Um Algoritmo Memético para o Problema do Caixeiro Alugador com Coleta de Prêmios

Marco César Goldbarg
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte, Natal, Brasil
gold@dimap.ufrn.br

Elizabeth Ferreira Goldbarg
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte, Natal, Brasil
beth@dimap.ufrn.br

Matheus da Silva Menezes
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte, Natal, Brasil
Universidade Federal Rural do Semi-Árido, Angicos, Brasil
matheus@ufersa.edu.br

RESUMO

Este artigo apresenta uma nova variante do problema do Caixeiro Alugador, denominado de Caixeiro Alugador com Coleta de Prêmios. Neste problema são disponibilizados um conjunto de vértices, cada um com um bônus associado e um conjunto de veículos. O objetivo do problema é determinar um ciclo que visite alguns vértices coletando, pelo menos, um bônus pré-definido e minimizando os custos de viagem através da rota, que pode ser feita com veículos de diferentes tipos. É apresentada uma formulação matemática e implementada em um *solver* produzindo resultados em sessenta e quatro instâncias. É proposto um algoritmo memético e os resultados obtidos são comparados com os resultados do *solver*.

PALAVARAS CHAVE. Caixeiro Alugador com Coleta de Prêmios, Algoritmo Memético, Otimização Combinatória.

ABSTRACT

This paper introduces a new variant of the Traveling Car Renter Problem, named Prize-collecting Traveling Car Renter Problem. In this problem, a set of vertices, each associated with a bonus, and a set of vehicles are given. The objective is to determine a cycle that visits some vertices collecting, at least, a pre-defined bonus, and minimizing the cost of the tour that can be traveled with different vehicles. A mathematical formulation is presented and implemented in a solver to produce results for sixty four instances. A memetic algorithm is proposed and its performance is evaluated in comparison to the results obtained with the solver.

KEYWORDS. Prize-collecting Traveling Car Renter Problem, memetic algorithm, combinatorial optimization.

1. Introdução

O problema do Caixeiro Alugador ou *Traveling Car Renter Problem* (CaRS) foi descrito na literatura em Goldbarg *et al*. (2012) e é uma generalização do Problema do Caixeiro Viajante (PCV), onde um cliente pretende utilizar carros alugados para visitar um determinado conjunto de cidades, minimizando o custo relacionado ao aluguel de carros e de percurso. Existem várias opções de veículos de empresas diferentes, disponíveis em cada cidade. Esta multiplicidade de opções abre uma variedade de possibilidades de escolha para o utilizador sobre os carros mais atraentes para viajar diferentes partes do percurso pretendido. No caso do turismo, os clientes costumam começar e terminar sua turnê, na mesma cidade.

O problema consiste em visitar um conjunto de cidades, começando e terminando no mesmo ponto, minimizando o custo total da rota. Para tomar a decisão sobre qual carro deve alugar em cada parte do percurso, o cliente deve considerar, além do custo de locação de cada carro disponível, os custos relacionados ao consumo de combustível e pagamento de taxas. Se um carro é alugado em uma cidade e entregue em uma diferente, o usuário deve considerar também uma taxa extra para levar o carro de volta à sua origem. Dado que o Problema do Caixeiro Viajante é *NP-hard* (Guttin e Punnen, 2002) e é um caso especial de CaRS, quando apenas um carro é usado para realizar o percurso, o problema CaRS também é um problema NP-hard.

Este artigo trata de uma variante do problema CaRS, denominda Caixeiro Alugador com Coleta de Prêmios ou *Prizecollecting Traveling Car Renter Problem* (pCaRS), que é uma generalização do Problema do Caixeiro Viajante com Coleta de Prêmios ou *The prize collecting traveling salesman problem* (PCTSP). Este último foi introduzido em Balas (1989), e também é *NP-hard*. No PCTSP uma recompensa e uma penalidade são atribuídas a cada cidade e deve-se escolher um subconjunto de cidades a serem visitadas para que o custo da rota e as penalidades associadas com cada cidade não visitada sejam minimizados e a recompensa total arrecadada deve ser de pelo menos o valor associado a um determinado parâmetro ω. O Problema pCaRS é uma variante do CaRS que, assim como no PCTSP, um bônus é associado a cada cidade. Este bônus define um nível de satisfação em visitar a cidade associada. É muito comum o caso em que um turista não pode visitar todas as atrações existentes durante uma viagem, devendo escolher as que possuam maior interesse.

Nesses casos, é interessante tentar maximizar a satisfação visitando os pontos mais atraentes. O *Mobile Tourist Guide*, apresentado por Souffriau *et al.* (2008), foi projetado para turistas que não podem visitar todos os lugares que estão interessados em uma grande cidade. Alguns problemas relacionados ao turismo foram introduzidos por Vansteenwegen e Oudheusden (2007) e um crescente volume de pesquisa tem sido dedicada a esses problemas (Vansteenwegen *et al*., 2011). Esses artigos, contudo, não abordam o problema que considera que mais de um veículo pode ser utilizado pelos turistas.

No problema pCaRS uma satisfação presumida de visita é atribuída a cada cidade e uma satisfação cumulativa pré-definida mínima deve ser atendida. Além disso, um vértice é escolhido para ser a cidade onde o passeio começa e termina, denominada cidade base. O objetivo é selecionar um subconjunto de cidades (vértices) a serem visitados, visando a minimização do custo total da viagem, incluindo o aluguel dos veículos e que a satisfação mínima de visitar as cidades, representada pelo parâmetro ω seja atendida.

Dessa forma, a definição do problema e uma formulação matemática são apresentadas na Seção 2. O modelo matemático foi implementado em um *solver* e os resultados são relatados na Seção 4. Este trabalho também apresenta um algoritmo memético proposto para o pCaRS na Seção 3. São analisadas 64 instâncias de problemas e os resultados são comparados com os obtidos com o *solver* na Secção 4. Finalmente, apresentamos algumas conclusões na Seção 5.

2. O Problema do Caixeiro Alugador com Coleta de Prêmios

Seja G = (V, A) um grafo completo, onde V é um conjunto com n nós (cidades) e A é

um conjunto de arcos (estradas entre as cidades). Um bônus SV_i, $i = 1,\ldots,n$, é atribuído a cada cidade $i \in V$. Neste problema, um conjunto K de veículos está disponível para aluguel. Custos operacionais específicos estão associados a cada carro, incluindo o consumo de combustível, pagamento de pedágio e custos de aluguel. Uma vez que as taxas de pedágio normalmente dependem do tipo de veículo e do comprimento do caminho percorrido, é possível assumir, sem perda de generalidade, que o custo operacional de cada carro para atravessar aresta $(i,j) \in A$ é uma função daquele carro.

Os custos operacionais associados ao carro k para cruzar a aresta $(i,j) \in A$ é denotado por C_{ij}^k. O Vértice 1 foi escolhido como o ponto de partida e final do trajeto a ser realizado, sendo definido como cidade base. Podem ser alugados k carros na cidade i e entregues na cidade j. Caso $i \neq j$, existe uma taxa a ser paga para retornar o carro k para a cidade i. Cada cidade pode ser visitada no máximo uma vez. A formulação matemática considera as seguintes variáveis binárias: f_{ij}^k com valor 1 quando o carro k trafega pela aresta (i,j) partindo de i até j e valor 0 caso contrário; d_{ij}^k com valor 1 quando o carro k é locado na cidade j e entregue na cidade i; H_i^k com valor 1 quando o carro k chega no nó i; S_i^k com valor 1 quando o carro k sai do nó i; a_i^k com valor 1 quando o carro k é alugado na cidade i; e_i^k com valor 1 quando o carro k é entregue na cidade i. A formulação também considera o parâmetro ω que é dado por $0.8ST$, $ST = \sum_{i \in V} SV_i$,que é o somatório da satisfação de visitar todas as cidades, e a variável inteira u_i que retorna a posição do vértice i no percurso.

$$\min \sum_{k \in K} \sum_{i,j \in V} C_{ij}^k f_{ij}^k + \sum_{k \in K} \sum_{i,j \in V} W_{ij}^k d_{ij}^k \quad (1)$$

$$\sum_{k \in K} \sum_{i \in V} f_{i1}^k = \sum_{k \in K} \sum_{j \in V} f_{1j}^k = 1 \quad (2)$$

$$\sum_{k \in K} (f_{ij}^k + f_{ji}^k) \leq 1 \qquad \forall i,j \in V,\ i \neq j \quad (3)$$

$$\sum_{k \in K} \sum_{j \in V, i \neq j} f_{ij}^k = \sum_{k \in K} \sum_{j \in V, i \neq j} f_{ji}^k \qquad \forall i \in V \quad (4)$$

$$H_i^k = \sum_{j \in V, i \neq j} f_{ji}^k \qquad \forall k \in K \ \ \forall i \in V,\ i > 1 \quad (5)$$

$$S_i^k = \sum_{j \in V, i \neq j} f_{ij}^k \qquad \forall k \in K \ \ \forall i \in V,\ i > 1 \quad (6)$$

$$e_i^k = H_i^k \sum_{x \in K, x \neq k} S_i^x \qquad \forall k \in K,\ i \in V,\ i > 1 \quad (7)$$

$$a_i^k = S_i^k \sum_{x \in K, x \neq k} H_i^x \qquad \forall k \in K,\ i \in V,\ i > 1 \quad (8)$$

$$d_{ij}^k = a_j^k e_i^k \qquad \forall k \in K,\ i,j \in V \quad (9)$$

$$\sum_{k \in K} a_1^k = 1 \quad (10)$$

$$\sum_{i \in N} a_i^k \leq 1 \qquad \forall k \in K \quad (11)$$

$$\sum_{i \in N} a_i^k = \sum_{i \in N} e_i^k \qquad \forall k \in K \quad (12)$$

$$\sum_{k \in K} \sum_{i \in V} H_i^k SV_i \geq 0.8ST \quad (13)$$

$$2 \leq u_i \leq n \qquad \forall i = 2,\ldots,n \quad (14)$$

$$u_i - u_j + 1 \leq (N-1)(1 - \sum_{k \in K} f_{ij}^k) \qquad \forall i,j = 2,\ldots,n \quad (15)$$

$$f_{ij}^k,\ d_{ij}^k,\ H_i^k,\ S_i^k,\ a_i^k,\ e_i^k \in \{0,1\} \quad (16)$$

$$u_i \in \aleph \quad (17)$$

A função objetivo é apresentada em (1). Ela inclui o custo de percorrer as rotas com diferentes carros e taxas referentes a devolução dos mesmos. A restrição (2) garante que a rota começa e termina na cidade base, ou seja, o vértice 1. As restrições (3) e (4) garantem que cada vértice é visitado, no máximo, uma vez e se um carro chega ao vértice *i*, um carro deve deixar esse vértice. A restrição (5) associa, no máximo, um carro chegando em cada nó *i* e a restrição (6) garante que, no máximo, um carro deixa cada nó *i*. As restrições (7), (8) e (9), estão relacionados ao aluguel e entrega do veículo. A restrição (10) garante que um carro é alugado no nó 1. A restrição (11) assegura que cada carro seja usado apenas uma vez, e a restrição (12) garante que cada carro locado seja entregue. A restrição (13) certifica que um nível mínimo de satisfação em visitar as cidades seja cumprido, previsto para *0.8ST*. As restrições (14) e (15) proíbem a formação de subrotas e foram adaptadas das restrições de Miller-Tucker-Zemlin (MTZ) para o PCV apresentadas em Miller *et al*. (1960) e também utilizadas em Vansteenwegen, *et al*. (2011). A restrição (16) declara algumas variáveis binárias e a restrição (17) diz que as variáveis u_i são inteiros positivos.

As restrições (7), (8) e (9) são quadráticas e suas variáveis são binárias. A linearização apresentada nas expressões (19) a (22) são conhecidas como linearização usual (Liberti, 2007) e foram propostas por Fortet (1960). A Equação (18) representa uma restrição não linear que é substituída pelo conjunto de equações (19)-(22).

$$z = x \times y \quad (18)$$

$$z \leq x \quad (19)$$

$$z \leq y \quad (20)$$

$$z \geq x + y - 1 \quad (21)$$

$$z, x, y \in [0,1] \quad (22)$$

Esta linearização foi aplicada para resolver o problema via GLPK *solver* de acordo com o proposto em AIMMS (2012).

3. Algoritmo Memético

Os algoritmos meméticos são uma hibridização dos algoritmos evolucionários com algum tipo de busca local, foram propostos por Moscato (1989) e são utilizados em uma vasta gama de aplicações. O algoritmo memético proposto para o pCaRS é apresentado na Figura 1 e detalhado a seguir.

```
1.  main(inst, tamPop,  #Cross)
2.  LeInstancia(inst)
3.  Pop[] ← geraPopInicial(tamPop)
4.  buscaMultiOperadores(Pop)
5.   enquanto(não satisfaz critério de parada) faça
6.      para i ← 1 até tamPop*#Cross faça
7.           pai,mãe  ← selecionaPais()
8.           filho1,filho2 ← Crossover(pai, mãe)
9.           buscaMultiOperadores (filho1, filho2)
10.          se filho1, filho2 > pai, mãe
11.                  Pop[pai] ← filho1, Pop[mãe] ← filho2
12.  fim
```

Figura 1. Algoritmo memético proposto para o pCaRS

Cromosomo

O cromossomo é representado por uma matriz de 2 dimensões, conforme ilustrado na Figura 2, onde a rota é representada numa dimensão (superior) e os carros são representados na outra (inferior). Na Figura 2 a matriz cinza representa o percurso e em branco representa os carros. Nesta figura dez cidades são visitadas, observe que as cidades 10 e 11 não são visitadas. Este cromossomo representa uma solução onde o carro 2 é contratado na cidade base e entregue na 4 cidade, onde o carro 3 é contratado e assim por diante.

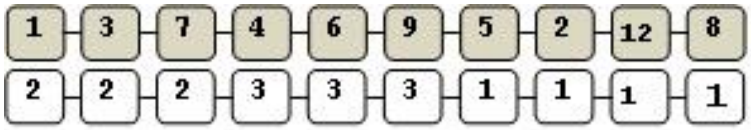

Figura 2. Representação do Cromossomo

População Inicial

A população inicial é gerada através do procedimento *geraPopInicial ()* que recebe o tamanho da população, *tamPop*, como parâmetro de entrada. Uma heurística gulosa adaptada do problema CaRS é usada para gerar os cromossomos da população inicial. Um carro k_1 é escolhido aleatoriamente para a primeira cidade. Em seguida, a cidade *i* onde o carro k_1 será entregue é também escolhida de forma aleatória. Um caminho é criado entre as cidades *1* e *i* através da heurística do vizinho mais próximo, considerando os custos de viagem com o carro k_1. A cidade *i* é a cidade inicial da próxima parte da rota. Em seguida, um novo carro k_2 e uma nova cidade *j* são escolhidos aleatoriamente. Claramente, a cidade *j* não pode ser uma cidade visitada anteriormente. Um caminho é construído entre as cidades *i* e *j* com as cidades não visitadas. Caso não existam mais carros disponíveis, utiliza-se o último carro sorteado. O procedimento continua sorteando outra cidade aleatoriamente e construindo o caminho até que o nível de satisfação seja alcançado, quando o ciclo é fechado considerando a ligação com a cidade de partida.

Busca Local

Os processos de busca local visam intensificar a procura por um ótimo local, enquanto a estrutura de diversificação fica por conta da definição de boas regiões de busca resultantes dos processos evolucionários (Krasnogor, 2004). Este tipo de algoritmo é bastante eficiente em problemas de otimização combinatória, do tipo NP-hard (Ong, 2010). O principal desafio entre os métodos heurísticos que usam a busca local é definir uma estratégia eficiente para cobrir o espaço de busca, explorando principalmente regiões promissoras (Vansteenwegen *et al*., 2011). No caso do problema pCaRS, existem três áreas onde a busca local pode atuar: a melhor rota, os melhores pontos para trocar de carro e o nível de satisfação. A fase de busca local do algoritmo proposto visa lidar com estas três áreas, a fim de melhorar os candidatos a solução. A busca local é implantada no procedimento *buscaMultiOperadores()* e é composta pelos seguintes métodos:

- ***removeCidade*:** Este método está relacionado ao nível de satisfação. Consiste em remover da solução as cidades com os menores índices de satisfação, enquanto a restrição referente a satisfação mínima é atendida.
- ***InverteSol*:** Este método inverte a ordem de visita das cidades na solução. Os mesmos carros estão associados com as mesmas cidades, mas os pontos de aluguel e entrega são trocados. Por exemplo, considere a rota (1, 2, 3, 4, 5) em cinco cidades com um carro sendo contratado na cidade 1 e sendo devolvido em 3 e o carro 2 alugado na cidade 3 e entregue na cidade 1. Após a aplicação do método *InverteSol*, a rota torna-se (1,5,4,3,2), com o carro 1 sendo contratado na cidade 1 e entregue na cidade 4 e o carro 2 sendo contratado na cidade 4 e sendo entregue na cidade 1.
- ***substituiCarro***: Este procedimento concentra-se em veículos que não estão ainda na solução a examinando a inserção de um novo carro, se possível. O novo carro substitui

outro na solução. Todos os carros que não fazem parte da solução de entrada são considerados. O carro que não está na solução é inserido em cada posição de forma iterativa. Por exemplo, suponha que uma instância onde os carros 1, 2, 3 e 4 podem ser alugados. Considere-se uma solução com cinco cidades representadas pelo tour (1,2,3,4,5) com carros 2, 3 e 4, sendo contratados (entregue), respectivamente, nas cidades 1,3,5 (3,5,1) . O vetor de atribuição de carros para as cidades é representado como (2,2,3,3,4). Carro 1 não é usada nesta solução. Em seguida, o procedimento *substituiCarro* examina as possibilidades de inserção do carro 1 na mesma. Primeiro o carro 1 é inserido na posição 1 produzindo a seqüência de carros (1,2,3,3,4), então é inserido o carro 1 nas posições 2, 3, 4 e 5 que produzem, respectivamente, (1,1,3,3,4), (1,1,1,3,4), (1,1,1,1,4), (1,1,1,1,1). No passo seguinte, o carro 1 substitui o segundo carro na seqüência original do carro, produzindo (2,1,3,3,4). A substituição continua a partir desse ponto, produzindo (2,1,1,3,4), (2,1,1,1,4) e (2,1,1,1,1). Em seguida, a terceira posição na seqüência original é definido como 1 e o processo continua até que todas as possibilidades sejam examinadas.

- ***substituiCidade***: Este procedimento atua sobre as cidades que ainda não estão na solução. Ele examina a substituição de cidades na solução por cidades que não estão presentes na solução. Todas as cidades da solução de entrada são consideradas para a inserção. Por exemplo, considere a rota [1 3 4 5 6]. A cidade 2 não está nesta rota. Em seguida, o procedimento examina as seguintes rotas: [1 2 4 5 6], [1 3 2 5 6], [1 3 4 2 6] e [1 3 4 5 2]. O vetor de carros associados com a rota de entrada não é alterado.
- ***insereCidade***: Este procedimento também foca em cidades que ainda não estão na solução de entrada. Insere uma nova cidade na turnê e não remove nenhuma cidade. O carro atribuído a nova cidade é o mesmo da cidade imediatamente anterior. A inserção de uma nova cidade é testada entre cada par de cidades da solução de entrada. Todas as cidades da solução de entrada, são considerados para a inserção. Por exemplo, considere novamente a turnê [1 3 4 5 6] e a inserção da cidade 2. Em seguida, o procedimento examina as seguintes rotas: [1 2 3 4 5 6], [1 3 2 4 5 6], [1 3 4 2 5 6], [1 3 4 5 2 6] e [1 3 4 5 6 2].
- ***2-opt*:** é um algoritmo de busca local simples proposto por Croes (1958) para o PCV. Um movimento 2-opt consiste em eliminar duas arestas e religar os dois caminhos, resultando em uma maneira diferente de obter uma nova rota. Entre todos os pares de arestas cuja troca *2-opt* atua, a que resultar no menor custo de rota é escolhido. Este procedimento é então iterado até que o par de arestas mais econômico seja encontrado. A rota resultante é chamada de 2-ótima. Neste caso, a seqüência de carros associados permanece a mesma

Estas buscas locais são aplicadas em seqüência no procedimento *buscaMultiOperadores()*, conforme ilustrado na Figura 3. Se a solução de entrada representada por um indivíduo é melhorada durante a fase de busca local, então o novo indivíduo substitui o antigo.

```
1. buscaMultiOperadores(filho1,filho2)
2.       filho1,filho2 ← removeCidade(filho1, filho2)
3.       filho1, filho2 ← inverteSol (filho1, filho2)
4.       filho1, filho2 ← insereCidade(filho1, filho2)
5.       filho1, filho2 ← substituiCidade(filho1, filho2)
6.       filho1, filho2 ← substituiCarro(filho1, filho2)
7.       filho1, filho2 ← 2Opt(filho1, filho2)
8. fim
```

Figura 3. Procedimento *buscaMultiOperadores*

Recombinação

Dois pais são escolhidos aleatoriamente a partir da população atual. O *crossover* de um ponto é utilizado. Uma vez que o número de genes em cromossomos pode variar, um ponto aleatório é escolhido na gama de índices do menor cromossomo. A recombinação dos dois pais, A e B, gera dois descendentes, C e D, tal como ilustrado na Figura 4. O ponto de cruzamento é

ilustrado com uma linha tracejada nos cromossomos A e B

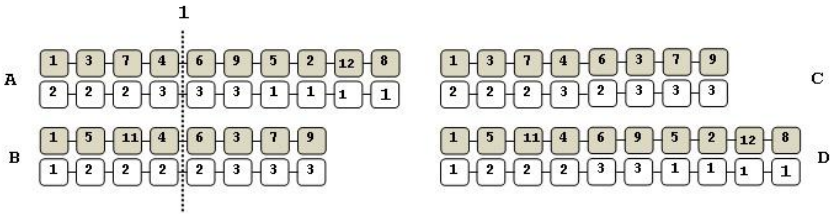

Figura 4. Operador de Recombinação

Pode ser necessário restaurar a viabilidade de soluções geradas durante a recombinação. A inviabilidade pode ocorrer sobre rotas, atribuição de carros ou de satisfação total. Por exemplo, imediatamente depois de ter sido gerado o cromossomo *C* o mesmo tem uma solução inviável devido ao percurso e atribuição de veículos. Inviabilidade em relação às cidades da rota ocorre devido às cidades 3 e 7 aparecem duas vezes cada uma na solução. Inviabilidade em relação a atribuição de carros ocorre devido aos carros 2 e 3 serem alugados por duas vezes. O procedimento de recuperação usado neste trabalho é o mesmo apresentado em Goldbarg *et al*. (2012). Para restaurar a viabilidade sobre as cidades da rota, cada cidade que aparece pela segunda vez no cromossomo *C*, é substituída por um asterisco. Por exemplo, a seqüência de cidades [1 3 7 6 4 3 7 9] é substituída por [1 3 7 4 6 * * 9]. Cada asterisco é substituído por uma cidade diferente, escolhida aleatoriamente entre aquelas que não fazem parte da solução. A atribuição de carros do cromossomo C, [2 2 2 3 2 3 3 3], também não é viável, pois o problema considerado neste trabalho exige que cada carro seja alugado no máximo uma vez. O processo de restauração substitui repetições do carro por asteriscos. Em seguida, cada asterisco é substituído pelo carro que aparece na primeira posição anterior. Assim, a atribuição de carro do cromossomo C é substituído por [2 2 2 3 ****] e cada um asterisco é substituído pelo carro 3.

Se depois de restaurado nos procedimentos anteriormente mencionados, um cromossomo não atender ao requisito mínimo de satisfação, um procedimento para restaurar a satisfação é executado. Em primeiro lugar, as cidades com baixo grau de satisfação são substituídas por outras com melhores níveis de satisfação. Se o nível mínimo de satisfação ainda não for atingido, então cidades são adicionadas aleatoriamente na solução até atingir a satisfação do mínimo requerido. O carro atribuído a cada nova cidade é o mesmo atribuído a uma cidade imediatamente anterior a ele.

4. Experimentos Computacionais

Como o pCaRS é um novo problema, uma biblioteca de instâncias, com o nome pCaRSLIB, foi criada. Eles foram adaptados a partir das instâncias CaRSLIB (Godbarg *et al*. 2012) e estão disponíveis no endereço: http://www.dimap.ufrn.br/lae/en/projects/CaRS.php. Essas instâncias possuem as seguintes características: todos os carros podem ser alugados em qualquer cidade, todos os carros podem ser entregues em qualquer cidade, cada carro pode ser alugado somente uma vez, a taxa paga para ter um carro de volta para sua cidade de origem não está associado com a topologia da instância; simétrica, os custos para ir de *i* para *j* e de *j* para *i* são iguais, o grafo é completo. As instâncias são divididas em duas classes: euclidiana e não-euclidiana. Três grupos de instâncias foram criadas para cada classe. A diferença entre cada grupo é a forma com que os pesos das arestas foram gerados. Primeiro, um conjunto principal de pesos de aresta é estabelecido para cada grupo. Os pesos das arestas primárias foram tomados a partir de mapas reais para o primeiro grupo, gerados uniformemente no intervalo [10,500] para o segundo grupo e com base em exemplos TSPLIB constantes em Reinelt (1991) para o terceiro grupo. Então, o peso de cada aresta correspondente ao carro c, $1 \leq c \leq |C|$, foi escolhida aleatoriamente, com probabilidade uniforme, no intervalo $[1.1w_e, 2.0w_e]$ onde w_e representa o peso primário da aresta e. O nível de satisfação atribuído a cada cidade foi gerado uniformemente no intervalo [0,100].

A formulação matemática apresentada na Secção 2 foi implementada no software GLPK, versão 4.45.2. A memória do sistema foi limitada a 14 Gb RAM. O tempo de

processamento foi limitado a 80000 segundos. O processamento do GLPK foi finalizado caso o problema fosse solucionado, ou caso atingisse os limites estabelecidos de memória e tempo.

Os parâmetros do algoritmo memético foram estabelecidos em testes preliminares e são: *tamPop* = 150 e *#Cross* = 0.6. O algoritmo finalizou o processamento após 80 gerações sem melhora.

As Figuras 5-7 ilustram o comportamento dos parâmetros *tamPop* em três diferentes instâncias usadas nos experimentos preliminares, evidenciando que o mínimo encontrado pelo algoritmo ficou estabilizado a partir dos parâmetros utilizados.

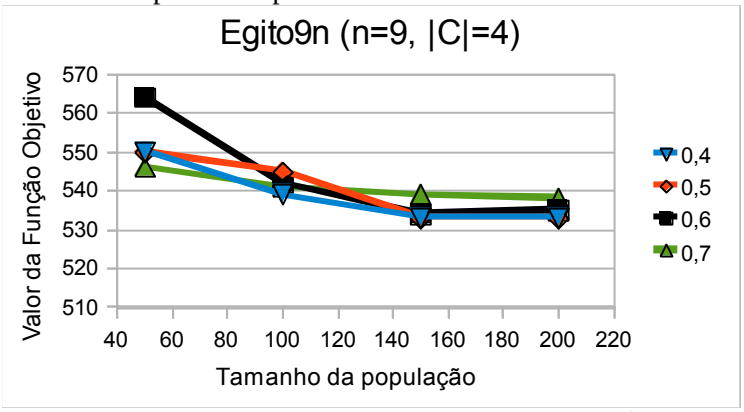

Figura 5. Tamanho da população versus valor da melhor solução na instância Egito9n

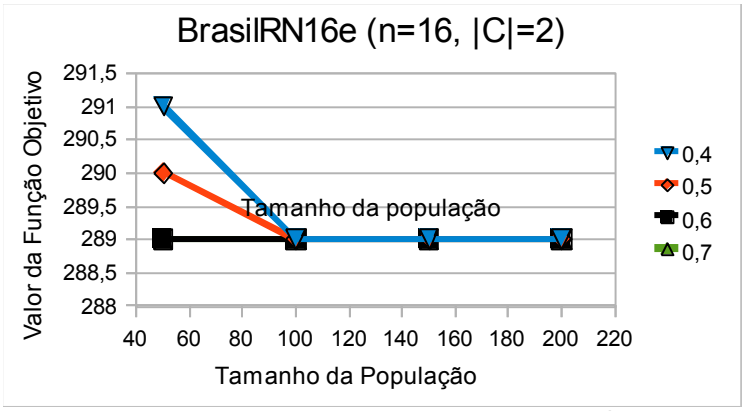

Figura 6. Tamanho da população versus valor da melhor solução na instância BrasilRN16e

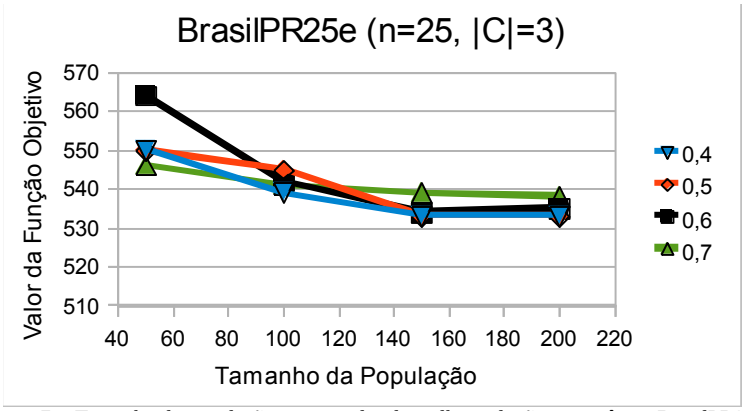

Figura 7. Tamanho da população versus valor da melhor solução na instância BrasilPR25e

Os algoritmos foram executados em um PC Intel Core i5, 16G de RAM, rodando Ubuntu 12.04 64bits. O algoritmo memético foi implementado em linguagem C++ e executado trinta vezes para cada instância.

As Tabelas I e II apresentam os resultados dos experimentos computacionais. As colunas *Nome*, *nCid* e *nCar* estão relacionadas com as características de cada instância e são,

respectivamente, a identificação do problema, o número de cidades e o número de carros disponíveis. As colunas *Min* e *T (s)*, estão relacionadas com a solução obtida pelo *solver* GLPK, e mostram o valor da melhor solução viável inteira e o tempo de processamento, em segundos. As colunas *Min, Av, F* e *T (s)*, dizem respeito ao algoritmo memético e mostram, respectivamente, o valor mínimo da solução encontrada, a média das soluções mínimas encontradas nas trinta execuções independentes, o número de vezes que a melhor solução foi encontrada e o tempo médio de processamento em segundos. A coluna *Gap*, para cada algoritmo, apresenta o desvio percentual do valor apresentado em *min* a partir de um limite global para a solução exata calculada pelo GLPK. Sendo o *epsilon machine* representado por $\in$, *best_mip* e *best_bound* representam, a melhor solução inteira para o problema e a melhor solução relaxada, respectivamente, o *GAP* é dado por:

$$GAP = \frac{|best\_mip - best\_bound|}{|best\_mip + \in|} \tag{23}$$

A Tabela I mostra que o *solver* GLPK encontra a solução ótima para quase todas as instâncias euclidianas, com exceção da *China17e*, *Russia17e*, *BrasilAM26e*, *BrasilMG30e* e *att48eA* onde o *solver* parou devido a limitação de memória. Para este casos, o desvio percentual produzido pelo GLPK foi de 6,40, 10,47, 28,60 e 20,50 38,12, respectivamente. Todas as soluções ótimas encontradas pelo GLPK também foram encontrados pelo algoritmo memético, exceto para a instância *Brasil16e* onde a solução produzida pelo último apresentou desvio de 2,91 por cento a partir do primeiro. Em doze dessas instâncias, o GLPK gastou menos tempo que o algoritmo memético utilizou em média. Esse fato ocorreu devido as seis fases de busca local do último algoritmo, implementado em *buscaMultiOperadores*, requererem tempos significativos de processamento. Um fato que deve ser observado é que em instâncias maiores do que estas doze, o esforço computacional gasto nas fases de busca local beneficia a pesquisa feita pelo algoritmo memético. As tabelas mostram que a medida que o tamanho das instâncias aumenta, o desempenho do algoritmo memético melhora em relação aos resultados obtidos pelo *solver*. Esta tendência também é observada com a análise dos dados apresentados na Tabela II. Nas outras vinte e uma instâncias euclidianas o algoritmo memético gastou, em média, um tempo menor de processamento do que o GLPK. Em média, o GLPK gastou 12.089,39 segundo para produzir soluções para as instâncias euclidianas enquanto o algoritmo memético utilizou 81,84 segundos. A freqüência média da melhor solução encontrada pelo algoritmo memético é 19,22 para as instâncias euclidianas.

A Tabela II mostra que o GLPK não resolveu dezenove instâncias não-euclidianas para as quais o algoritmo parou devido a limitação de tempo ou de memória. Os desvios percentuais variam, entre 2 e 37,60. Em quinze dentre estes dezenove casos, o algoritmo memético encontrou soluções melhores do que os produzidos pelo GLPK. O primeiro algoritmo também gasta, em média, um menor tempo de processamento para obter essas soluções. O algoritmo memético gastou, em média, 13 até 1232 segundos para processar as instâncias não-euclidianas. Os resultados obtidos através do GLPK levou entre 1 e 80000 segundos. A freqüência média das melhores soluções encontradas pelo algoritmo memético é 12,75 para as instâncias não-euclidianas.

5. Conclusões

Este artigo apresentou o Problema do Caixeiro Alugador com Coleta de Prêmios (pCaRS), uma nova variante do CaRS apresentado em Goldbarg *et al*. (2012). Um modelo matemático foi apresentado e submetido a resolução através do *solver* GLPK com limitações de tempo e memória. Um algoritmo memético que usa seis procedimentos de busca local foi proposto. Uma análise experimental foi realizada para investigar o potencial do algoritmo memético proposto. As soluções e os tempos de processamento produzido por estes foram

comparados com os resultados da aplicação do modelo matemático no *solver* GLPK. Um conjunto com trinta e duas instâncias euclidianas e trinta e duas instâncias não-euclidianas foi utilizado nos experimentos. Através do *solver* GLPK foram encontradas soluções exatas para quarenta instâncias e o algoritmo heurístico proposto estabeleceu os primeiros limites máximos para as outras vinte e quatro instâncias. Os resultados das experiências computacionais mostraram que, para instâncias Euclidianas o algoritmo memético proposto encontra a solução ótima para grande maioria dos casos solucionados pelo GLPK. O algoritmo memético também encontrou soluções melhores do que as produzidas pelo GLPK quando o mesmo teve o processamento encerrado devido ao tempo ou limitações de memória. De uma forma geral, o desempenho em tempo computacional do algoritmo memético é significativamente superior ao método exato no conjunto de instâncias examinadas. Em vários casos é também qualitativamente superior, particularmente no conjunto de instâncias não-euclidianas. A experiência também mostrou que quanto maiores as instâncias, há uma tendência de o algoritmo memético melhorar o seu comportamento quantitativo e qualitativo em comparação com o *solver* GLPK.

Como o problema aqui proposto é novo, diversas inovações podem ser implementadas para pesquisas futuras, tais como: (a) examinar o uso de novas vizinhanças de busca para o problema (b) desenvolver outras metaheurísticas para o pCaRS.

Referências

AIMMS Integer Programming Tricks, avaliable at http://www.aimms.com. Acess 12/07/2012, 15h.

Balas, E. The prize collecting traveling salesman problem. *Networks*, vol. 19, 1989, pp. 621-636.

Croes, G. A. A method for solving traveling salesman problems. *Operations Research*, vol. 6 , 1958, pp. 791-812.

Fortet, R. Applications de l'algebre de boole en recherche operationelle. *Revue Française de Recherche Operationelle*, 4ed , 1960, pp. 17-26.

GLPK (GNU Linear Programming Kit) package. Version 4.45.2, avaliable at http://www.gnu.org/software/glpk/

Goldbarg, M. C. Asconavieta, P. e Goldbarg, E. F. G. Memetic algorithm for the traveling car renter problem: an experimental investigation. *Memetic Computing*, vol. 4, 2012, pp. 89-108.

Gutin, G. e Punnen, A.P. Traveling salesman problem and its variations, *Kluwer Academic Publishers*, Dordrecht, 2002.

Krasnogor, N. e Gustafson, S. A Study on the use of "self-generation" in memetic algorithms. *Natural Computing*, vol. 3, 2004, pp. 53–76.

Liberti, L. Compact linearization for binary quadratic problems. *4OR*, vol. 5, no. 3, 2007, pp. 231-245.

Miller, C. Tucker, A. e Zemlin, R. Integer programming formulations and travelling salesman problems. *Journal of the ACM*, vol. 7, 1960, pp. 326–329.

Moscato, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards Memetic Algorithm. *Caltech Concurrent Computation Program*. California Institute of Technology, USA, 1989.

Ong,Y. S. Lim, M. H. e Chen, X. S. Research frontier: memetic computation - past, present & future, *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, 2010, pp. 24 -36.

P. Vansteenwegen, W. Souffriau, e D. Van Oudheusden, The orienteering problem: a survey. *European Journal of Operational Research* vol. 209, 2011, pp. 1-10.

Reinelt, G. TSPLIB - A traveling salesman problem library, *ORSA Journal on Computing*, vol. 3, 1991, pp. 376-384.

Souffriau, W. Vansteenwegen,P. Vertommen,J. Berghe, G. V. e Van Oudheusden, D. A personalised tourist trip design algorithm for móbile tourist guides. *Applied Artificial Intelligence*, vol. 22, no. 10, 2008, pp. 964–985.

Vansteenwegen, P. Souffriau, W. Berghe, G. V. e Van Oudheusden, D. The city trip planner: an expert system for tourists, *Expert Systems with Applications*, vol. 38, 2011, pp. 6540–6546.

Vansteenwegenand, P. Van Oudheusden, D. The mobile tourist guide: An or opportunity. *OR Insights*, vol. 20, no. 3, 2007, pp. 21–27.

TABELA I. RESULTADOS PARA AS INSTÂNCIAS EUCLIDEANAS

Instâncias			**Exato (GLPK)**			**Memético**				
Nome	***nCid***	***nCar***	***Min***	***T(s)***	***Gap***	***Min***	***Av***	***F***	***T(s)***	***GAP***
Mauritania10e	10	2	422	5	0,00%	422	422	30	16	0,00%
Colombia11e	11	2	326	1	0,00%	326	326	30	14	0,00%
Angola12e	12	2	490	8	0,00%	490	490	30	15	0,00%
Peru13e	13	2	556	46	0,00%	556	556	30	16	0,00%
Libia14e	14	2	521	45	0,00%	521	523	23	34	0,00%
BrasilRJ14e	14	2	230	560	0,00%	230	232	14	43	0,00%
Congo15e	15	2	513	28	0,00%	513	516	18	37	0,00%
Argentina16e	16	2	719	2276	0,00%	719	719	30	45	0,00%
EUA17e	17	2	602	71,5	0,00%	602	604	23	69	0,00%
Bolivia10e	10	3	384	3	0,00%	384	384	30	20	0,00%
AfricaSul11e	11	3	402	11	0,00%	402	402	30	21	0,00%
Niger12e	12	3	564	53	0,00%	564	576	7	34	0,00%
Mongolia13e	13	3	543	607	0,00%	543	544	5	30	0,00%
Indonesia14e	14	3	504	22	0,00%	504	508	21	40	0,00%
Argelia15e	15	3	487	351	0,00%	487	489	22	34	0,00%
India16e	16	3	705	36	0,00%	705	710	20	66	0,00%
China17e	17	3	735	57332	6,40%	728	732	9	78	5,50%
Etiopia10e	10	4	283	2	0,00%	283	283	30	17	0,00%
Mali11e	11	4	428	10	0,00%	428	428	30	25	0,00%
Chade12e	12	4	655	861	0,00%	655	664	12	44	0,00%
Ira13e	13	4	532	49	0,00%	532	537	21	60	0,00%
Mexico14e	14	4	492	144	0,00%	492	492	30	30	0,00%
Sudao15e	15	4	422	46	0,00%	422	423	28	31	0,00%
Australia16e	16	4	682	453	0,00%	682	732	3	83	0,00%
Canada17e	17	4	783	1599	0,00%	783	785	22	93	0,00%
Arabia14e	14	5	482	50	0,00%	482	482	30	42	0,00%
Cazaquistao15e	15	5	574	1473	0,00%	574	588	5	60	0,00%
Brasil16e	16	5	619	718	0,00%	637	639	28	51	2,83%
Russia17e	17	5	760	80000	10,40%	750	784	1	96	9,21%
BrasilAM26e	25	3	371	80000	20,50%	342	348	1	168	13,76%
BrasilMG30e	26	3	419	80000	28,60%	370	379	1	257	19,14%
att48eA	30	4	25234	80000	38,12%	20444	20954	1	950	23,62%

TABELA II. RESULTADOS PARA AS INSTÂNCIAS NÃO EUCLIDEANAS

Instancias			Exato (GLPK)			Memético				
Nome	*nCid*	*nCar*	*Min*	*T(s)*	*Gap*	*Min*	*Av*	*F*	*T(s)*	*GAP*
Mauritania10n	10	2	306	1	0,00%	306	306	30	13	0,00%
Colombia11n	11	2	461	224	0,00%	461	462	24	17	0,00%
Angola12n	12	2	409	50	0,00%	409	409	30	21	0,00%
Peru13n	13	2	502	3634	0,00%	502	506	8	42	0,00%
Libia14n	14	2	504	77307	0,00%	504	504	30	31	0,00%
BrasilRJ14n	14	2	101	58346	0,00%	101	102	9	39	0,00%
Congo15n	15	2	573	5747	0,00%	573	584	14	35	0,00%
Argentina16n	16	2	647	62007	22,70%	643	648	4	54	22,22%
EUA17n	17	2	579	80000	8,10%	579	600	2	68	8,10%
Bolivia10n	10	3	448	54	0,00%	448	455	12	19	0,00%
AfricaSul11n	11	3	537	7755	0,00%	537	537	30	23	0,00%
Niger12n	12	3	607	4069	0,00%	607	613	9	32	0,00%
Mongolia13n	13	3	551	80000	2,00%	551	552	16	42	2,00%
Indonesia14n	14	3	522	16188	0,00%	522	524	15	42	0,00%
Argelia15n	15	3	619	48859	17,30%	616	629	2	60	16,90%
India16n	16	3	734	73024	24,90%	723	734	5	80	23,76%
China17n	17	3	651	62162	20,10%	645	659	1	59	19,36%
Etiopia10n	10	4	403	153	0,00%	403	403	30	21	0,00%
Mali11n	11	4	494	262	0,00%	494	494	30	31	0,00%
Chade12n	12	4	654	69111	10,40%	649	651	17	39	9,71%
Ira13n	13	4	697	54936	20,90%	693	701	12	40	20,44%
Mexico14n	14	4	620	61358	18,70%	610	611	28	40	17,37%
Sudao15n	15	4	793	80000	29,50%	769	777	3	62	27,30%
Australia16n	16	4	551	80000	16,90%	525	533	11	83	12,78%
Canada17n	17	4	827	77542	23,80%	824	877	3	108	23,52%
Arabia14n	14	5	701	37832	27,80%	692	699	4	65	26,86%
Cazaquistao15n	15	5	843	67421	28,80%	833	867	3	94	27,95%
Brasil16n	16	5	769	32052	32,50%	742	745	18	48	30,04%
Russia17n	17	5	820	52468	37,60%	782	788	3	123	34,57%
BrasilAM26n	25	3	107	80000	17,80%	107	109	1	160	17,80%
BrasilMG30n	26	3	179	80000	30,20%	162	167	3	360	22,88%
att48nA	30	4	443	80000	17,50%	443	598	1	1232	17,50%