftarrow$ distância média entre i e todo $j \in V, i \neq j$;
4: **for** cada vizinhança $\eta \in N$ **do**
5: $\text{LRV}^{\eta}[i] \leftarrow \emptyset$;
6: $\overline{\text{LRV}}^{\eta}[i] \leftarrow \emptyset$;
7: **end for**;
8: **end for**;
9: **for** cada vizinhança $\eta \in N$ **do**
10: **for** cada par de clientes $(i, j) \in A, i \neq j$ **do**
11: **if** (i, j) não foram eliminados do espaço de busca de η **then**
12: **if** $(n > 60)$ **then**
13: **if** $(c_{ij} < AvgCusto[i])$ **then**
14: $\text{LRV}^{\eta}[i] \leftarrow j$
15: **else**
16: $\overline{\text{LRV}}^{\eta}[i] \leftarrow j$
17: **end if**;
18: **end if**;
19: **end if**;
20: **end for**;
21: **end for**;
22: end Preprocessamento.

4. Resultados Computacionais

O algoritmo proposto foi implementado utilizando a linguagem de programação C++ e executado em um PC Intel Core 2 Quad 2.50 GHz com 3.0 GB de memória RAM e sistema operacional Linux Ubuntu 8.04 (*kernel* 2.6.24-19). O procedimento foi testado em instâncias do 1-PDTSP apresentadas em Hernández-Pérez et al. (2008). Estas instâncias foram geradas aleatoriamente, onde $n - 1$ clientes são localizados dentro da área $[-500, 500] \times [-500, 500]$, cada uma possuindo uma demanda por entrega ou coleta cujo valor pertence ao intervalo $[-10, 10]$, e o depósito localizado em $(0, 0)$.

O número de iterações ($MaxIter$) e o número de pertubações ($MaxIterILS$) foram respectivamente 25 e 200. Estes valores foram calibrados empiricamente por meio de testes preliminares. Para cada instância foram realizadas 10 execuções do algoritmo GILS-RVND.

Nas tabelas apresentadas a seguir, **Problema** indica o nome da instância, ***n*** o número de clientes, ***opt*** a solução ótima, **#exec** indica o número de execuções em que o ótimo foi encontrado, **Best Sol.** a melhor solução encontrada pelo respectivo trabalho, **Avg. Sol.** a média das soluções obtidas, **Avg.Time** a média dos tempos, em segundos, das 10 execuções e **Avg. Time to Target** o tempo médio para alcançar ou melhorar a melhor solução conhecida.

As Tabelas 1 e 2 mostram, respectivamente, os resultados computacionais das instâncias menores $n \leq 60$ e maiores $100 \leq n \leq 500$. Vale ressaltar que as soluções ótimas das instâncias menores foram determinadas por Hernández-Pérez e Salazar-González (2004a), enquanto para as instâncias maiores somente os melhores resultados baseados em heurísticas encontrados na literatura foram considerados.

Tabela 1. Resultados Computacionais para as instâncias menores

Problema	n	*opt*	Hernández-Pérez et al. (2008) Best Sol.	Avg. Sol.	Avg. Time*	Zhao et al. (2009) #exec	Best Sol.	Avg. Sol.	GILS-RVND #exec	Best Sol.	Avg. Sol.	Avg. Time	Avg. Time to Target
n20q10A	20	4963	**4963**	4963,0	0,07	10	**4963**	4963,0	10	**4963**	4963,0	0,76	0,05
n20q10B	20	4976	**4976**	4976,0	0,06	10	**4976**	4976,0	10	**4976**	4976,0	0,84	0,05
n20q10C	20	6333	**6333**	6333,0	0,10	10	**6333**	6333,0	6	**6333**	6372,6	0,26	0,17
n20q10D	20	6280	**6280**	6280,0	0,07	10	**6280**	6280,0	10	**6280**	6280,0	0,51	0,02
n20q10E	20	6415	**6415**	6415,0	0,07	10	**6415**	6415,0	10	**6415**	6415,0	0,85	0,05
n20q10F	20	4805	**4805**	4805,0	0,08	10	**4805**	4805,0	10	**4805**	4805,0	0,61	0,02
n20q10G	20	5119	**5119**	5119,0	0,04	10	**5119**	5119,0	10	**5119**	5119,0	0,90	0,05
n20q10H	20	5594	**5594**	5594,0	0,06	10	**5594**	5594,0	10	**5594**	5594,0	0,91	0,04
n20q10I	20	5130	**5130**	5130,0	0,10	10	**5130**	5130,0	10	**5130**	5130,0	0,17	0,03
n20q10J	20	4410	**4410**	4410,0	0,08	10	**4410**	4410,0	10	**4410**	4410,0	0,49	0,02
n30q10A	30	6403	**6403**	6406,8	0,43	10	**6403**	6403,0	10	**6403**	6403,0	2,88	0,24
n30q10B	30	6603	**6603**	6603,0	0,24	10	**6603**	6603,0	10	**6603**	6603,0	4,02	0,11
n30q10C	30	6486	**6486**	6486,0	0,21	10	**6486**	6486,0	10	**6486**	6486,0	4,35	0,31
n30q10D	30	6652	**6652**	6655,1	0,40	10	**6652**	6652,0	10	**6652**	6652,0	3,03	0,09
n30q10E	30	6070	**6070**	6070,0	0,39	10	**6070**	6070,0	10	**6070**	6070,0	2,73	0,08
n30q10F	30	5737	**5737**	5737,0	0,37	10	**5737**	5737,0	10	**5737**	5737,0	3,04	0,12
n30q10G	30	9371	**9371**	9371,0	0,30	10	**9371**	9371,0	10	**9371**	9371,0	5,29	0,35
n30q10H	30	6431	**6431**	6431,2	0,33	10	**6431**	6431,0	10	**6431**	6431,0	1,88	0,15
n30q10I	30	5821	**5821**	5821,0	0,25	10	**5821**	5821,0	10	**5821**	5821,0	3,66	0,25
n30q10J	30	6187	**6187**	6187,4	0,38	10	**6187**	6187,0	10	**6187**	6187,0	3,76	0,19
n40q10A	40	7173	**7173**	7188,5	0,67	8	**7173**	7179,0	10	**7173**	7173,0	5,26	0,74
n40q10B	40	6557	**6557**	6568,5	0,91	5	**6557**	6564,5	5	**6557**	6564,5	7,49	5,55
n40q10C	40	7528	**7528**	7528,4	0,66	10	**7528**	7528,0	10	**7528**	7528,0	6,98	0,73
n40q10D	40	8059	**8059**	8135,6	1,00	8	**8059**	8075,4	10	**8059**	8059,0	8,76	1,92
n40q10E	40	6928	**6928**	6959,3	1,04	10	**6928**	6928,0	10	**6928**	6928,0	6,21	1,11
n40q10F	40	7506	**7506**	7590,5	0,83	10	**7506**	7506,0	7	**7506**	7512,6	7,23	5,83
n40q10G	40	7624	**7624**	7682,8	0,75	10	**7624**	7624,0	10	**7624**	7624,0	7,18	1,03
n40q10H	40	6791	**6791**	6795,7	0,83	10	**6791**	6791,0	10	**6791**	6791,0	7,68	0,75
n40q10I	40	7215	**7215**	7219,0	0,76	8	**7215**	7215,2	10	**7215**	7215,0	6,32	1,67
n40q10J	40	6512	**6512**	6513,3	0,53	10	**6512**	6512,0	10	**6512**	6512,0	4,97	0,54
n50q10A	50	6987	**6987**	6996,7	0,96	10	**6987**	6987,0	10	**6987**	6987,0	10,59	1,98
n50q10B	50	9488	**9488**	9512,6	1,73	8	**9488**	9501,8	10	**9488**	9488,0	11,81	1,08
n50q10C	50	9110	**9110**	9133,7	1,76	1	**9110**	9119,5	10	**9110**	9110,0	15,06	3,33
n50q10D	50	10260	**10260**	10464,3	1,82	2	**10260**	10354,8	7	**10260**	10262,7	16,44	16,67
n50q10E	50	9492	**9492**	9625,1	1,85	7	**9492**	9574,5	10	**9492**	9492,0	14,19	1,25
n50q10F	50	8684	**8684**	8773,2	1,72	8	**8684**	8692,5	10	**8684**	8684,0	14,59	1,50
n50q10G	50	7126	**7126**	7217,4	1,34	9	**7126**	7133,5	9	**7126**	7128,7	12,46	6,67
n50q10H	50	8885	8895	9006,5	1,46	1	**8885**	8956,9	7	**8885**	8885,9	13,85	14,48
n50q10I	50	8329	**8329**	8412,5	0,89	7	**8329**	8357,5	6	**8329**	8341,6	14,32	5,56
n50q10J	50	8456	**8456**	8666,1	1,61	1	**8456**	8475,8	9	**8456**	8459,3	11,78	2,77
n60q10A	60	8602	**8602**	8726,6	2,37	5	**8602**	8634,8	10	**8602**	8602,0	18,86	8,49
n60q10B	60	8514	**8514**	8683,2	2,38	10	**8514**	8514,0	10	**8514**	8514,0	19,92	2,01
n60q10C	60	9453	**9453**	9565,6	2,70	3	**9453**	9485,5	6	**9453**	9457,1	22,97	42,73
n60q10D	60	11059	11061	11320,6	2,62	1	**11059**	11140,2	-	11061	11084,9	27,32	-
n60q10E	60	9487	9572	9724,8	2,56	1	**9487**	9592,1	10	**9487**	9487,0	18,75	7,44
n60q10F	60	9063	**9063**	9437,2	2,36	1	**9063**	9192,2	2	**9063**	9098,1	22,32	75,97
n60q10G	60	8912	8967	9107,9	2,49	1	**8912**	8996,0	9	**8912**	8913,3	18,17	5,14
n60q10H	60	8424	**8424**	8467,3	2,19	3	**8424**	8472,3	10	**8424**	8424,0	17,20	7,59
n60q10I	60	9394	**9394**	9529,6	1,99	1	**9394**	9505,8	7	**9394**	9405,4	24,34	35,68
n60q10J	60	8750	**8750**	8956,5	2,29	1	**8750**	8803,3	7	**8750**	8761,3	19,12	10,09

*Intel Core 2 (2.4 GHz) PC

Tabela 2. Resultados Computacionais para as instâncias maiores

Problema	n	Hernández-Pérez et al. (2008)			Zhao et al. (2009)		GILS-RVND			
		Best Sol.	Avg. Sol.	Avg. Time*	Best Sol.	Avg. Sol.	Best Sol	Avg. Sol	Avg. Time	Avg. Time to Target
n100q10A	100	11874	12087,6	8,48	11828	11922,6	**11692**	11745,3	49,18	10,99
n100q10B	100	13172	13582,6	10,23	13114	13301,6	**12938**	13013,5	58,71	28,25
n100q10C	100	14063	14421,3	10,27	13977	14095,2	**13893**	13929,4	57,35	14,92
n100q10D	100	14490	14787,5	8,95	14253	14406,4	**14245**	14276,0	54,21	44,38
n100q10E	100	11546	12502,6	6,13	11411	11436,4	**11408**	11429,4	49,42	38,36
n100q10F	100	11734	12010,7	7,67	11644	11699,0	**11609**	11669,2	48,18	45,37
n100q10G	100	12049	12366,9	7,82	12038	12120,2	**11866**	11925,6	48,10	14,80
n100q10H	100	12892	13169,2	9,39	12818	12906,2	**12653**	12696,0	54,16	10,25
n100q10I	100	14048	14390,2	7,94	14032	14137,2	**13795**	13880,7	52,26	9,13
n100q10J	100	13430	13737,6	11,65	13297	13516,8	**13162**	13226,7	57,96	29,41
n200q10A	200	18013	18564,0	36,00	17686	17987,0	**17062**	17348,1	392,23	89,41
n200q10B	200	18154	18932,5	33,68	17798	18069,4	**17496**	17590,3	409,75	83,18
n200q10C	200	16969	17280,3	41,01	16466	16751,2	**16171**	16311,3	383,74	141,10
n200q10D	200	21565	22285,7	33,51	21306	21564,4	**20895**	21000,1	527,15	66,22
n200q10E	200	19913	20643,2	39,75	19299	19713,0	**18949**	19139,8	445,16	161,59
n200q10F	200	21949	22284,6	80,93	21910	22144,0	**21314**	21428,5	560,26	22,94
n200q10G	200	17956	18627,7	28,58	17712	17797,8	**17042**	17206,4	574,53	42,65
n200q10H	200	21463	22084,9	47,45	21276	21584,0	**20751**	20893,9	514,43	58,14
n200q10I	200	18606	19184,8	34,41	18380	18509,8	**17931**	18041,6	451,50	40,10
n200q10J	200	19273	19839,5	42,43	18970	19274,2	**18685**	18847,8	411,03	268,10
n300q10A	300	23244	24052,9	112,51	23242	23592,0	**22414**	22573,6	1566,46	69,79
n300q10B	300	23187	23845,6	109,55	22934	23028,6	**22249**	22380,6	1626,11	80,47
n300q10C	300	21800	22516,6	104,48	21922	22083,4	**21008**	21194,4	1582,83	104,95
n300q10D	300	25971	26462,1	162,95	25883	26289,8	**24820**	24986,2	1910,95	81,29
n300q10E	300	27420	27892,1	139,56	27367	27923,8	**26133**	26274,4	2077,98	91,80
n300q10F	300	24852	25278,2	153,93	24826	25055,4	**23764**	23924,8	1752,90	68,94
n300q10G	300	24308	24760,5	151,22	23868	24300,6	**23000**	23362,9	1841,27	138,19
n300q10H	300	22684	23116,5	67,49	21625	21965,0	**21119**	21329,2	1847,72	543,66
n300q10I	300	24633	25492,6	76,72	24513	24959,2	**23504**	23726,5	1989,32	82,58
n300q10J	300	23086	23530,2	100,05	22810	23045,0	**21953**	22047,6	1592,19	88,07
n400q10A	400	31486	31912,0	282,00	31678	31964,4	**29846**	29997,4	5467,17	207,90
n400q10B	400	24883	25606,4	204,21	24262	24752,4	**23589**	23884,0	3895,67	494,97
n400q10C	400	28942	29463,2	246,29	28741	29287,4	**27462**	27635,5	5008,02	166,66
n400q10D	400	24597	25308,6	142,84	24508	24794,8	**23257**	23413,3	4050,31	160,42
n400q10E	400	25548	26120,0	219,87	25071	25473,0	**24209**	24368,1	4287,08	164,43
n400q10F	400	27169	27755,1	273,01	26681	27362,8	**25982**	26084,9	4561,11	257,62
n400q10G	400	24626	25088,4	181,55	23891	24290,4	**23252**	23384,5	4201,82	207,50
n400q10H	400	26030	26468,8	220,74	25348	25811,4	**24458**	24603,6	4344,88	209,15
n400q10I	400	28992	29596,6	202,43	28714	29261,6	**27619**	27735,6	4950,41	200,49
n400q10J	400	26204	26916,2	231,03	26010	26489,4	**24631**	24953,0	4330,67	165,39
n500q10A	500	28742	29323,6	400,63	28857	29258,8	**27132**	27347,5	10124,90	306,90
n500q10B	500	27335	27711,1	332,67	26648	27454,8	**25510**	25702,6	10290,43	329,59
n500q10C	500	31108	31692,7	440,35	30701	31426,8	**29289**	29474,2	11874,76	399,91
n500q10D	500	30794	31428,4	426,51	30994	31442,2	**29168**	29313,0	11147,78	357,82
n500q10E	500	30674	31371,7	398,15	30905	31154,6	**29033**	29198,2	11291,73	323,01
n500q10F	500	28957	29812,3	263,14	28882	29241,0	**27244**	27648,3	11130,60	303,10
n500q10G	500	27198	27958,2	306,38	27107	27473,0	**25480**	25782,0	10233,98	346,93
n500q10H	500	36857	37361,1	600,00	37626	38142,4	**34847**	34985,8	15394,98	382,22
n500q10I	500	31045	31536,0	316,74	30796	31044,6	**29273**	29469,3	12638,36	313,47
n500q10J	500	31412	31877,9	425,56	31255	32310,0	**29481**	29597,8	12208,72	349,07

*Intel Core 2 (2.4 GHz) PC

A Tabela 1 ilustra os resultados obtidos pelo GILS-RVND nas instâncias menores. Uma comparação é feita com os algoritmos propostos por Zhao et al. (2009) e Hernández-Pérez e Salazar-González (2008). Os resultados mostram que o algoritmo proposto foi capaz de alcançar todas as soluções ótimas com exceção de uma instância. Verifica-se, também, que o tempo médio para encontrar as soluções ótimas foi consideravelmente menor que o tempo total da execução do algoritmo. Ao comparar os resultados obtidos pelo GILS-RVND e os demais algoritmos da literatura, observa-se que seu desempenho foi bastante satisfatório, sobretudo na qualidade das soluções médias, bem como a frequência de vezes em que a solução ótima foi alcançada.

A Tabela 2 apresenta os resultados obtidos nas instâncias maiores. Pode-se observar que o algoritmo proposto melhora todas as soluções conhecidas na literatura. O *gap* médio entre as melhores soluções obtidas pelo GILS-RVND e as melhores soluções conhecidas foi de -3,22%. Já o *gap* médio entre as soluções médias (**Avg. Sol.**) e as melhores soluções conhecidas foi de -2,55%. Com relação aos tempos médios para alcançar/melhorar a solução alvo, nota-se que estes foram substancialmente inferiores ao tempo médio de execução completa do algoritmo.

Como Zhao et al. (2009) não reportaram a média dos tempos de execução de cada instância em particular, apresentou-se na Tabela 2 apenas o tempo médio reportado por Hernández-Pérez e Salazar-González (2008). Por outro lado Zhao et al. (2009) reportaram os tempos médios para cada grupo de instância, para 100, 200, 300, 400 e 500 clientes. A Tabela 3 apresenta, então, a média dos tempos obtidos pelos algoritmos nas instâncias de diferentes dimensões. Uma comparação direta, em termos de tempo computacional entre os algoritmos, não foi realizada pois estes foram executados em máquinas com configurações distintas.

Tabela 3. Média dos tempos de execução para instâncias maiores

Grupo de Instâncias	Hernández-Pérez* et al. (2008)	Zhao** et al. (2009)	GILS-RVND	
			Avg. Time	Avg. Time to Target
100	8,85	21,12	52,95	24,59
200	41,76	95,23	466,98	97,34
300	117,86	212,23	1778,77	134,97
400	220,4	358,22	4509,71	223,45
500	391,47	570,15	11633,62	341,20

*Intel Core 2 (2.4 GHz) PC
**Pentium M 1.6 GHz PC com 256 MB de memória RAM

5. Considerações Finais

O presente trabalho tratou do *One-Commodity Pickup and Delivery TSP* (1-PDTSP). Um algoritmo heurístico híbrido foi desenvolvido baseado nas metaheurísticas *Greedy Randomized Adaptative Search Procedure* (GRASP), *Iterated Local Search* (ILS) e *Variable Neighborhood Descent* com ordem aleatória de vizinhanças (RVND).

O algoritmo proposto foi testado em 50 instâncias contendo de 20 a 60 clientes, e 50 instâncias com 100 a 500 clientes. Com exceção de apenas uma instância, GILS-RVND alcançou todas as soluções ótimas das instâncias menores. Quanto as instâncias maiores, o algoritmo proposto foi capaz de superar todas soluções conhecidas.

Como trabalho futuro, pretende-se desenvolver novas estratégias de redução do espaço de busca de maneira a tornar o algoritmo mais eficiente em termos de tempo de execução, mas sem comprometer a qualidade das soluções geradas.

Referências

Hernández-Pérez, H. e Salazar González, J. J. (2004a). A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145, 126-139.

Hernández-Pérez, H. e Salazar González, J. J. (2004b). Heuristics for the one commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38, 245-255.

Hernández-Pérez, H. e Salazar González, J. J. (2007). The one-commodity pickup and delivery traveling salesman problem: Inequalities and algorithms. *Networks*, 50, 258-272.

Hernández-Pérez, H., Rodríguez-Martín, I. e Salazar González, J. J. (2008). A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36, 1639-1645.

Martin, O., Otto, S. W. e Felten, E. W. (1991). Large-step Markov chains for the traveling salesman problem. *Complex Systems*, 5, 299-326.

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

Lourenço, H. R., Martin, O. C. e Stutzle, T. (2002). Iterated Local Search, In Glover F. e Kochenberger G., editores, *Handbook of Metaheuristics*, 321-353.

Resende, M. G. C. e Ribeiro, C. C. (2003). Greedy randomized adaptive search procedures. In Glover F. e Kochenberger G., editores, *Handbook of Metaheuristics*, 219-249.

Zhao, F., Li, S., Sun, J. e Mei, D. (2009). Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Industrial Engineering*, 56, 1642-1648