Uma Heurística Baseada em ILS Para o Problema do Caixeiro Alugador

Sávio S. Dias, Luiz Satoru Ochi, Victor M. C. Machado
Instituto de Computação - Universidade Federal Fluminense
Av. Gal. Milton Tavares de Souza, s/nº, São Domingos - Niterói - RJ
{sdias,satoru,vmouffron}@ic.uff.br

Luidi Gelabert Simonetti
Departamento de Computação e Engenharia de Sistemas - Universidade Federal do Rio de Janeiro
Cidade Universitária, Centro de Tecnologia - Rio de Janeiro - RJ
luidi@cos.ufrj.br

André Renato Villela da Silva
Departamento de Computação - ICT/UFF
R. Recife s/n Jardim Bela Vista, Rio das Ostras/RJ CEP: 28895-532
avillela@ic.uff.br

RESUMO

Este trabalho aborda o Problema do Caixeiro Alugador (PCA), uma variante do Problema do Caixeiro Viajante. No PCA um turista deseja viajar por um conjunto de cidades alugando veículos a um custo mínimo. O objetivo é determinar a rota ótima e o tipo de veículo que será alugado para cumprir cada viagem. Por se tratar de um problema $\mathcal{NP}$-Difícil, neste artigo é apresentado uma abordagem heurística para sua resolução. A abordagem é baseada em uma metaheurística *Multi-Start Iterated Local Search* com uma etapa de busca local inspirada na metodologia de *Random Variable Neighborhood Descent*. Os resultados computacionais obtidos para as instâncias euclidianas são comparados com o estado da arte, mostrando que a abordagem proposta possui um melhor desempenho.

PALAVRAS CHAVE. Problema do Caixeiro Alugador. Busca Local Iterada. Otimização Combinatória.

Tópicos (Metaheurísticas, Otimização Combinatória, Logística e Transportes)

ABSTRACT

This paper tackles the Car Renter Salesman Problem (CaRS), a Traveling Salesman Problem variant. In CaRS a tourist intends to travel along a set of cities renting vehicles at a minimum cost. The goal is to establish the optimal route and the rented vehicle type assigned to each trip. Since CaRS is $\mathcal{NP}$-Hard, this paper presents a heuristic approach to solve it. This approach is based on a Multi-Start Iterated Local Search metaheuristic with a local search step based on the Random Variable Neighborhood Descent methodology. The computational results obtained for euclidean instances show that the proposed method outperfoms the state-of-art.

KEYWORDS. Car Renter Salesman. Iterated Local Search. Combinatorial Optimization.

Paper topics (Metaheuristics, Combinatorial Optimization, Logistics and Transport)

1. Introdução

O Problema do Caixeiro Viajante (PCV) é um dos problemas mais estudados em Otimização Combinatória. Este problema é $\mathcal{NP}$-Díficil e tem como objetivo encontrar um ciclo Hamiltoniano de custo mínimo em um grafo com peso nas arestas [Garey e Johnson, 1979]. A alta aplicabilidade do PCV em situações reais gerou grande interesse por esse problema e por variantes que abordam aplicações práticas mais complexas, como o PCV com Seleção de Hotéis [Vansteenwegen et al., 2012], Problema do Caixeiro Múltiplo [Bektas, 2006], PCV Colorido [Xiong et al., 2007].

Uma das variantes menos exploradas é o Problema do Caixeiro Alugador (PCA) que lida com o mercado de aluguel de carros. Este mercado é um dos segmentos mais investigados nos recentes trabalhos de Pesquisa Operacional devido ao seu crescimento significante nos últimos anos [Seay e Narsing, 2013]. Embora alguns trabalhos tenham levado em consideração a companhia provedora dos serviços, com o objetivo de maximizar os ganhos, poucos levam em consideração o lado do consumidor.

No PCA, um turista normalmente deseja conhecer algumas localidades viajando com um custo mínimo. Esses custos variam de acordo com o tipo do carro alugado e algumas taxas específicas. O Problema do Caixeiro Alugador é uma generalização do PCV clássico, e foi proposto inicialmente em Goldbarg et al. [2012]. Este problema possui aplicações na área de Engenharia de Transportes, podendo representar importantes situações no setor de aluguel de veículos para turismo, e também em manufatura flexível [Global Industry Analysts, 2014].

Neste problema, o alugador (ou turista) deseja viajar entre um conjunto de lugares ou cidades, usando um veículo. Existe um conjunto de tipos de veículos disponíveis e o turista pode escolher qual carro usar para cumprir cada viagem. Neste trabalho, é assumido como viagem um trecho do ciclo Hamiltoniano percorrido por um tipo de veículo. Deste modo, a união de todas as viagens levará à solução. Partindo desse pressuposto, muitas variantes podem surgir com a restrição de e.g., disponibilidade do tipo do veículo, lugares para onde um veículo pode ser retornado, utilização de um tipo de veículo mais de uma vez.

A tarefa de resolver o PCA não é trivial, uma vez que sua versão de otimização é $\mathcal{NP}$-Díficil [Goldbarg et al., 2012], i.e. não se conhece algoritmo capaz de encontrar soluções ótimas em tempo polinomial. Como consequência, métodos aproximados são frequentemente utilizados para a resolução do problema, visto que o elevado custo computacional exigido para sua resolução torna inviável, em muitos casos, o uso de técnicas exatas. Neste contexto, as metaheurísticas surgem como alternativas aos métodos exatos ao obter, de maneira geral, soluções de boa qualidade em um tempo significativamente menor.

Algumas heurísticas foram propostas na literatura para resolver este problema. Para instâncias euclidianas e não-euclidianas foi apresentado um GRASP (*Greedy Randomized Adaptive Search Procedure*) com VND (*Variable Neighborhood Descent*) e um Algoritmo Memético [Goldbarg et al., 2012]. Um Algoritmo Transgenético [Goldbarg et al., 2013], metaheurística inspirada na evolução mutualística intracelular endo-simbiótica, configura o atual estado-da-arte para as instâncias euclidianas deste problema. Foi proposto também uma Heurística baseada no Método Científico para resolver algumas instâncias não-euclidianas [Felipe et al., 2014]. Uma formulação quadrática também foi oferecida em Goldbarg et al. [2012], a qual foi linearizada e utilizada para a resolução na otimalidade de instâncias com até 16 vértices e 2 veículos. da Silva Menezes et al. [2014] propôs uma nova variante do PCA, com coleta de prêmios, junto a um Algoritmo Memético para resolvê-la.

2. Descrição do Problema

Dado um grafo $G = (V, E)$, no qual V ($|V| = n$) é o conjunto de vértices ou cidades e E é o conjunto de arestas entre cada par de vértices em V. Um grupo de diferentes tipos de veículos C está disponível em cada vértice. Associado a cada tipo de veículo $c \in C$ e par de vértices $i, j \in V$, $i \neq j$, temos:

- D_{ij}^c - Custo operacional para viajar de i para j usando o tipo de veículo c, ou peso da aresta (i, j) com o veículo c;
- F_{ij}^c - Taxa de retorno (ou devolução) que precisa ser paga toda vez que um carro do tipo c é alugado no vértice i e entregue no vértice j.

Com estes dados, o objetivo é estabelecer um ciclo Hamiltoniano com custo total mínimo, incluindo custos operacionais e taxa de retorno. O turista também deve iniciar e finalizar o ciclo no vértice inicial. Logo, obrigatoriamente, o primeiro veículo é alugado em v_1 e o último entregue em v_1.

A versão do problema tratada neste artigo também obedece a algumas restrições e características. São elas:

- Qualquer veículo pode ser alugado e entregue em qualquer vértice;
- Qualquer tipo de veículo pode ser alugado no máximo uma vez, i.e. se um carro do tipo c já foi entregue, ele não pode mais ser alugado;
- A taxa de retorno não depende da topologia do grafo, ou quaisquer outras restrições;
- Os custos operacionais são simétricos, i.e. $D_{ij}^c = D_{ji}^c$. Porém isso não garante que $F_{ij}^c = F_{ji}^c$.

Na Figura 1 temos um exemplo de solução viável para um exemplo com 5 vértices e 2 veículos. O grafo com o peso das arestas para o veículo 1 (Figura 1.a.) e veículo 2 (Figura 1.b.) são mostrados, junto com o custo final da solução (Figura 1.c.). Este custo é calculado pelo valor das arestas percorridas por cada veículo adicionado das taxas de retorno $F_{15}^2 = 10$ e $F_{51}^1 = 30$.

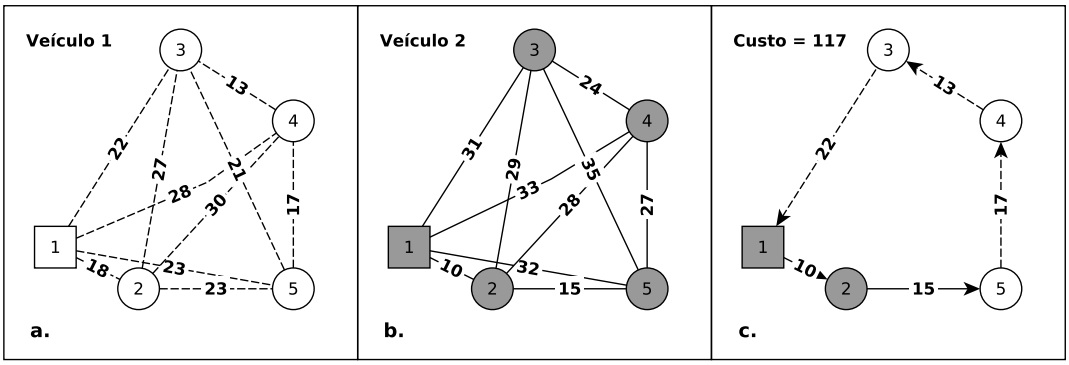

Figura 1: Custos operacionais para Veículos 1 e 2 e solução viável para um exemplo com $n = 5$ e $|C| = 2$.

As instâncias deste problema podem seguir um modelo euclidiano ou não-euclidiano para definir as distâncias entre os vértices. Neste trabalho, somente o grupo de instâncias euclidianas foi considerado para uma avaliação inicial do método proposto em relação ao estado-da-arte.

3. Algoritmo Proposto

Pela natureza do problema não permitir que algoritmos o resolvam otimamente em tempo polinomial, os métodos exatos trazem, em muitos casos, um elevado custo computacional, portanto um método heurístico foi considerado. Foi adotada uma heurística baseada em ILS [Lourenço et al., 2010] com comportamento *Multi-Start* (MS-ILS) para a resolução do PCA. Algoritmos MS-ILS têm sido utilizados com sucesso na literatura recente para diversas variantes do PCV [Subramanian e Battarra, 2013; Silva et al., 2012]. Essa heurística é apresentada no Algoritmo 1.

O algoritmo constrói uma solução usando um critério guloso aleatório para cada uma das ms_{max} iterações, baseado nos dados de entrada da instância (Linha 4). A solução construída passa

por ils_{max} aplicações de um procedimento de busca local (Linha 6), utilizado para refinar a solução corrente s', e de um procedimento de perturbação (Linha 10), utilizado com o intuito de escapar de ótimos locais. Nas Linhas 7-9, é verificado o critério de aceitação dessa solução refinada, sendo puramente elitista, i.e. s' só será aceita caso ela seja melhor que a melhor solução global, s^*.

Algoritmo 1: MS-ILS(entrada, ms_{max}, ils_{max}, α, seed)

```
1  início
2      s* ← ∅; f(s*) ← ∞
3      para i ← 1, ..., ms_max faça
4          s' ← construir_sol(entrada, α, seed)
5          para j ← 1, ..., ils_max faça
6              s' ← e_RVND(s', seed)
7              se f(s') < f(s*) então
8                  s* ← s'
9              fim
10             s' ← perturbacao(s', seed)
11         fim
12     fim
13     Retornar s*
14 fim
```

Os métodos de construção da solução, busca local e perturbação serão detalhados nas Seções 3.2, 3.3 e 3.4 respectivamente.

3.1. Representação da Solução

Para representação da solução, foram utilizadas duas estruturas de dados. São elas:

- `rota` - Um vetor de dimensão n com a ordem de percurso da rota, tal que `rota[i]` $\in V$, $\forall i = 0, ..., n-1$;
- `v_pos` - Um vetor de dimensão 3 x $|C|$, tal que para cada viagem temos os dados: posição do vértice, no vetor `rota`, onde o veículo foi alugado; posição do vértice, no vetor `rota`, onde o veículo foi retornado; e o tipo do veículo em questão. Este vetor mantem a ordem de uso dos veículos. Caso o número de viagens for menor que o número de veículos disponíveis, as posições restantes deste vetor serão nulas.

Desta forma, uma solução como a da Figura 1 é representada como: `rota` $= [1, 2, 5, 4, 3]$; `v_pos` $= [(0, 2, 2), (2, 5, 1)]$. Note que no caso do último veículo, o ponto final de retorno é uma posição inválida do vetor `rota`.

3.2. Método Construtivo

A etapa de construção da solução inicial pode ser observada no Algoritmo 2. Esse algoritmo possui componentes gulosas e aleatórias, como a proposta por Feo e Resende [1989]. A ideia do algoritmo é construir uma rota diversa e bem distribuída em vértices que utilize todos os veículos, para que a etapa de busca local tenha um espaço de busca mais variado de modo a alcançar melhores soluções.

Inicialmente, o algoritmo define a Lista de Candidatos (LC) com todos os vértices do grafo, exceto o vértice inicial, e inicializa o conjunto de viagens vazias com todos os veículos possíveis da instância (Linhas 2 e 3). O número de vértices para cada veículo é determinado igualmente (Linhas 4 e 5), sendo que o último pode ter um vértice a mais, caso a divisão seja fracionária. Em seguida, para cada veículo é construída uma viagem, iniciando o aluguel pelo vértice inicial.

O veículo selecionado aleatoriamente é retirado do conjunto de viagens vazias, e em seguida é adicionado à sua viagem o vértice em que será alugado (Linhas 8-10). Após este passo,

é montada uma Lista Restrita de Candidatos (LRC), com os elementos presentes em LC que satisfaçam à Equação 1, para que o vértice em que o veículo será devolvido (e consequentemente, um novo será alugado) seja escolhido aleatoriamente (Linhas 11-14). Observe que, caso este seja o último veículo, o local de devolução deve ser obrigatoriamente o vértice de origem. Na Equação 1, a é o vértice de aluguel do veículo c na viagem corrente, s^c, e α é um parâmetro que define qual o requisito mínimo para um vértice r ser aceito como possível vértice de devolução de c.

$$LRC = \left\{ r \in LC | F^c_{ar} \leq \min_{j \in LC}(F^c_{aj}) + \alpha \left[\max_{j \in LC}(F^c_{aj}) - \min_{j \in LC}(F^c_{aj}) \right] \right\}. \quad (1)$$

Algoritmo 2: construir_sol(entrada, α, seed)

```
1  início
2      Inicializar LC
3      Seja S = {s^1, ..., s^|C|} o conjunto de viagens vazias
4      para i ← 1, ..., |C| − 1 faça w_{s^i} ← ⌊n/|C|⌋
5      w_{s^|C|} ← ⌈n/|C|⌉
6      a ← 1 ; S̄ ← ∅
7      enquanto S ≠ ∅ faça
8          Selecione s^c ∈ S aleatoriamente
9          S ← S − {s^c}
10         s^c ← s^c ∪ {a}
11         se S = ∅ então LRC ← {1}
12         senão Criar LRC
13         d ← a
14         a ← elemento aleatório da LRC
15         LC ← LC −{a}
16         enquanto LC ≠ ∅ e |s^c| < w_{s^c} faça
17             k' ← argmin_{k∈LC} {g(k, a, d)}
18             s^c ← s^c ∪ {k'}
19             LC ← LC −{k'}
20         fim
21         S̄ ← S̄ ∪ s^c
22     fim
23     Retornar S̄
24 fim
```

Definidos os vértices de aluguel e retorno do veículo escolhido, a viagem é montada elemento a elemento avaliando a inserção para todos os vértices da LC em cada posição da viagem. Esta avaliação é feita por meio da Equação 2, baseada em Penna et al. [2013], que visa evitar inserções tardias de vértices localizados longe dos pontos de aluguel e devolução do veículo.

$$g(k,a,d) = \begin{cases} D^c_{ik} - \gamma(D^c_{dk} + D^c_{ka}) & \text{se inserção no inicio ou fim} \\ (D^c_{ik} + D^c_{kj} - D^c_{ij}) - \gamma(D^c_{dk} + D^c_{ka}) & \text{se inserção no meio} \end{cases}. \quad (2)$$

Onde i e j são, respectivamente, vértices anterior e seguinte à posição onde se deseja inserir k na rota e γ é um valor, escolhido aleatoriamente dentro do conjunto $\{0.00, 0.05, 0.10, ..., 1.65, 1.70\}$, que pondera a importância da distância do vértice k para os vértices de aluguel e devolução do veículo na avaliação da inserção de k nesta viagem. Este conjunto foi definido em Subramanian e Battarra [2013] após a realização de experimentos preliminares.

O vértice $k' \in LC$ que tiver o menor valor de $g(k, a, d)$ será adicionado à viagem s^c e removido de LC. Este procedimento é executado até que s^c atinja seu número máximo de vértices (Linhas 16-20). Após definida a viagem s^c, esta será adicionada ao final da rota $\bar{S}$ (Linha 21). Ao final, as viagens de todos os veículos estarão presentes em $\bar{S}$, que será retornado como solução.

3.3. Busca Local

A metodologia de busca local adotada foi uma variação do VND proposto por Mladenović e Hansen [1997], chamada *Random Variable Neighborhood Descent* (RVND), onde a ordem de aplicação das vizinhanças é definida de forma aleatória. Estratégia semelhante foi usada em Penna et al. [2013].

Neste trabalho as estruturas de vizinhança foram divididas em dois grupos, que diferenciam sua aplicação. As vizinhanças são classificadas como inter e intra-viagem. Este agrupamento foi feito devido à natureza da aplicação destas. Enquanto as vizinhanças inter-viagem realizam operações levando em consideração viagens de veículos diferentes, as intra-viagem trabalham com a vizinhança de uma mesma viagem. Estas vizinhanças serão detalhas nas Seções 3.3.1 e 3.3.2.

Algoritmo 3: inter_RVND(s, seed)

```
1  início
2      ListaVizinhancasInter ← iniciar_aleatorio(seed)
3      i ← 0
4      enquanto i < NUM_VIZ_INTER faça
5          s' ← aplicar_vizinhanca(ListaVizinhancasInter[i], s)
6          se f(s') < f(s) então
7              s ← s'
8              s ← i_RVND(s, seed)
9              i ← 0
10         senão i ← i + 1
11     fim
12     Retornar s
13 fim
```

No Algoritmo 3 é demonstrado o funcionamento da metodologia, onde a ordem de aplicação das vizinhanças inter-viagem é decidida aleatoriamente no início (Linha 2). Nas iterações seguintes, a vizinhança i definida aleatoriamente é aplicada (Linha 5). Caso a aplicação desta vizinhança melhore a solução de referência, um novo RVND é aplicado sobre a solução melhorada usando apenas vizinhanças intra-viagem (Linhas 6-9). Caso não haja melhora, o algoritmo tenta aplicar uma nova vizinhança inter-viagem (Linha 10). Ao final, ao aplicar `NUM_VIZ_INTER` vizinhanças inter-viagem sem melhoras, o algoritmo retorna a melhor solução encontrada.

Algoritmo 4: intra_RVND(s, seed)

```
1  início
2      ListaVizinhancasIntra ← iniciar_aleatorio(seed)
3      i ← 0
4      enquanto i < NUM_VIZ_INTRA faça
5          s' ← aplicar_vizinhanca(ListaVizinhancasIntra[i], s)
6          se f(s') < f(s) então
7              s ← s'
8              i ← 0
9          senão i ← i + 1
10     fim
11     Retornar s
12 fim
```

Quando uma nova melhor solução é encontrada por alguma vizinhança inter-viagem, o Algoritmo 4 é chamado de modo a melhorar esta solução do ponto de vista de cada viagem individualmente. O funcionamento é semelhante ao do Algoritmo 3, com a diferença de que aqui são aplicadas somente as NUM_VIZ_INTRA vizinhanças intra-viagem. Ao final, o vizinho com menor custo será retornado.

3.3.1. Estruturas de Vizinhança Inter-Viagem

Nove vizinhanças inter-viagem foram implementadas, sendo que sete delas foram baseadas nas vizinhanças de trocas clássicas para o Problema de Roteamento de Veículos (PRV) [Osman, 1993]. As outras duas vizinhanças são propostas neste trabalho especificamente para o PCA, que consistem em alterar o tamanho da viagem de um veículo e a ordem de utilização dos veículos.

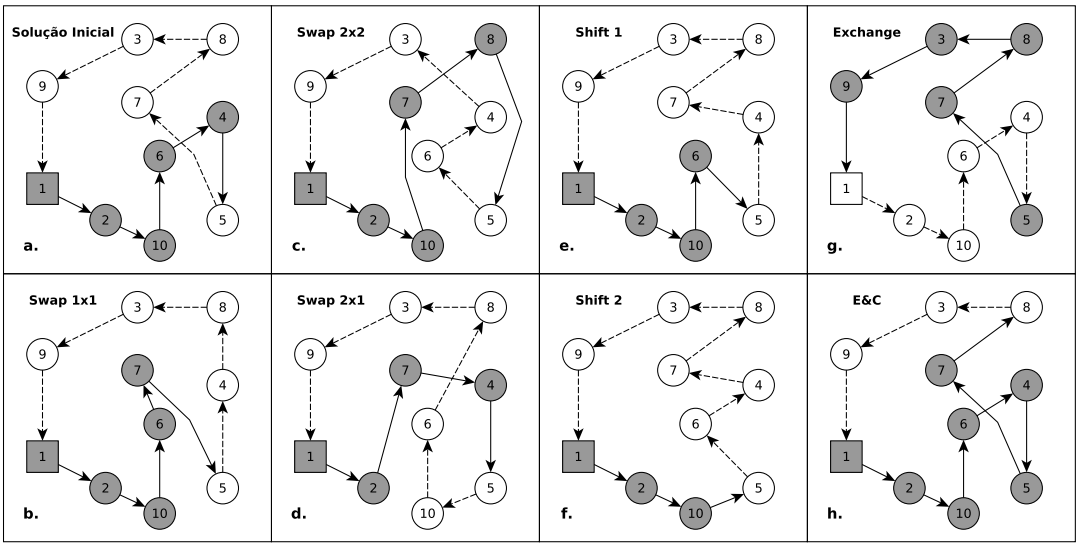

Figura 2: Vizinhanças Inter-Viagem.

As soluções encontradas por todas as estruturas de vizinhanças foram obtidas através de exploração exaustiva, levando em consideração todas as combinações e escolhendo a melhor delas, i.e. estratégia de *best improvement*. Para as vizinhanças baseadas nas de troca clássicas, os vértices onde os veículos são alugados e devolvidos não são considerados. Desta forma, as estruturas de vizinhança inter-viagem são:

- ***Swap* 1x1** - Troca entre um vértice $i \in c_i$ com um vértice $j \in c_j$, onde c_i e c_j são viagens tais que $c_i \neq c_j$. Desta forma, ao aplicar a vizinhança, $i \in c_j$ e $j \in c_i$. Na Figura 2.b. os vértices 4 e 7 das viagens dos veículos 1 e 2 são trocados.

- ***Swap* 2x2** - Troca entre dois vértices adjacentes $i, i+1 \in c_i$ com dois outros vértices adjacentes $j, j+1 \in c_j$, onde c_i e c_j são viagens tais que $c_i \neq c_j$. Desta forma, ao aplicar a vizinhança, $i, i+1 \in c_j$ e $j, j+1 \in c_i$. Os vértices 6 e 4 são trocados pelos vértices 7 e 8, de suas respectivas viagens, na Figura 2.c.

- ***Swap* 2x1** - Troca entre dois vértices adjacentes $i, i+1 \in c_i$ com um vértice $j \in c_j$, onde c_i e c_j são viagens tais que $c_i \neq c_j$. Desta forma, ao aplicar a vizinhança, $i, i+1 \in c_j$ e $j \in c_i$. Troca de 10 e 6 por 7 na Figura 2.d.

- ***Swap* 3x3** - Troca entre três vértices adjacentes $i, i+1, i+2 \in c_i$ com três outros vértices adjacentes $j, j+1, j+2 \in c_j$, onde c_i e c_j são viagens tais que $c_i \neq c_j$. Desta forma, ao aplicar a vizinhança, $i, i+1, i+2 \in c_j$ e $j, j+1, j+2 \in c_i$.

- ***Shift* 1** - Um vértice $i \in c_i$ é transferido para uma viagem c_j, com $c_i \neq c_j$. A inserção $i \in c_j$ é considerada em todas as posições da viagem, exceto o vértice de aluguel e/ou devolução do veículo. Na Figura 2.e., o vértice 4 é deslocado para depois do vértice 5, na viagem 2.

- ***Shift* 2** - Dois vértices adjacentes $i, i+1 \in c_i$ são transferidos para uma viagem c_j, com $c_i \neq c_j$. A inserção $i, i+1 \in c_j$ é considerada em todas as posições da viagem, exceto o vértice de aluguel e/ou devolução do veículo. Na Figura 2.f. Os vértices 6 e 4 foram deslocados para depois do vértice 5 na viagem 2.

- ***Shift* 3** - Três vértices adjacentes $i, i+1, i+2 \in c_i$ são transferidos para uma viagem c_j, com $c_i \neq c_j$. A inserção $i, i+1, i+2 \in c_j$ é considerada em todas as posições da viagem, exceto o vértice de aluguel e/ou devolução do veículo.

- ***Vehicle Exchange*** - Troca de viagens entre dois veículos. Seja c_1 o veículo que faz a viagem 1 e c_2 o veículo que faz a viagem 2. Após a troca, os veículos c_1 e c_2 passarão a fazer, respectivamente, as viagens 2 e 1. Na Figura 2.g. essa troca é exemplificada. Observe que nessa vizinhança, em alguns casos, toda a solução deverá ser recalculada, pois há alteração nos vértices de aluguel e devolução dos veículos bem como nos pesos das arestas em cada viagem.

- ***Extend&Contract* (E&C)** - Modifica os vértices de aluguel e devolução de veículos com viagens adjacentes. As alterações vão sendo avaliadas para cada vértice que for alterado da viagem. A ideia é ir estendendo a viagem de um veículo, ao mesmo tempo em que diminui a viagem de outro adjacente. No exemplo da Figura 2.h. a primeira viagem foi estendida do vértice 5 (ponto de devolução do veículo) para o vértice 8. Deste modo, o veículo responsável por cumprir a viagem 1 passará a ser alugado no vértice 1 e devolvido no vértice 8, e o veículo da viagem 2 será alugado no vértice 8 e devolvido no vértice 1.

A complexidade para as vizinhanças de trocas clássicas é de $\mathcal{O}(n^2)$. Para a vizinhança *Vehicle Exchange*, $\mathcal{O}(|C|^2 n)$, e para a vizinhança E&C é $\mathcal{O}(|C|n)$. Para as vizinhanças propostas neste artigo que tratam diretamente da alteração das viagens, o fator $\mathcal{O}(n)$ aparece na complexidade devido à necessidade de recálculo de uma porção não-constante da solução, enquanto que para as demais vizinhanças a solução pode ser reavaliada em $\mathcal{O}(1)$.

3.3.2. Estruturas de Vizinhança Intra-Viagem

Oito vizinhanças intra-viagem foram implementadas, sendo todas baseadas nas vizinhanças de trocas clássicas do PRV. A estratégia de *best improvement* também foi adotada para encontrar o vizinho de menor custo. Para todas as vizinhanças, exceto a *Reverse*, os vértices de aluguel e devolução dos veículos não são considerados. Assim sendo, temos:

- ***Swap* 1x1, 2x2 e 3x3** - Semelhante ao *Swap* 1x1, 2x2 e 3x3 da Seção 3.3.1, porém as trocas são feitas apenas com vértices internos da viagem selecionada.

- ***Shift* 1, 2 e 3** - Semelhante ao *Shift* 1, 2 e 3 da Seção 3.3.1, porém as transferências são feitas apenas com vértices internos da viagem selecionada.

- **2-*Opt*** - Duas arestas não adjacentes são excluídas e outras duas são adicionadas de forma a gerar uma nova rota. Na Figura 3.b. as arestas $(6, 4)$ e $(5, 7)$ da viagem 1 são excluídas, e as arestas $(6, 5)$ e $(4, 7)$ são adicionadas.

- ***Reverse*** - Essa vizinhança inverte a viagem de um veículo c caso o valor de $F^c_{ij} > F^c_{ji}$, i.e. se o custo de devolver o veículo no vértice de aluguel for menor que o original. Observe que essa vizinhança só é aplicável caso $|C| > 2$, pois mudar o vértice de aluguel da primeira

viagem e o de devolução da última tornaria a solução inviável. A viagem 2 da Figura 3.c. é um exemplo da aplicação desta vizinhança, onde o vértice 8 é o novo local de aluguel do veículo.

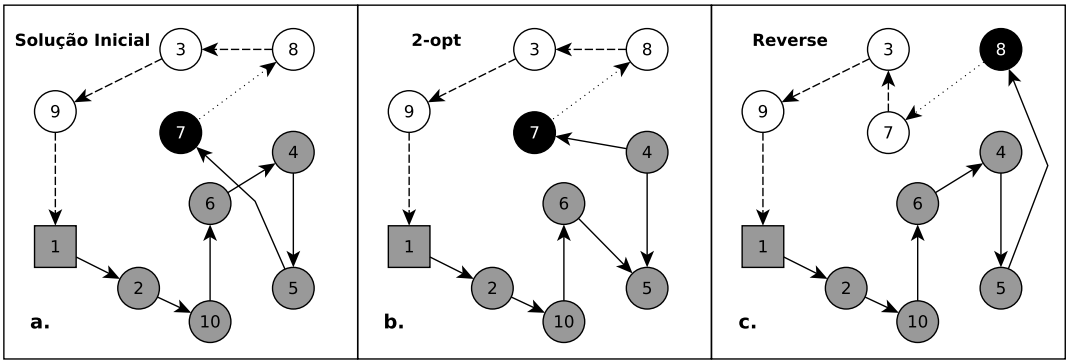

Figura 3: Vizinhanças Intra-Viagem.

A complexidade de avaliação das vizinhanças de *Swap*, *Shift* e 2-*opt* é $\mathcal{O}(n^2)$, enquanto para a vizinhança *Reverse* é $\mathcal{O}(n)$.

3.4. Perturbação

Como mecanismo de perturbação foram adaptadas três vizinhanças inter-viagem: *Shift* 1, *Swap* 1x1 e *Vehicle Exchange*. Para estas, os vértices e veículos foram selecionados de forma aleatória, e não mais levando em consideração o critério de melhoria da solução. Logo, a alteração é efetuada imprevisivelmente, possivelmente piorando a solução, porém permitindo explorar melhor o espaço de busca. A perturbação aplicada é escolhida aleatoriamente durante a execução.

4. Resultados Computacionais

O método proposto foi codificado em C++ usando o compilador g++ 4.8.2. Os testes foram executados em um computador com arquitetura Intel i7-870 2,93GHz e 8GB de memória RAM. O sistema operacional utilizado foi o Ubuntu 14.04 64-bit, sendo que todos os testes foram executados por uma única *thread*.

Os parâmetros do algoritmo foram definidos de maneira empírica, com os valores $\alpha = 0,25$, $ms_{max} = 20$ e $ils_{max} = 50$. Foram realizadas 30 execuções para cada instância. Os experimentos foram realizados com sementes controladas, i.e. o valor de *seed* é determinado como o número da execução corrente no intervalo discreto $[1, 30]$. As instâncias utilizadas, fornecidas por Goldbarg et al. [2012], estão disponíveis em `http://www.dimap.ufrn.br/lae/en/projects/CaRS.php`.

Na Tabela 1 os resultados médios obtidos com as 30 execuções da metodologia proposta por este artigo (MS-ILS) são comparados com um Algoritmo Memético (MA) [Goldbarg et al., 2012], e o estado-da-arte (TA) [Goldbarg et al., 2013]. Nesta tabela, as informações da instância são dados no **Nome**, embutido da quantidade de vértices, e o número de tipos de veículos disponíveis, $|C|$. Para as outras abordagens da literatura o valor médio da solução obtida, **Sol**, e o tempo de execução em segundos, **T(s)**, são apresentados. Estes resultados, também codificados em C++, foram obtidos em uma arquitetura Intel Xeon QuadCore W3520 2,8GHz com 8GB de RAM executando um Scientific Linux 5.5 64 bits. As colunas com os resultados do MS-ILS apresentam também o **GAP** $= 100(S_1 - S_2)/S_2$, onde S_1 é o resultado do MS-ILS, em relação à melhor solução heurística. Para cada instância, a heurística que obteve a melhor solução média é destacada em negrito.

Na Tabela 2 é feita uma comparação quanto à melhor solução encontrada em todas as execuções para as respectivas metodologias. Nesta tabela, a coluna **Best Sol** traz o melhor resultado

Tabela 1: Resultados médios para instâncias euclidianas

Instância		**MA**		**TA**		**MS-ILS**		
Nome	$\|C\|$	**Sol**	**T (s)**	**Sol**	**T (s)**	**Sol**	**GAP**	**T (s)**
BrasilRJ14e	2	**294**	1	**294**	1	**294,00**	0,00%	0,33
BrasilRN16e	2	**375**	1	**375**	1	**375,00**	0,00%	0,34
BrasilPR25e	3	523	2	**508**	12	**508,00**	0,00%	1,17
BrasilAM26e	3	477	3	**467**	13	505,00	8,14%	1,11
BrasilMG30e	4	549	8	532	26	**529,00**	-0,56%	2,10
BrasilSP32e	4	606	7	593	27	**588,77**	-0,71%	2,78
BrasilRS32e	4	496	7	**493**	24	545,97	10,74%	2,19
BrasilCO40e	5	696	23	676	51	**671,37**	-0,69%	6,65
BrasilNO45e	5	857	30	840	70	**832,57**	-0,88%	7,98
BrasilNE50e	5	766	35	763	70	**759,97**	-0,40%	11,81
Livramento30e	3	**739**	2	**739**	16	**739,00**	0,00%	1,71
Pelotas50e	3	1.288	17	1.265	77	**1.253,63**	-0,90%	7,73
BoaVista80e	4	1.725	49	1.666	350	**1.608,10**	-3,48%	36,74
Betim100e	3	1.401	247	1.408	348	**1.396,97**	-0,29%	59,39
Vitoria100e	5	**1.357**	292	1.362	382	1.402,13	3,33%	70,36
JoaoPessoa140e	4	2.557	249	2.461	1.297	**2.288,77**	-7,00%	176,17
Natal160e	5	2.792	298	2.688	1.944	**2.570,40**	-4,38%	367,91
PortoVelho200e	3	2.375	1.862	2.376	3.165	**2.299,80**	-3,17%	587,32
Cuiaba200e	3	2.398	1.682	2.332	3.039	**2.243,43**	-3,80%	698,85
Belem300e	4	3.095	5.184	3.056	9.649	**3.030,80**	-0,82%	2.731,24
att48eA	3	**34.572**	14	34.643	39	34.591,17	0,06%	3,79
berlin52eA	3	8.950	43	**8.949**	59	8.960,20	0,13%	6,46
ch130e	5	8.931	190	**8.828**	893	8.854,57	0,30%	120,62
eil76eB	4	1.826	184	1.779	420	**1.667,97**	-6,24%	36,39
lin105e	5	17.053	100	**16.988**	384	17.191,10	1,20%	56,58
pr107e	5	46.867	104	**46.840**	394	47.046,37	0,44%	64,03
rat99eB	5	3.188	248	3.133	309	**2.971,10**	-5,17%	78,06
rd100eB	4	9.954	255	**9.951**	255	10.026,63	0,76%	37,96
st70eB	4	1.898	148	1.858	310	**1.763,50**	-5,09%	28,94
w100eB	4	**8.310**	82	**8.310**	331	8.493,17	2,20%	40,86
Médias			378,90		1.023,08		**-0,72%**	**187,97**

de cada heurística nas 30 execuções. Na coluna **GAP** é apresentada a diferença relativa do MS-ILS com a melhor solução conhecida até este trabalho.

Pelos resultados da Tabela 1, é observado que o MS-ILS fornece limites superiores de alta qualidade a um baixo custo computacional quando comparado com as demais heurísticas existentes. O GAP médio em relação às demais heurísticas propostas na literatura foi negativo, obtendo os melhores desempenhos médios em 20 das 30 instâncias testadas em um tempo médio quase cinco vezes menor que o atual estado-da-arte[1]. Pela Tabela 2 é observado que a abordagem ainda foi capaz de propor novas melhores soluções em nove instâncias, além de empatar em outras 14. Em apenas 7 ocasiões o algoritmo não foi capaz de atingir as melhores soluções conhecidas para as instâncias euclidianas.

5. Conclusões

Este artigo apresentou uma nova abordagem para o Problema do Caixeiro Alugador, usando uma metodologia baseada na metaheurística *Iterated Local Search* com uma busca local intensiva através de várias vizinhanças clássicas para problemas de roteamento e algumas novas propostas para este problema em específico.

Os testes realizados provaram que a metologia proposta foi capaz de reduzir o tempo computacional requerido para encontrar uma solução de boa qualidade, em muitos casos tendo

[1]Esta comparação é possível devido a dados encontrados em `http://www.cpubenchmark.net/cpu_list.php`, que mostram que ambas arquiteturas possuem desempenho similar.

Tabela 2: Melhores resultados para instâncias euclidianas

Instância		**MA**	**TA**	**MS-ILS**	
Nome	$\|C\|$	**Best Sol**	**Best Sol**	**Best Sol**	**GAP**
BrasilRJ14e	2	**294**	**294**	**294**	0,00%
BrasilRN16e	2	**375**	**375**	**375**	0,00%
BrasilPR25e	3	510	**508**	**508**	0,00%
BrasilAM26e	3	468	**467**	505	8,14%
BrasilMG30e	4	530	**529**	**529**	0,00%
BrasilSP32e	4	**588**	**588**	**588**	0,00%
BrasilRS32e	4	494	**491**	541	10,18%
BrasilCO40e	5	672	**668**	**668**	0,00%
BrasilNO45e	5	**829**	**829**	**829**	0,00%
BrasilNE50e	5	**756**	**756**	**756**	0,00%
Livramento30e	3	**739**	**739**	**739**	0,00%
Pelotas50e	3	1.274	**1.249**	**1.249**	0,00%
BoaVista80e	4	1.707	1.591	**1.575**	-1,01%
Betim100e	3	**1.394**	**1.394**	**1.394**	0,00%
Vitoria100e	5	**1.354**	**1.354**	**1.354**	0,00%
JoaoPessoa140e	4	2.521	2.368	**2.235**	-5,62%
Natal160e	5	2.753	2.641	**2.532**	-4,13%
PortoVelho200e	3	2.327	2.312	**2.281**	-1,34%
Cuiaba200e	3	2.329	2.275	**2.220**	-2,42%
Belem300e	4	3.007	**2.985**	2.991	0,20%
att48eA	3	**34.571**	**34.571**	**34.571**	0,00%
berlin52eA	3	**8.948**	**8.948**	**8.948**	0,00%
ch130e	5	8.818	8.729	**8.653**	-0,87%
eil76eB	4	1.756	1.703	**1.647**	-3,29%
lin105e	5	**16.916**	**16.916**	16.981	0,38%
pr107e	5	46.827	**46812**	46.836	0,05%
rat99eB	5	3.113	3.042	**2.929**	-3,71%
rd100eB	4	**9.909**	**9.909**	9.962	0,53%
st70eB	4	1.849	1.777	**1.714**	-3,55%
w100eB	4	**8.310**	**8.310**	8.382	0,87%
Médias					**-0,19%**

melhor desempenho que o estado da arte. Estas comparações são possíveis pois os testes foram executados em uma arquitetura semelhante à dos demais trabalhos da literatura.

Como um estudo inicial, o algoritmo provou ser promissor na exploração do PCA, entretanto mais estudos são necessários. A aplicação desta metodologia para instâncias não-euclidianas, que são as de maior aplicação prática, deve ser considerada de modo a melhor avaliar seu desempenho com o atual estado-da-arte. Uma nova possibilidade também é a de estender essa metodologia para outras variantes do PCA, como o PCA com coleta de prêmios, ou a versão sem restrição de uso único de um veículo.

Para trabalhos futuros, é esperado a implementação de Mecanismos de Filtragem da aplicação de Buscas Locais em algoritmos *Multi-Start* e de Estruturas de Dados Auxiliares, que evitam recálculo de parte da solução na aplicação de algumas vizinhanças, de modo a diminuir o tempo computacional necessário para obter uma boa solução. Desta forma, é interessante também realizar um estudo mais amplo do impacto dos parâmetros do algoritmo e das vizinhanças, podendo inclusive realizar uma aplicação genérica pseudo-aleatória para as vizinhanças de *Swap* e *Shift*, na solução final.

Agradecimentos

Os autores gentilmente agradecem o apoio fornecido pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), e o fornecimento de recursos computacionais por parte do Laboratório de Algoritmos, Grafos e Otimização (LAGO - UFF) da Universidade Federal Fluminense.

Referências

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

da Silva Menezes, M., Goldbarg, M. C., e Goldbarg, E. F. (2014). A memetic algorithm for the prize-collecting traveling car renter problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, p. 3258–3265. IEEE.

Felipe, D., Ferreira Gouvea Goldbarg, E., e Goldbarg, M. C. (2014). Scientific algorithms for the car renter salesman problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, p. 873–879. IEEE.

Feo, T. A. e Resende, M. G. C. (1989). A probabilistic heuristic for a computational difficult set covering problem. *Operations Research Letters*, 8:67–71.

Garey, M. R. e Johnson, D. S. (1979). A guide to the theory of np-completeness. *WH Freemann, New York*.

Global Industry Analysts, I. (2014). Car rental business - global strategic business report. Technical Report 338373, Research and Markets. URL `http://www.researchandmarkets.com/reports/338373/car_rental_business_global_strategic_business`.

Goldbarg, M. C., Goldbarg, E. F., Asconavieta, P. H., Menezes, M. d. S., e Luna, H. P. (2013). A transgenetic algorithm applied to the traveling car renter problem. *Expert Systems with Applications*, 40(16):6298–6310.

Goldbarg, M. C., Asconavieta, P. H., e Goldbarg, E. F. G. (2012). Memetic algorithm for the traveling car renter problem: an experimental investigation. *Memetic Computing*, 4(2):89–108.

Lourenço, H. R., Martin, O. C., e Stützle, T. (2010). Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, p. 363–397. Springer.

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

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451.

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

Seay, S. e Narsing, A. (2013). Transitioning to a lean paradigm: A model for sustainability in the leasing and rental industries. *Academy of Strategic Management Journal*, 12(1):113.

Silva, M. M., Subramanian, A., Vidal, T., e Ochi, L. S. (2012). A simple and effective metaheuristic for the minimum latency problem. *European Journal of Operational Research*, 221(3):513–520.

Subramanian, A. e Battarra, M. (2013). An iterated local search algorithm for the travelling salesman problem with pickups and deliveries. *Journal of the Operational Research Society*, 64(3): 402–409.

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

Xiong, Y., Golden, B., e Wasil, E. (2007). The colorful traveling salesman problem. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, p. 115–123. Springer.