X Congresso Brasileiro de Inteligência Computacional (CBIC'2011), 8 a 11 de Novembro de 2011, Fortaleza, Ceará


# UM ALGORITMO ILS APLICADO AO PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA E ENTREGA

**Gustavo Silva Semaan, Anand Subramanian, Luiz Satoru Ochi**
Instituto de Computação da Universidade Federal Fluminense (IC–UFF)
{gsemaan, anand, satoru}@ic.uff.br

**José André de Moura Brito**
Escola Nacional de Ciências Estatísticas (ENCE–IBGE)
jose.m.brito@ibge.gov.br

**Resumo –** O Problema do Caixeiro Viajante com Coleta e Entrega (PCVCE) é uma variante do conhecido Problema do Caixeiro Viajante onde cada cliente possui uma demanda por coleta ou entrega. Este trabalho apresenta um algoritmo heurístico que combina os conceitos da metaheurística Busca Local Iterada (*Iterated Local Search - ILS)* e do método de Descida em Vizinhança Variável (*Variable Neighborhood Descent - VND)* para resolver o PCVCE. A abordagem desenvolvida foi testada em 35 instâncias da literatura, variando de 11 a 71 clientes. Os resultados obtidos foram bem competitivos, onde as melhores soluções conhecidas foram igualadas ou melhoradas.

**Palavras-chave –** Busca Local Iterada, Descida em Vizinhança Variável, Problema do Caixeiro Viajante com Coleta e Entrega, Metaheurísticas, Inteligência Computacional.

**Abstract –** The Traveling Salesman Problem with Pickup and Delivery (TSPPD) is a variant of the well-known Traveling Salesman Problem in which each customer has a demand for pickup or delivery. This work presents a heuristic algorithm that combines the concepts of the Iterated Local Search (ILS) metaheuristic and the Variable Neighborhood Descent (VND) method to solve the TSPPD. The developed approach was tested in 35 instances from the literature, varying from 11 to 71 customers. The results obtained were quite competitive, where the best known solutions were either equaled or improved.

**Keywords –** Iterated Local Search, Variable Neighborhood Descent, Traveling Salesman Problem with Pickup and Delivery , Metaheuristics, Computational Intelligence.

## 1. INTRODUÇÃO

As áreas de Inteligencia Computacional e Otimização Combinatória têm recebido contribuições relevantes tanto no aspecto técnico como no algorítmico entre as quais estão as denominadas Metaheurísticas ou Heurísticas Inteligentes. De forma resumida, tratam-se de heurísticas genéricas com ferramentas que permitem que algoritmos heurísticos associados sejam capazes de escapar de ótimos locais, que podem estar distantes de uma solução ótima global ( [1], [2]).

Algumas das metaheurísticas amplamente utilizada na literatura são apresentadas em [1], tais como: *Iterated Local Search* (ILS), *Genetic Algorithms*, *Memetic Algorithms*, *Simulated Annealing*, *Tabu Search* e *Greedy Randomized Adaptive Search Procedures - GRASP*.

O Problema do Caixeiro Viajante com Coleta e Entrega PCVCE é uma variante do clássico Problema do Caixeiro Viajante (PCV) na qual cada cliente possui uma demanda associada, que pode ser exclusivamente, de coleta ou entrega. Tal problema pode ser definido em um grafo que possui vértices de coleta ou de entrega, que possuem relacionamento um para um. Esse problema consiste em determinar uma rota de custo mínimo em que cada vértice de coleta é visitado antes de seu vértice correspondente, de entrega.

Inicialmente, para a resolução desse problema, define-se um grafo que possui dois conjuntos de vértices associados aos clientes de coleta e de entrega, respectivamente. Acrescenta-se, ainda, que nesse grafo entre cada dois vértices $v_i$ e $v_j$, $i < j$, existe uma aresta $e(i, j) \in E$ associada a um custo não-negativo $c_{ij}$. O valor $c_{ij}$ trata-se do arredondamento da distância euclidiana entre os clientes $i$ e $j$.

Seja $G = (V, E)$ um grafo em que $V$ é o conjunto de vértices e $E$ o conjunto de arestas. O conjunto $V$ é constituído pelos vértices de coleta ($P = \{1, .., n\}$, de entrega ($D = \{n + 1, .., 2n\}$) e o depósito. O vértice de entrega correspondente ao vértice de coleta $i \in P$ é o $i + n \in D$. Desta forma, uma solução é uma sequência ordenada formada por $V = P \cup D \cup \{0, 2n + 1\}$, em que os clientes 0 e $2n + 1$ correspondem a saída e chegada ao depósito, respectivamente.

Considerando essas informações, a resolução do problema corresponde a determinar uma rota de custo mínimo na qual cada vértice de coleta é visitado antes de seu vértice correspondente, de entrega. Mais especificamente, esse problema consiste em encontrar um ciclo hamiltoniano em $G$ que deve iniciar no vértice 0 e terminar em $(2n + 1)$, passando por vérties de coleta $i$ antes de seus correspondentes de entrega $i + n$. O objetivo, então, é minimizar a soma dos custos $c_{ij}$ das arestas que pertencem ao ciclo hamiltoniano em $G$.

Segundo [3], o PCVCE possui muitas aplicações reais, como em serviços de correio ( [4]) e transporte de passageiros ( [5]). Trata-se também da versão que considera um veículo (*single-vehicle*) do problema de roteamento de veículos com coleta e entrega denominado *multi-vehicle one-to-one pickup and delivery problem* (VRPPD) sobre o qual existe uma literatura rica [5]. Esse problema é considerado um caso especial da precedência restrita do *PCV* em que cada vértice pode ter vários antecessores e que, neste caso, cada vértice de coleta deve preceder seu correspondente de entrega [6]. Já no *PCV com Backhauls*, todos os vértices de coleta devem ser visitados antes de qualquer um dos vértices de entrega [7], enquanto no problema *Dial-a-Ride* considera-se a capacidade do veículo para o transporte de passageiros.

O problema abordado pertence a classe de problemas *NP-difícil*, uma vez que qualquer instância *PCV* pode ser modelado em uma instância PCVCE utilizando uma transformação em tempo polinomial [8]. Além disso, [3] afirma que trata-se de um problema muito difícil do ponto de vista empírico, em que o tamanho maior exemplo resolvido de forma até o momento da publicação de seu trabalho era de apenas 31 clientes ($n = 15$).

A primeira formulação linear inteira para o problema foi apresentado por [9]. Outros trabalhos propuseram algoritmos exatos e, inclusive, de algumas de suas variantes [10], [11] e [12].

Em [3] e em [12] foram apresentados estudos sobre a estrutura poliédrica do PCVCE. Porém, nesses trabalhos, foram aplicadas as desigualdades encontradas em um algoritmo de *branch-and-cut*. Heuristicas para este problema também são relatadas na literatura, como em [8], [13] e [14].

Esse trabalho esté dividido em cinco seções, incluindo esta introdução. A Seção 2 apresenta o algoritmo proposto bem como as heurísticas utilizadas. Já os experimentos, resultados e análises são apresentados na Seção 3. As Seções 4 e 5 apresentam, respectivamente as conclusões desse trabalho e sugestões e idéias para pesquisas e trabalhos futuros.

## 2. ALGORITMO PROPOSTO

O algoritmo proposto foi baseado na metaheurística *Iterated Local Search* (ILS) [15] e um procedimento baseado no método *Variable Neighborhood Descent* (VND) [16], [1]. Uma vez que a as buscas locais utilizadas no *VND* são selecionadas aleatoriamente, o algoritmo foi denominado *Random VND* ou simplesmente *RVND*. Assim, considerando a utilização dessas duas metaheurísticas, o algoritmo heurístico proposto nesse trabalho denominou-se *ILS-RVND*.

A metaheurística ILS procura focar a busca em um pequeno subespaço, com a idéia de que um procedimento de busca local pode ser melhorado com a geração de novas soluções de partida, obtidas por meio de perturbações na solução ótima local [15].

### 2.1 Representação da Solução

A estrutura de dados utilizada para a representação de soluções foi um vetor de inteiros, em que o identificador dos clientes seguem a ordem das visitas. Uma vez que nesse problema o início e o fim da solução é o depósito, ele não foi representado na estrutura de dados utilizada.

Seguindo os trabalhos da literatura, o presente algoritmo considerou como soluções válidas aquelas que têm as seguintes características:

- Cada cliente deve ser visitado exatamente uma vez, com exceção do depósito, onde a rota é iniciada e finalizada.
- O cliente de coleta $i$ deve ser visitado antes do cliente de entrega $i'$ associado.
- Deve existir apenas uma rota.
- Não existe restrição de capacidade.

É importante ressaltar que todos os procedimentos desse trabalho consideram apenas a formação e manipulação de soluções válidas.

### 2.2 Procedimentos Construtivos

Embora qualquer heurística de construção de soluções possa ser utilizada, é interessante a formação de soluções diversificadas. Dessa forma, foram implementados e incorporados ao ILS-RVND cinco procedimentos para a construção de soluções iniciais, em que três baseam-se na formação de soluções aleatórias válidas e dois em algoritmos da literatura [17] que buscam a formação de soluções de melhor qualidade, em que a inserção de menor custo é realizada. A solução incial é construída quando todos os clientes foram adicionados à rota.

Procedimentos para criação de soluções aleatórias válidas:

- **Entrega Gulosa**: Inicialmente os clientes de coleta são adicionados a uma LRC (Lista Restrita de Candidatos). A cada cliente de coleta selecionado aleatoriamente na LRC seu correspondente, cliente de entrega, é visitado imediatamente. Esse algoritmo baseia-se na construção de soluções iniciais do problema de coleta e entrega *Dial-a-Ride*, problema este em que a capacidade do veículo é considerada e ocorre o transporte de pessoas.
- **Coleta e Entrega Aleatória**: Inicialmente, os clientes de coleta são adicionados a uma LRC. A cada cliente de coleta selecionado aleatoriamente na LRC seu correspondente, cliente de entrega, é adicionado à LRC. Dessa forma a visita a um cliente de entrega ocorre apenas quando o cliente de coleta associado já foi visitado.

- ***Backhauls***: Nesse procedimento os clientes de coleta são adicionados a uma LRC (Lista Restrita de Candidatos). Após todos os clientes da LRC terem sido adicionados na solução, ou seja, $LRC = \{\}$, a lista é preenchida com os clientes de entrega. Assim, todos os clientes de coleta serão visitados antes de qualquer cliente de entrega, caracterizando uma solução típica do problema com *backhauls*.

Procedimentos para criação de soluções que realizam a inserção minimizando o custo mas considerando também apenas a formação de soluções válidas:

- **Criterio de inserção viável mais próxima**: O custo de inserção de um cliente $k$ é calculado pela equação (1), considerando a inserção do cliente após cada cliente $i$ da solução em construção.

$$g(k) = c_{ik} \tag{1}$$

- **Criterio de inserção viável mais barata**: Inserção de um novo cliente entre cada par adjacente da solução. O custo de inserção de um cliente $k$ é calculado pela equação (2) em que a primeira parcela representa o custo de inserção desse cliente entre todos os pares de clientes adjacentes $i$ e $j$. Já a segunda parcela corresponde a uma bonificação que objetiva evitar que clientes distantes do depósito sejam incluidos no final da solução. O custo de ida e volta do depósito é ponderado por um fator $\gamma$, critério guloso inspirado em [18]. Esse fator corresponde a um valor obtido de forma aleatória dentro do intervalo discreto $\{0.00, 0.05, 0.10, .., 1.65, 1.70\}$, obtido por [17] em experimentos preliminares.

$$g(k) = (c_{ik} + c_{kj} - c_{ij}) - \gamma(c_{0k} + c_{k0}) \tag{2}$$

A construção das soluções iniciais nesse trabalho considera os cinco procedimentos descritos, que são executados aleatoriamente. Porém, com base em experimentos preliminares, foram priorizados os procedimentos *Criterio de inserção viável mais próxima* e *Criterio de inserção viável mais barata*. Em particular, esses dois procedimentos possuem probabilidade de execução de aproximadamente 33%, enquanto os demais possuem 11% de probabilidade de execução.

### 2.3 Procedimentos de Busca Local

Uma vez que uma solução para o problema abordado nesse trabalho consiste na formação de uma rota de custo mínimo, utilizou-se apenas estruturas de vizinhança *intra-rotas*. Com o objetivo de explorar de forma mais eficiente o espaço de soluções foram utilizadas 3 estruturas de vizinhanças (buscas locais) do problema de Roteamento de Veículos clássico. Essas estruturas foram adaptadas de forma a admitir somente movimentos válidos, que não tornam as soluções inválidas. São elas:

- *Or-opt* $w$: $w$ clientes adjacentes são removidos e inseridos em outra posição da rota. Nos experimentos foi utilizado $w \in \{1, 2, 3\}$ [17].
- *2-opt*: duas arestas não adjacentes são substituídas por duas novas arestas formando uma nova rota [17].
- *Permutação*: cada par de clientes associados, correspondentes de coleta e entrega, são alocados nas extremidades da solução: cliente de coleta no início e de entrega no final. Em seguida ocorre a permutação entre esses clientes.

### 2.4 Perturbações

Foram desenvolvidos dois procedimentos de perturbação e a intensidade da perturbação (percentual de clientes selecionados), relacionada ao tamanho da entrada, variou aleatoriamente entre 1% e 5% . O procedimento de perturbação utilizado a cada iteração também é selecionado de forma aleatória.

1. Seleciona um cliente qualquer na solução e verifica-se seu tipo. Caso esse cliente seja do tipo coleta, ele é reinserido no inicio da solução. Caso contrário é inserido no final da solução.
2. Realiza trocas aleatórias entre dois pares de clientes.

### 2.5 ILS-RVND

A Figura 1 apresenta os pseudocódigos do ILS-RVND e do RVND. No ILS-RVND (linhas 3-19), e executada $iterMax$ vezes, caracterizando uma metaheurísica multi-start. A linha 4 corresponde a chamada dos procedimentos para a construção de soluções iniciais, formando soluções diversificadas. Entre as linhas 7 e 15 o algoritmo trabalha no refinamento das soluções, em que o algoritmo RVND é utilizado na linha 8.

Para perturbar as soluções obtidas foi utilizado o procedimento $Perturbacao$, apresentado na linha 8. Além disso, é importante ressaltar que a cada melhoria na qualidade da solução o algoritmo é reiniciado com a atribuição $iterILS = 0$, conforme é apresentado na linha 11. Após a fase de refinamento da solução, a melhor solução da iteração $i$ ($s'$) é comparada com a melhor solução obtida nas iterações anteriores ($s^*$). Caso $s'$ seja melhor, a solução $s^*$ é atualizada.

Já no RVND foi utilizado para o refinamento das soluções. O primeiro passo do algoritmo é inicializar a estrutura de vizinhanças $listaEstrtVizinhanca$. Em seguida, a cada iteração, uma estrutura de vizinhança ($N^{(\alpha)} \in listaEstrtVizinhanca$) é selecionada de maneira aleatória, conforme o código da linha 4. Então, com base na solução $s'$ inicia-se uma busca local utilizando $N^{(\alpha)}$. A estrutura de vizinhança selecionada só é removida da lista caso não tenha melhorado a solução submetida. Dessa forma, o RVND é executado até que $listaEstrtVizinhanca = \{\}$.

```
ILS-RVND(iterMax, iterMaxIls)
1  início
2      s* = ∞;
3      para (i = 1, 2, ..., iterMax) faça
4          s ← ConstroiSI();
5          s' ← s;
6          iterIls ← 0;
7          enquanto (iterIls ≤ iterMaxIls) faça
8              s ← RVND(N(.), f(.));
9              se (f(s) < f(s')) então
10                 s' ← s;
11                 iterIls ← 0;
12             fim
13             s ← Perturbacao(s');
14             iterIls ← iterIls + 1;
15         fim
16         se (f(s') < f(s*)) então
17             s* ← s';
18         fim
19     fim
20     retorna s*;
21 fim
```

```
RVND(N(.), f(.)), s
1  início
2      Inicializa listaEstrtVizinhanca;
3      enquanto (listaEstrtVizinhanca ≠ ∅) faça
4          N(α) ← Selecione um elemento de listaEstrtVizinhanca;
5          s' ← MelhorVizinho(s, N(α));
6          se (f(s') < f(s)) então
7              s ← s';
8              s ← IntraRota(s);
9          fim
10         senão
11             Remove N(α) de listaEstriVizinhanca;
12         fim
13     fim
14     retorna s
15 fim
```

Figura 1: *Pseudocódigos ILS-RVND e RVND*

## 3. RESULTADOS COMPUTACIONAIS

A presente Seção traz um conjunto de resultados computacionais obtidos a partir do algoritmo ILS-RVND.

Os experimentos foram realizados em um computador dotado de um processador i7 3.0 GHz, com 8GB de memória e sistema operacional linux Kubuntu 10.09, kernel 2.6.31-22-generic. Todos os procedimentos que compõem o algoritmo foram desenvolvidos em linguagem C++. Nesse trabalho foram realizados três experimentos utilizando um conjunto com 35 instâncias da literatura, 50 execuções da heurística em cada instância e critério de parada 3600 segundos. Embora e equipamento possibilite, nenhum recurso ou técnica para processamento paralelo foi utilizada.

O primeiro experimento teve por objetivo avaliar os valores das melhores soluções e os tempos de execução (menores, médios e maiores) foram obtidos. Já o segundo experimento consistiu em analisar de forma mais detalhada a robustez dos algoritmos propostos, utilizando gráficos *TTTPlot* (*Time-to-Target Plots*, [19]) das duas instâncias em que o algoritmo demandou mais tempo de execução para alcançar o *gap* 0%. O terceiro experimento também realizou análise de probabilidade empírica mas, considerando a execução do algoritmo proposto nas 15 instâncias que possuem acima de 50 clientes e utilizando como alvo *gap* = 0%.

### 3.1 Instâncias Utilizadas

O conjunto de instâncias utilizadas nos experimentos foi gerado por [12] através de pontos aleatórios em um quadrado $[0, 1000] \times [0, 1000]$. Foi considerado o conceito de demanda na nomenclatura das instâncias, em que uma demanda corresponde a um par de clientes associados, sendo um de coleta e um de entrega. Cada instância, então, possui um ponto que representa o depósito e *2n* pontos que representam os clientes. Foram geradas 5 instâncias para cada demanda *n* em $\{5, 10, .., 35\}$, diferenciadas pelas letras *m* em $\{a, b, c, d, e\}$ e o custo de viagem ($c_{ij}$ ) deve ser obtido pelo arredondamento das distâncias euclidianas.

A nomenclatura geral é a prob***nm***, ou seja, a instância *prob35c* possui 71 clientes. As instâncias utilizadas nesse trabalho estão disponíveis no endereço *http://www.diku.dk/ sropke/*.

### 3.2 Experimentos

A Figura 2 apresenta os resutados computacionais obtidos no primeiro experimento, com a execução da heurística ILS-RVND em cada instância e os resultados apresentados em [3]. Nessa figura, a terceira coluna (*opt*) informa se a melhor solução da literatura (*best*) é uma solução ótima(s) ou não. Já coluna 5, mostra os *gaps* do algoritmo proposto em relação ao *best* da literatura.

Com base nas médias dos *gaps* na Figura 2 pode-se observar que em todas as 50 execuções, para cada uma das 35 instâncias, o algoritmo encontrou a mesma solução disponibilizada na literatura para 34 instâncias. Além disso uma solução melhor do que a da literatura no caso particular da instância *prob35e* foi obtida. Em relação ao tempo, nas execuções para as instâncias até 41 clientes o maior valor foi de apenas 2 segundos e, na média, menos de 1 segundo.

Mesmo tendo como o critério de parada o tempo de execução estipulado, com base no menor tempo, observa-se que o algoritmo alcançou a melhor solução conhecida em menos de 4 segundos em pelo menos uma execução para todas as instâncias. Quando considerando o tempo médio, apenas nas instâncias *prob35b* e *prob35e* o valor foi superior a 100 segundos e além disso, a média dos tempos médios foi de 23,23 segundos. Já em relação ao maior tempo, embora em duas instâncias o valor seja superior a 2000 segundos, a média dos maiores tempos foi inferior a 200 segundos.

| | | | Solução | | Tempo (segundos) ILS-RVND ** | | |
|---|---|---|---|---|---|---|---|
| Instância | Clientes | Opt | Best | ILS-RVND | Menor | Média | Maior |
| prob5a | 11 | s | 3585 | 0,00 | 0 | 0 | 0 |
| prob5b | 11 | s | 2565 | 0,00 | 0 | 0 | 0 |
| prob5c | 11 | s | 3787 | 0,00 | 0 | 0 | 0 |
| prob5d | 11 | s | 3128 | 0,00 | 0 | 0 | 0 |
| prob5e | 11 | s | 3123 | 0,00 | 0 | 0 | 0 |
| prob10a | 21 | s | 4896 | 0,00 | 0 | 0 | 0 |
| prob10b | 21 | s | 4490 | 0,00 | 0 | 0 | 0 |
| prob10c | 21 | s | 4070 | 0,00 | 0 | 0 | 0 |
| prob10d | 21 | s | 4551 | 0,00 | 0 | 0 | 0 |
| prob10e | 21 | s | 4874 | 0,00 | 0 | 0 | 0 |
| prob15a | 31 | s | 5150 | 0,00 | 0 | 0 | 0 |
| prob15b | 31 | s | 5391 | 0,00 | 0 | 0 | 0 |
| prob15c | 31 | s | 5008 | 0,00 | 0 | 0 | 0 |
| prob15d | 31 | s | 5566 | 0,00 | 0 | 0 | 0 |
| prob15e | 31 | s | 5229 | 0,00 | 0 | 0 | 0 |
| prob20a | 41 | s | 5698 | 0,00 | 0 | 0 | 1 |
| prob20b | 41 | s | 6213 | 0,00 | 0 | 0 | 2 |
| prob20c | 41 | s | 6200 | 0,00 | 0 | 0 | 1 |
| prob20d | 41 | s | 6106 | 0,00 | 0 | 0 | 1 |
| prob20e | 41 | s | 6465 | 0,00 | 0 | 0 | 1 |
| prob25a | 51 | | 7332 | 0,00 | 0 | **99** | 390 |
| prob25b | 51 | s | 6665 | 0,00 | 0 | 21 | 106 |
| prob25c | 51 | s | 7095 | 0,00 | 0 | 2 | 12 |
| prob25d | 51 | s | 7069 | 0,00 | 1 | 2 | 12 |
| prob25e | 51 | s | 6754 | 0,00 | 0 | 2 | 17 |
| prob30a | 61 | | 7309 | 0,00 | 0 | 10 | 63 |
| prob30b | 61 | s | 6857 | 0,00 | 1 | 13 | 53 |
| prob30c | 61 | s | 7723 | 0,00 | 1 | 9 | 46 |
| prob30d | 61 | s | 7310 | 0,00 | 1 | 33 | 185 |
| prob30e | 61 | | 7213 | 0,00 | 1 | 64 | 467 |
| prob35a | 71 | s | 7746 | 0,00 | 2 | 74 | 399 |
| prob35b | 71 | | 7904 | 0,00 | 2 | **220** | 2425 |
| prob35c | 71 | | 7949 | 0,00 | 3 | 17 | 49 |
| prob35d | 71 | | 7905 | 0,00 | 3 | 68 | 361 |
| prob35e | 71 | | 8530 | **-0,07** | 3 | **179** | 2245 |
| | | | **Médias** | **0,00** | **0,51** | **23,23** | **195,31** |

Figura 2: Resultados Computacionais

É possível observar que o algoritmo alcançou a melhor solução conhecida para todas as instâncias. Além disso, melhorou o resultado *best* para a instância *prob35e* que ainda não possuí ótimo comprovado com $gap = -0,1\%$, tendo como média $gap = -0,07\%$ nas 50 execuções.

No segundo experimento as instâncias em que o algoritmo demandou os maiores tempos, *prob35b* e *prob35e*, foram foram submetidas a experimentos de análise de probabilidade empírica. Com base nos resultados obtidos, por meio de gráficos, é possível verificar a probabilidade que o algoritmo possui de alcançar os alvos (*gaps*) em relação do tempo de execução.

As Figuras 3 e 4, apresentam os gráficos TTTPlot com resultados do algoritmo na busca pelos *gaps* especificados.

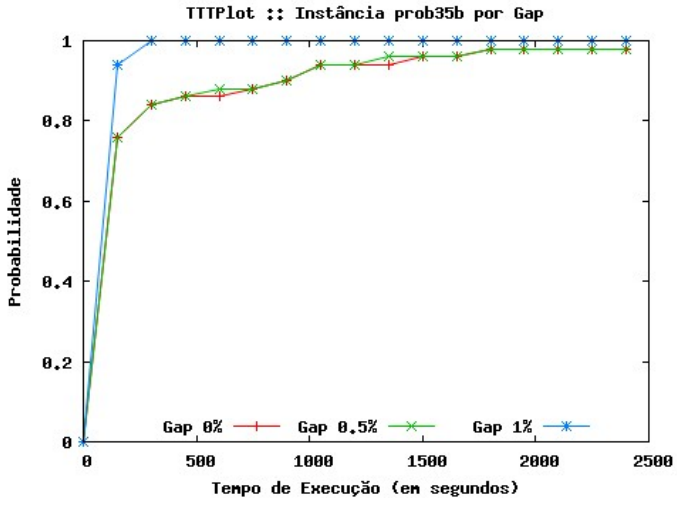


Figura 3: TTTPlot da instância prob35b e alvos Gap 0%, 0.5% e 1%.

Com base na Figura 3, o algoritmo apresentou um comportamento semelhante para alcançar os gaps 0% e 0.5%, em que apesar de algumas execuções alcançarem a melhor solução conhecida em cerca de 2400, em pouco mais de 200 segundos a probabilidade de alcançar a solução *best* é de 80%. Em todas as execuções o algoritmo alcançou soluções com gap $\leq 1\%$ em menos de 300 segundos .

Já na Figura 4 embora em algumas execuções o algoritmo demandou 2200 segundos para alcançar os alvos 0% e 0,5%, cerca de *90%* das execuções alcançaram a solução *best* em cerca de 500 segundos . Além disso, o algoritmo alcançou gaps menores

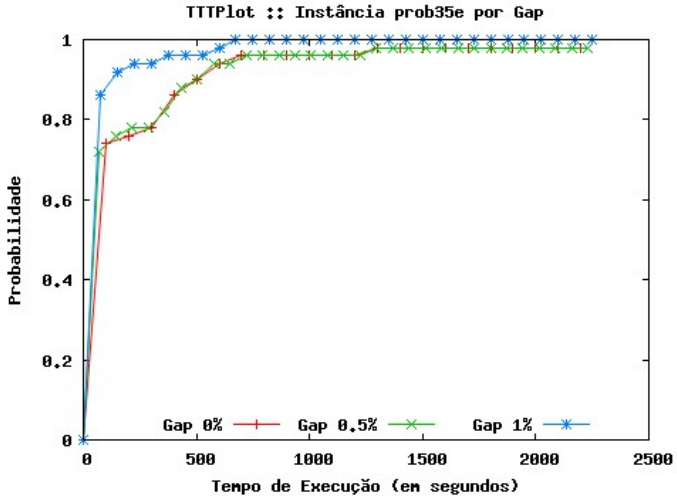


Figura 4: TTTPlot da instância prob35e e alvos Gap 0%, 0.5% e 1%.

ou iguais a 1% em todas as execuções em aproximadamente 700 segundos de execução.

O terceiro experimento, também utilizando análise de probabilidade empírica, executou o algoritmo proposto nas 15 instâncias que possuem acima de 50 clientes e o alvo foi a solução *best* (*gap* = 0%). As Figuras 5, 6 e 7 apresentam os gráficos TTTPlot desse experimento.

Com base na Figura 5, observa-se que o algoritmo teve mais dificuldade para alcançar o alvo nas instâncias *prob25a* e *prob25b*. Esse comportamento aconteceu também, em menor intensidade, para as instâncias com 61 clientes *prob30d* e *prob30e* (Figura 6), e para as instâncias com 71 clientes *prob35b* e *prob35e* (Figura 7).

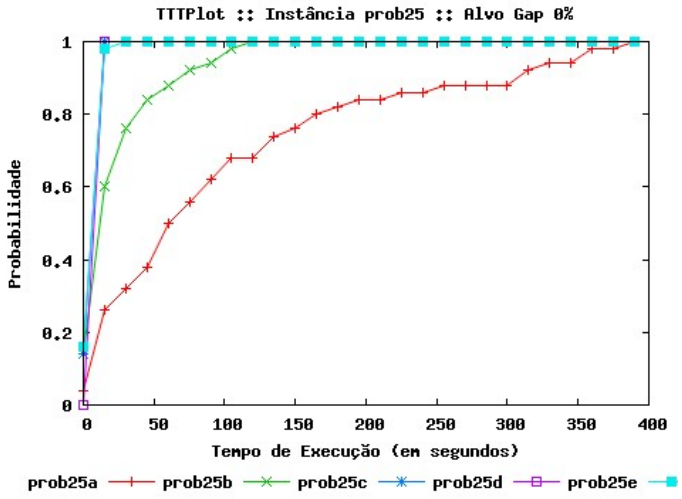


Figura 5: TTTPlot das instâncias com 51 clientes e alvo Gap 0%.

## 4 Conclusões

O presente trabalho apresentou um algoritmo baseado nas metaheurísticas ILS(*Iterated Local Search* e VND (*Variable Neighborhood Descent*).

Foram realizados três experimentos utilizando 35 instâncias disponíveis na literatura, que possuem entre 11 e 71 clientes. No primeiro experimento os valores das melhores soluções e os tempos de execução (menores, médios e maiores) foram obtidos e apresentados na Figura 2. Já o segundo experimento apresentou uma análise para avaliar de forma mais detalhada a robustez dos algoritmos propostos por meio de gráficos *TTTPlot* nas instâncias em que o algoritmo demandou mais tempo de execução para alcançar a solução *best*. No terceiro também foi apresentada uma análise de probabilidade empírica porém, considerando a execução do algoritmo proposto nas 15 instâncias que possuem acima de 50 clientes.

O algoritmo encontrou a melhor solução conhecida em todas as instâncias e melhorou em uma das instâncias, em que a média dos *gaps* para esta instância foi -0.07%.

Para todas as instâncias o algoritmo alcançou a melhor solução conhecida em menos de 4 segundos em pelo menos uma execução. Além disso a média dos maiores tempos foi inferior a 200 segundos e a média dos tempos médios foi de apenas 23

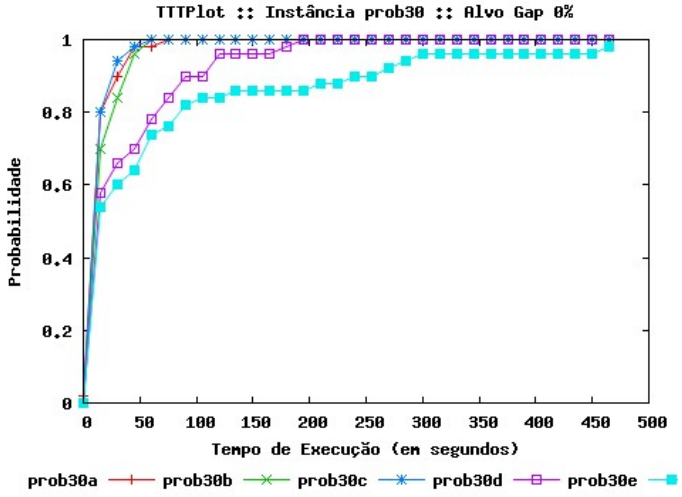


Figura 6: TTTPlot das instâncias com 61 clientes e alvo Gap 0%.

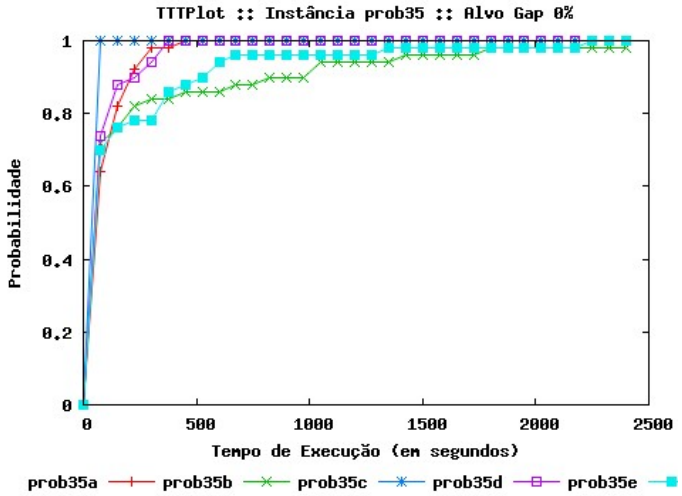


Figura 7: TTTPlot das instâncias com 71 clientes e alvo Gap 0%.

segundos.

Em relação a instância *prob35b*, em pouco mais de 200 segundos a probabilidade de alcançar a solução *best* é de 80% e de 100% de alcançar soluções com gap $\leq 1\%$ em menos de 300 segundos . Já a instância *prob35e*, em cerca de 500 segundos a probabilidade de alcançar a solução *best* é de 90% e de 100% de alcançar soluções com gap $\leq 1\%$ em menos de 700 segundos .

Tais resultados e observações indicam que o algoritmo ILS-RVND pode ser uma boa alternativa para resolver o PCVCE.

## 5 Trabalhos Futuros

Em [3] foram realizados experimentos com mais dois conjuntos de dados propostos por [8]. O primeiro possui instâncias entre 51 e 441 clientes, embora [3] tenha utilizado apenas instâncias com até 101 clientes, no total de 33 unidades. Já o segundo conjunto de dados foi baseadas em instâncias do caixeiro viajante tradicional resolvidos na otimalidade, modelados como instâncias do caixeiro viajante com coleta e entrega. Esse conjunto possui 20 instâncias em que a metade possui 101 clientes e a outra metade 201.

Entre as pesquisas e trabalhos futuros, destacam-se:

- A realização de experimentos com os conjuntos de dados propostos por [8], inclusive com instâncias maiores, na busca por melhoria em soluções com ótimo global ainda não comprovado.
- Pesquisar o desenvolvimento de uma proposta híbrida entre a solução de [3] e o algoritmo proposto nesse trabalho.
- Desenvolvimento de novas estruturas de vizinhanças e perturbações para o *ILS-RVND*.
- Verificar a possibilidade de utilização de mineração de dados para obter padrões em subrotas e apoiar o algoritmo proposto. Em [20] foram apresentados estudos em que a mineração de dados auxiliou algoritmos evolutivos na resolução de problemas de roteamento de veículos.

# REFERÊNCIAS

[1] F. Glover. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[2] Neves, T.A. ;Ochi, L.S. "GRASP com Memória Adaptativa Aplicado ao Problema de Roteamento e Scheduling de Sondas de Manutenção". In *ENIA 2007 - VI Encontro Nacional de Inteligência Artificial*, 2007.

[3] Irina Dumitrescu, I.;Ropke, S.; Cordeau, J.; Laporte, G. "The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm". In *Mathematical Programming*, pp. 269–305, 2008.

[4] Cordeau, J.F.; Laporte, G.; Ropke, S. "Recent models and algorithms for one-to-one pickup and delivery problems". In *The Vehicle Routing Problem, Latest Advances and Challenges*, edited by S. R. B.L. Golden and E. Wasil. Springer, 2007.

[5] J. Cordeau. "A branch-and-cut algorithm for the dial-a-ride problem". In *Operations Research*, 2006.

[6] Balas, E.; Fischetti, M.; Pulleyblank, W. R. "The precedence-constrained asymmetric traveling salesman polytope". In *Mathematical Programming*, 1995.

[7] Gendreau, M.; Hertz, A.; Laporte, G. "The traveling salesman problem with backhauls". In *Computers and Operations Research*, 1996.

[8] Renaud, J.; Boctor, F.F.; Laporte, G. "Pertubation heuristics for the pickup and delivery traveling salesman problem". In *Computers and Operations Research*, 2002.

[9] K. Ruland. "Polyhedral solution to the pickup and delivery problem". Washington University, 1995.

[10] I. X. A. Lima. "Algoritmos para problemas de roteamento de veÃÂculos com entrega e coleta". In *DissertaÃ§Ã£o de Mestrado*. Universidade Federal Fluminense, 2010.

[11] Hernandez-Perez, H.; Salazar-Gonzalez, J.J. "A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery". In *Discrete Applied Mathematics*, 2004.

[12] Ruland, K. S.; Rodin, E. Y. "The pickup and delivery problem: Faces and branch-and-cut algorithm". In *Computers and Mathematics with Applications*, pp. 1–13, 1997.

[13] Renaud, J.; Boctor, F.F.; Ouenniche, J. "A heuristic for the pickup and delivery traveling salesman problem". In *Computers and Operations Research*, 2000.

[14] Healy, P. ;Moll, R. In *A new extension of local search applied to the dial-a-ride problem*, 1995.

[15] Lourenco, H. R.; Martin, O. C.; Stutzle, T. *Handbook of Metaheuristics, Cap. 11*. Kluwer Academic Publishers, 2003.

[16] Hansen, P.; Mladenovic, N. "An introduction to variable neighborhood search". In *MetaHeuristics: Advances and Trends in Local Search Paradigms for Optimization*, edited by I. O. Voss, S. Martello and C. Roucairol. Kluwer, 1999.

[17] Subramanian, A.; Drummond, L. M. A.; Bentes, C.; Ochi, L. S. e Farias, R. "A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery". In *Computers and Operations Research*, pp. 1899–1911, 2010.

[18] J. Dethloff. "Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up". In *OR Spektrum 23*, pp. 79–96, 2001.

[19] Aiex, R. M.; Resende, M. G. C.; Ribeiro, C. C. "TTTPLOTS: A Perl program to create time-to-target plots". In *Optimization Letters*, 2007.

[20] Santos, H. G., Ochi, L. S., Marinho, E. H., Drummond, L. M. A. "Combining an Evolutionary Algorithm with Data Mining to solve a Vehicle Routing Problem". In *Neurocomputing Journal*, pp. 70–77. Elsevier, 2006.