HEURÍSTICAS E METAHEURÍSTICAS PARA O PROBLEMA DO CAIXEIRO VIAJANTE BRANCO E PRETO

André Cordeiro Macedo Maciel, Carlos Alberto Martinhon, Luis Satoru Ochi.

Instituto de Computação - Universidade Federal Fluminense
Rua: Passo da Pátria nº156, Bloco E, 3º Andar - São Domingos.
Niterói - RJ - Brasil. CEP: 24210 -240

e-mail: {acordeiro, mart, satoru}@ic.uff.br

Resumo: O Problema do Caixeiro Viajante Branco e Preto - *PCVBP* é definido sobre um grafo *G* (orientado ou não), de maneira que o conjunto de vértices associado está particionado em vértices brancos e pretos. O objetivo é encontrar um circuito hamiltoniano de custo mínimo sujeito a restrições adicionais de *cardinalidade* e *comprimento*. Estas restrições estão relacionadas respectivamente, ao número de vértices brancos (cardinalidade) e a distância total percorrida (comprimento) entre dois vértices pretos consecutivos em uma solução viável. Neste artigo, introduzimos uma formulação matemática para o caso orientado e propomos novas buscas locais e uma heurística de construção *randomizada* que são utilizadas junto as metaheurísticas *GRASP*, *VNS* e *VND*. Resultados computacionais mostram a viabilidade dos métodos propostos.

Palavras Chaves: Problema do Caixeiro Viajante, Heurísticas, Metaheurísticas.

Abstract: The Black and White Traveling Salesman Problem - *BWTSP* is defined in a graph *G* (oriented or not), in a such way that the associated vertex set is partitioned into black and white vertices. The objective is to find a shortest hamiltonian tour subject to *cardinality* and *length* constraints. These constraints are related to the number of white vertices (cardinality) and the total distance (length) between two consecutive black vertices in a feasible solution. In this article, we introduce a mathematical formulation for the oriented case, and propose a new randomized construction heuristic and local search procedures that are used in the metaheuristics *GRASP*, *VNS* and *VND*. Computational results show the viability of the proposed methods.

Keywords: The Traveling Salesman Problem, Heuristics, Metaheuristics.

1 Introdução

O Problema do Caixeiro Viajante Branco e Preto - *PCVBP* é definido em um grafo *G* (orientado ou não), de maneira que o conjunto de vértices associado está dividido em *brancos* e *pretos*. O objetivo é encontrar o menor circuito hamiltoniano partindo arbitrariamente de um vértice branco ou preto, e sujeito às restrições adicionais de *cardinalidade* e *comprimento*. Estas restrições estão relacionadas ao número de vértices brancos (representado por *Q*) e a distância total percorrida (representada por *L*) entre dois vértices pretos consecutivos em uma rota viável do problema. O *PCVBP* é NP-Dificil já que, para *Q = L =* ele se reduz ao Problema do Caixeiro Viajante - *PCV* reconhecidamente NP-Difícil [4].

Em Bourgeois *et. al.*[2], os autores introduziram o *PCVBP* e propuseram três heurísticas distintas de construção (*C1,C2 e C3*) baseadas na heurística *GENIUS* [5], além de uma heurística de viabilização (*F*) e uma busca local.

Neste trabalho, propomos uma heurística de construção *randomizada* e quatro buscas locais que são utilizadas na implementação das metaheurísticas *GRASP* [3], *VNS* [6] e *VND* [6]. Adicionalmente, apresentamos uma formulação matemática para o *PCVBP* assimétrico descrevendo-o como um problema de programação linear inteira. Alguns experimentos computacionais foram realizados com instâncias geradas de forma aleatória, os resultados obtidos nos experimentos mostraram a validade de se usar metaheurísticas na determinação de limites superiores de boa qualidade para o problema, além de demonstrar de forma empírica, que versões híbridas tendem a apresentar melhor desempenho computacional.

Este artigo está dividido da seguinte forma: na seção 2, apresentamos uma definição formal para o *PCVBP* e uma formulação matemática para o caso assimétrico. Na seção 3 são apresentadas regras de redução, que serão utilizadas em uma fase de pré-processamento no intuito de eliminar as arestas que, reconhecidamente não fazem parte de uma solução viável do problema. Nas seções 4 e 5 são apresentadas uma rápida revisão da literatura para o *PCVBP* e das metaheurísticas *GRASP*, *VNS* e

VND respectivamente. As heurísticas propostas são descritas na seção 6. As implementações e resultados computacionais são ilustrados na seção 7. Finalmente, a seção 8 apresenta as conclusões finais e algumas propostas para trabalhos futuros.

2 O PCVBP

O *PCVBP* foi proposto por Bourgeois *et al*. [2] e pode ser definido por um grafo direcionado $G = (V,A)$ (ou não direcionado $G = (V,E)$) onde $V = \{v_0, v_1, v_2, ... ,v_n\}$ representa o conjunto de vértices e $A = \{(v_i ,v_j): v_i ,v_j \in V, i \neq j\}$ um conjunto de arcos (ou $E = \{(v_i ,v_j): v_i, v_j \in V, i<j\}$ o conjunto de arestas) de G. O conjunto de vértices é particionado nos subconjuntos B e P, onde B são os vértices brancos, P são os vértices pretos e $|P| \geq 2$. O *PCVBP* consiste em determinar um circuito (ciclo) hamiltoniano de custo mínimo em G obedecendo as restrições do *PCV* e sujeito às seguintes restrições adicionais:

1. **Cardinalidade:** o número de vértices brancos entre dois vértices pretos consecutivos não pode ser superior a um número inteiro positivo Q.
2. **Comprimento:** a distância entre dois vértices pretos consecutivos não deve exceder um número positivo L.

Como já foi dito anteriormente o *PCVBP* é um problema NP- Difícil. Além desta dificuldade, garantir a viabilidade de uma solução do *PCVBP* pode ser uma tarefa complicada em algumas situações. Por exemplo, quando uma das condições abaixo for satisfeita, pode-se garantir que o *PCVBP* possui o conjunto de soluções viáveis vazio:

- $|B| > Q.|P|$: Se o número de vértices brancos for maior que o número de vértices pretos multiplicado por Q, isto significa que a restrição de cardinalidade não pode ser satisfeita.
- $c_{ij_1(i)} + c_{ij_2(i)} > L$, para todo $v_i \in B$, onde $v_{j_1(i)}$ e $v_{j_2(i)}$ são os dois vértices pretos mais próximos de v_i. Isto significa que não existe um caminho que passe por v_i que seja capaz de satisfazer a restrição de comprimento (L).

Note que esta última condição de inviabilidade só será aplicada se a desigualdade triangular for satisfeita.

2.1 Formulação Matemática

Nesta seção é apresentada uma proposta de uma formulação matemática para o *PCVBP* assimétrico descrevendo-o como um Problema de Programação Linear Inteira. Para se ter uma melhor compreensão do modelo, as seguintes considerações devem ser feitas: dado um arco $(i,j) \in A$, dizemos que (i,j) *parte* (*outcoming edge*) de $i \in A$ e *chega (incoming edge*) a $j \in A$. Os vértices são numerados de 0 a n, sendo que a origem é indicada pelo vértice 0 e pode representar um vértice branco ou preto indistintamente. Representa-se por f um conjunto de "*cores*" associadas a cada caminho entre dois vértices pretos, dessa forma, todas as arestas pertencentes a este caminho deverão ser "*coloridas*" com uma mesma cor. Assume-se ainda que z_{ij} e x^k_{ij} são variáveis binárias. Assim, se $z_{ij} = 1$, a aresta (v_i,v_j) de custo c_{ij} pertence a solução viável, e $z_{ij}=0$ caso contrário. Se $x^k_{ij} = 1$, a aresta (v_i,v_j) de custo c_{ij} e cor k pertencerá a solução viável, e $x^k_{ij} = 0$ caso contrário. Assim, o objetivo será encontrar o menor circuito hamiltoniano e que satisfaça as restrições de cardinalidade Q e comprimento L.

Na função objetivo adiante, representada por (1) desejamos minimizar o somatório do custo das arestas presentes em uma solução viável. O conjunto de restrições (2) garante que apenas uma única aresta de uma única cor parte do vértice $i \in V$. Analogamente, o conjunto de restrições (3) garante que apenas uma única aresta de uma única cor chegue ao vértice $i \in V$. O conjunto de restrições (4) garante que para todo vértice branco ($j \hat{I} B$) a cor da aresta que chega em j é igual à cor da aresta que parte de j. Como no modelo proposto, cada cor $k \in f$ está associada a uma cadeia entre dois vértices pretos, o somatório de todas as arestas de cor k indica o seu comprimento, logo para atender a restrição de comprimento, este somatório deve ser menor ou igual a L (conjunto de restrições 5). Note que uma cadeia de cor k composta por p vértices, terá $p - 1$ arestas e $p - 2$ vértices brancos associados. Logo, para satisfazer a restrição de cardinalidade, o número de vértices brancos em uma cadeia de cor k deve ser menor ou igual a Q (conjunto de restrições 6). O conjunto de restrições (7) garante que uma cadeia

de cor *k* possua origem em um vértice preto. Analogamente, o conjunto de restrições (8) garante que uma cadeia de cor *k* possui um único destino preto. Os conjuntos de restrições (9), (10) e (11) são restrições de fluxo e garantem a formação de um ciclo *hamiltoniano* contendo a origem. Note que o conjunto de restrições (9) garante a não existência de sub-rotas não passando pela origem. Entretanto, podemos ter várias rotas distintas satisfazendo (9). Assim, a restrição (10) assegura que, se $y_{ij} > 0$ para algum $i,j \hat{I} A$, então $z_{ij} = 1$. Logo, como as variáveis *x* e *z* estão relacionadas através de (11) teremos uma única sub-rota contendo a origem. Desta forma, uma solução viável irá percorrer todos os vértices uma única vez atendendo respectivamente as restrições de cardinalidade e comprimento.

$$Min \sum_{(i,j)\in A} c_{ij}.z_{ij} \qquad (1)$$

sujeito a:

$$\begin{cases}
\sum_{k\in f} \sum_{j\in V/\{i\}} x_{ij}^k = 1, & \forall i \in V \quad (2)\\
\sum_{k\in f} \sum_{j\in V/\{i\}} x_{ji}^k = 1, & \forall i \in V \quad (3)\\
\sum_{i\in V/\{j\}} x_{ij}^k = \sum_{l\in V/\{j\}} x_{jl}^k, & \forall j \in B, \forall k \in f \quad (4)\\
\sum_{(i,j)\in A} c_{ij}.x_{ij}^k \le L, & \forall k \in f \quad (5)\\
\sum_{(i,j)\in A} x_{ij}^k \le Q+1, & \forall k \in f \quad (6)\\
\sum_{i\in P \text{ e } (i,j)\in A} x_{ij}^k = 1, & \forall k \in f \quad (7)\\
\sum_{l\in P \text{ e } (j,l)\in A} x_{jl}^k = 1, & \forall k \in f \quad (8)\\
\sum_{i=0}^{n} y_{ij} - \sum_{i=0}^{n} y_{ji} = 1, & j = 1,2...,n \quad (9)\\
y_{ij} \le n.z_{ij}, & \forall (i,j) \in A \quad (10)\\
\sum_{k\in f} x_{ij}^k = z_{ij}, & \forall (i,j) \in A \quad (11)\\
x_{ij}^k, z_{ij} \in \{0,1\}, y_{ij} \ge 0, & \forall (i,j) \in A \quad (12)
\end{cases}$$

3 Regras de Redução para o PCVBP

Nas regras de redução busca-se uma diminuição no espaço de busca eliminando implicitamente soluções que sabidamente não farão parte do conjunto de soluções ótimas. Estas regras, normalmente, são baseadas em propriedades matemáticas simples e buscam a eliminação de arestas que inviabilizam a construção de uma solução.

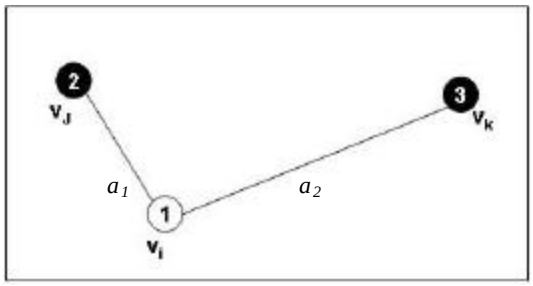

Figura 1 – Regra de redução 1

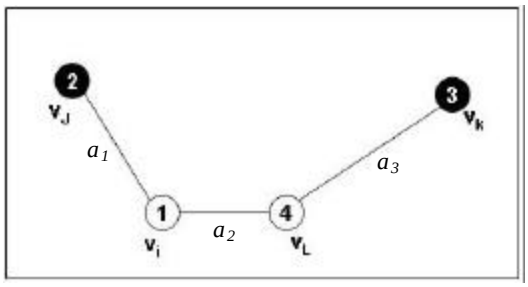

Figura 2 – Regra de redução 2

A seguir são apresentadas algumas regras de redução para o *PCVBP*:

- **Regra 1:** Sejam v_j, $v_k \in P$ e suponha ser v_j o vértice preto mais próximo de $v_i \in B$. Alem disso considere $a_1 = (v_i,v_j)$ e $a_2 = (v_i,v_k)$, tais que se $c_{a1} + c_{a2} >$ L então se elimina $a_2 \in A$ (*Figura 1*).

- **Regra 2:** Seja v_j o vértice preto mais próximo de $v_i \in B$, e suponha ser v_k o vértice preto mais próximo de $v_l \in B$. Além disso, considere $a_1 = (v_i, v_j)$, $a_2 = (v_i, v_l)$ e $a_3 = (v_l, v_k)$. Se $c_{a1} + c_{a2} + c_{a3} > L$, então se elimina $a_2 \in$ A (*Figura 2*).
- **Regra 3:** Se $Q = 1$ e o total de vértices pretos for igual ao total de vértices brancos eliminar todas as arestas entre vértices pretos.
- **Regra 4:** Eliminar todas as arestas que possuem comprimento maior que L.

4 Heurísticas da Literatura

Nesta seção são apresentadas, resumidamente, as heurísticas propostas em Bourgeois *et al.* [2] para o Problema do Caixeiro Viajante Branco e Preto. Os autores propuseram cinco heurísticas distintas sendo que, as três primeiras são heurísticas de construção e visam determinar uma solução inicial (viável ou inviável) para o *PCVBP*. A quarta é uma heurística de viabilização. A quinta e última, busca melhorar as soluções viáveis geradas pelas heurísticas anteriores através da exploração de uma vizinhança.

4.1 Heurísticas de Construção da literatura

As heurísticas de construção propostas em [2] são baseadas essencialmente na heurística *GENIUS* [5]. O *GENIUS* foi proposto inicialmente para o *PCV* e é composto por duas fases. A primeira denominada *GENI* constitui a fase de construção, a segunda chamada de *US* é a fase de refinamento da solução obtida na primeira fase. Em linhas mais gerais o *GENIUS* pode ser descrito da seguinte forma:

- Inicialmente a fase *GENI* considera uma rota composta por três vértices quaisquer, e a cada iteração um novo vértice v é inserido à solução corrente. A inserção do vértice v é feita levando-se em consideração os p vizinhos mais próximos de v na solução corrente. A principal diferença entre o *GENI* e os demais métodos de inserção, é que no *GENI* um vértice pode ser inserido entre dois vértices não adjacentes pertencentes à solução parcial corrente.
- Após a inserção de todos os vértices, a fase *US* é iniciada. A fase *US* consiste em remover cada vértice pertencente à solução corrente, reinserindo-o novamente através procedimento *GENI*.

Para mais detalhe da heurística *GENIUS* consulte [5]. A seguir são apresentadas de forma sucinta as heurísticas de construção *C1*, *C2* e *C3* propostas em [2].

4.1.1 Heurística de Construção *C1*

A heurística *C1* despreza as restrições de comprimento e cardinalidade e trata o *PCVBP* como se fosse o *PCV*. Para construir uma boa solução para o *PCV* é utilizada a heurística *GENIUS*. Ao final da heurística *C1* é produzida uma solução que pode ser ou não viável ao *PCVBP*, já que pode violar tanto a restrição de comprimento quanto a restrição de cardinalidade.

4.1.2 Heurística de Construção *C2*

A heurística *C2* leva em consideração tanto a restrição de cardinalidade quanto a restrição de comprimento. Para gerar uma solução viável, *C2* inicia aplicando sobre os vértices brancos o *GENIUS*, obtendo desta forma um ciclo somente com os vértices brancos.

Em seguida é aplicado um procedimento de corte onde um vértice branco v_r do ciclo parcial corrente é escolhido aleatoriamente, v_r é inserido em um conjunto R (conjunto dos vértices que iniciam a formação de cadeias brancas), e seguindo tanto o sentindo horário quanto anti-horário, cadeias de vértices brancos são construídas a partir do ciclo de vértices brancos obedecendo-se às seguintes regras:

1. O número de vértices brancos pertences à cadeia não deve exceder Q.
2. O comprimento da cadeia não deve exceder o valor da restrição $L \quad l$, onde l é duas vezes a distância média de um vértice branco a um vértice preto.

Se o número de cadeias produzidas em cada direção (sentido horário e anti-horário) for superior ao número de vértices pretos $|P|$, a heurística é finalizada (a heurística neste caso não consegue

construir uma solução inicial), caso contrário, uma matriz D com distâncias d_{ij} é construída levando-se em consideração todos os vértices e as seguintes regras:

$$d_{ij} = \begin{cases} c_{ij} & Se\ v_i\ ou\ v_j \in P \\ \infty & Se\ v_i, v_j \in B\ e\ pertenc\ em\ a\ cadei\ as\ \ difer\ entes \\ -\infty & Se\ \ v_i, v_j \in B\ e\ pertence\ m\ a\ mesma\ \ cadeia \end{cases}$$

A heurística *GENIUS* então é aplicado sobre todos os vértices (brancos e pretos) levando-se em consideração a nova matriz de distâncias D para a construção de uma solução para o *PCV*. Enquanto as soluções produzidas pela heurística *GENIUS* usando a matriz D são sempre viáveis quanto à restrição de cardinalidade, a restrição de comprimento pode ser desrespeitada. Isto acontece quando uma cadeia (trecho da solução iniciada por um vértice preto e terminada com um vértice preto) excede L. Se a cadeia inviável contém no mínimo dois vértices brancos, algumas medidas podem ser tomadas na tentativa de torná-la viável, pois se escolhermos um novo v_r (vértices branco) pertencente a uma cadeia inviável como ponto partida para que a heurística *C2* seja reinicializada, soluções viáveis poderão ser construídas. A escolha do novo v_r é efetuada da seguinte maneira: dada uma aresta branca $(v_s,v_t) \in A$ (onde $v_s,v_t \in B$) e pertencente à cadeia inviável, fazemos $v_r \leftarrow v_s$ se $v_s \notin R$, fazemos $v_r \leftarrow v_t$ se $v_s \in R$ e $v_t \notin R$. Se v_s, $v_t \in R$ a heurística é finalizada, caso contrário $R \leftarrow R \cup \{v_r\}$ e a heurística *C2* é reinicializada a partir do procedimento de corte.

4.1.3 Heurística de Construção *C3*

A heurística *C3* também leva em consideração as restrições de cardinalidade e comprimento. Assim como as outras heurísticas de construção, *C3* também utiliza a heurística *GENIUS*.

Inicialmente a heurística *C3* é aplicada sobre os vértices brancos através do procedimento *GENIUS*. Sobre o ciclo obtido, um vértice $v_s \in B$ é escolhido aleatoriamente. O próximo passo é associar a v_s o vértice preto mais próximo (v_r) e inseri-lo onde cause o menor incremento do custo da solução (à direita ou à esquerda de v_s). A partir de v_r o ciclo é percorrido (tanto no sentido horário quanto no sentido anti-horário) enquanto as restrições de cardinalidade e comprimento possam ser satisfeitas. Quando Q e L não podem ser mais satisfeitos, o vértice preto mais próximo do último vértice branco percorrido deve ser inserido de forma que a cadeia preta formada respeite as restrições de cardinalidade e comprimento. Esta heurística se diferencia de *C2*; entre outras coisas por não trabalhar com a matriz D e não garantir que as restrições de comprimento e/ou cardinalidade sejam satisfeitas.

A heurística *C3* finaliza quando é impossível iniciar uma nova cadeia. Se o último vértice branco é atingido e vértices pretos não foram inseridos à solução, estes são inseridos ao final da cadeia de forma a causar o menor incremento possível à solução.

4. 2 Heurística de viabilização F

De acordo com as heurísticas de construção vistas anteriormente pode-se ter ao final, uma solução inviável produzida por *C1* ou duas soluções inviáveis produzidas por *C2* e *C3*. Note que *C2* e *C3* geram uma solução associada a cada orientação.

A heurística *F* tem como objetivo eliminar a inviabilidade gerada determinando uma solução viável caso seja possível. Se a restrição de comprimento é violada isto implica que alguma(s) cadeia(s) preta(s) possui (em) comprimento maior do que L. Esta inviabilidade é chamada de *inviabilidade preta*. A inviabilidade preta pode ser eliminada pela inserção de um vértice preto entre dois pontos da cadeia que viola L. Quando a restrição de cardinalidade é violada é chamada de *inviabilidade branca.* Esta pode ser resolvida pela remoção de vértices brancos do interior da cadeia e reinserindo-os em cadeias que possuem no mínimo $Q \quad 1$ vértices brancos.

A heurística *F* sempre tenta eliminar inicialmente a inviabilidade preta e posteriormente a inviabilidade branca. Desta forma, é importante ressaltar que na tentativa de se eliminar a inviabilidade preta, esta pode causar inviabilidade branca, mas a recíproca não deve ocorrer. Outro aspecto importante na eliminação da inviabilidade preta é que a inserção de um vértice preto em uma cadeia que viole L implica na remoção de um vértice preto pertencente à outra cadeia. Esta remoção

não deve causar uma nova inviabilidade preta e para evitar a formação de ciclos um vértice só pode ser removido uma única vez.

4. 3 Heurística de Refinamento

A heurística de refinamento descrita em [2] é a Heurística 2-Optimal (*2-OPT*), proposta inicialmente em [7]. O *2-OPT* é aplicado somente sobre as soluções viáveis e busca melhorar uma solução preservando sua viabilidade através da exploração de sua vizinhança. O *2-OPT* explora a vizinhança da solução corrente efetuando todas as possíveis trocas de arestas entre vértices não adjacentes.

5 Metaheurísticas

A seguir serão descritas as metaheurísticas *GRASP*, *VNS* e *VND* que posteriormente serão utilizadas na resolução do *PCVBP*.

Para os algoritmos apresentados nas próximas seções considere: *s* uma solução viável ou inviável, *s** a melhor solução encontrada até o momento, s' como uma solução temporária, *custo*(*s*) uma função que retorna o custo da solução *s* e *viável*(*s*) é uma função que retorna *verdadeiro* se a solução *s* satisfaz as restrições de cardinalidade e comprimento, e retorna *falso*, caso contrário.

5.1 *GRASP - Greed Randomized Adaptive Search Procedure*

Proposto por Feo e Resende [3] em 1995 o *GRASP* é um processo iterativo multi-partida em que cada iteração consiste de duas etapas: uma etapa de construção e outra de busca local.

A fase de construção é também iterativa, adaptativa, randômica e gulosa. Ela é iterativa uma vez que a solução é construída elemento a elemento. É adaptativa, pois na escolha de um elemento são levadas em consideração as escolhas dos elementos anteriores.

A cada iteração desta fase uma percentagem dos melhores elementos candidatos a serem incluídos na solução corrente são colocados em uma *Lista de Candidatos Restritos - LCR*. Estes elementos que constituem a *LCR* são escolhidos segundo uma função adaptativa e gulosa, que estima o beneficio da escolha de cada um dos elementos. O nível de *aleatoriedade* e o quanto é gulosa a fase de construção do *GRASP* é controlado pelo parâmetro $\alpha \in [0,1]$. Um valor $\alpha = 0$ faz gerar soluções puramente gulosas, ou seja, somente o melhor candidato é inserido na *LCR*; enquanto que $\alpha = 1$, considera todos os candidatos na *LCR*. Após a construção da *LCR* o elemento que irá fazer parte da solução é escolhido aleatoriamente, desta forma permitimos que a cada iteração *GRASP* soluções diferentes sejam construídas.

Como na fase construção a solução resultante não necessariamente representa um ótimo local, é recomendável aplicar métodos de busca local que explore uma vizinhança na tentativa de melhorar a solução inicial. A metaheurística *GRASP* pode ser ilustrada pelo *Algoritmo 1*

Algoritmo 1 – GRASP(num_iterações, α)

```
1.  custo(s*) ← ∞;
2.  k ← 0;
3.  Enquanto k ≤ num_interações Faça
4.    s ← Construtivo(α);
5.    s ← Busca_Local(s);
6.    Se (custo(s) < custo(s*)) Então
7.      s* ← s;
8.    Fim Se
9.    k ← k + 1;
10. Fim Enquanto
11. Retorne s*;
```

5.2 *VNS - Variable Neighborhood Search*

Método proposto em 1997 por Hansen & Mladenovic [6], o *VNS* consiste em explorar a partir de uma solução inicial (*s*), o espaço de soluções vizinhas através de trocas sistemáticas de estruturas de vizinhança (representadas por $N_k(s)$, onde $k = 1, 2, ..., N$. Sendo que N indica o número total de estruturas de vizinhanças).

Em sua proposta original o *VNS* é composto por duas fases, a fase de construção e a fase de busca local, porem se diferencia das demais metaheurísticas pelo fato de não seguir uma trajetória, mas sim explorar gradativamente diferentes mecanismos de busca local através de trocas sistemáticas de vizinhanças. Uma característica importante do *VNS* é que este método aceita somente movimentos de melhora da solução e sempre que isto ocorre, ele retorna à primeira estrutura de vizinhança. A idéia por detrás deste método é explorar uma vizinhança ao máximo enquanto resultados satisfatórios forem sendo obtidos. Uma ilustração do *VNS* pode ser visto no *Algoritmo 2*.

Algoritmo 2 – VNS(s, num_iterações)

```
1.  s*←s;
2.  custo(s*) ← custo(s);
3.  i ← 0;
4.  Enquanto ( i ≤ num_iterações) Faça
5.    s' ← Gerar randomicamente s' ∈ N^K(s);
6.    k ← 1;
7.    Enquanto (k ≤ N) Faça
8.      s'' ← Explorar N^k(s');
9.      Se (custo(s'') ≤ custo(s)) Então
10.       s ← s'';  k ← 1;
11.     Senão
12.       k ← k + 1;
13.     Fim Se
14.   Fim Enquanto
15.   Se (custo(s) ≤ custo(s*)) Então
16.       s* ← s;
17.   Fim Se
18.   i ← i + 1;
19. Fim Enquanto
20. Retorne s*;
```

5.3 *VND - Variable Neighborhood Descent*

O *VND* é uma variante do *VNS,* ou mais especificamente um caso particular, também proposto por Hansen & Mladenovic em 1997. Este método pode ser descrito conforme o *Algoritmo 3*.

Algoritmo 3 – VND(s)

```
1.   s*← s;  k ← 1;
2.   Enquanto (k ≤ N) Faça
3.       s' ← melhor solução ∈ N^K (s*);
4.       Se (custo(s') ≤ custo(s*)) Então
5.          s* ← s';  k ← 1;
6.       Senão
7.          k ← k + 1;
8.       Fim Se
9.   Fim Enquanto
10. Retorne s*;
```

6 Heurísticas Propostas

Nesta seção são apresentadas uma heurística de construção e quatro estruturas de vizinhança para o *PCVBP*. A heurística de construção proposta, as heurísticas propostas em [2] e as buscas locais são combinadas e implementadas utilizando - se as metaheurísticas *GRASP*, *VNS* e *VND* na tentativa de se obter bons resultados aproximados e de boa qualidade para o *PCVBP*. Inicialmente descrevemos o método proposto para a construção de soluções iniciais e as buscas locais. Numa etapa seguinte, descrevemos a utilização das metaheurísticas *GRASP*, *VNS* e *VND*.

6.1 Heurística de construção e Buscas Locais

6.1.1 Construtivo: Heurística *Randomizada C3_R*

Este método incorpora os conceitos propostos na heurística *C3* apresentada na seção 4.1.3, sendo modificada de forma a torná-la mais adaptativa e randômica. A cada iteração deste método um elemento é inserido a solução corrente pela escolha aleatória de um elemento pertencente a *LCR*.

Assuma que c_{min} representa o menor custo de se inserir um vértice v à solução corrente entre todos os vértices candidatos, c_{max} o maior custo e *alpha* o parâmetro que controla o quanto é gulosa a fase construção. A *LCR* é construída segundo uma função de custo $f(v)$ que retorna o custo c de se inserir v a solução corrente. Se c pertence ao intervalo $c_{min} \leq c \leq c_{min} + \alpha (c_{max}-c_{min})$, v é inserido a *LCR*.

6.1.2 Busca Local 1 (*BL1*)

Esta busca local visa "*reorganizar*" cada cadeia entre dois vértices pretos de forma a otimizar seu custo sem perder a viabilidade. A *BL1* funciona da seguinte maneira, para toda cadeia cuja cardinalidade é maior ou igual a dois, faça: Remova cada vértice branco adicionando-o na cadeia corrente, no local onde cause o maior decremento do custo da solução sem torná-la inviável.

6.1.3 Busca Local 2 (*BL2*)

A Busca Local 2 é uma adaptação da fase *US* pertencente à heurística *GENIUS* proposto em [5] e descrita na seção 4.1. A Busca Local *2* consiste em percorrer uma solução viável removendo cada vértice $v_i \in V$ e adicionando-o usando o procedimento *GENI* [5], porém esta adição do vértice v_i à solução corrente, deve obedecer as restrições de cardinalidade e comprimento.

6.1.4 Busca Local 3 (*BL3*)

A Busca Local 3 efetua todas as permutações possíveis entre dois vértices distintos (brancos ou pretos) mesmo sendo de cadeias diferentes checando sempre sua viabilidade.

6.1.5 Busca Local 4 (*BL4*)

A *BL4* é uma variação da busca local proposta em [2] descrito na seção 4.3. Estas se diferenciam pelo fato de que para cada movimento de troca de arestas que implica na melhoria do custo da solução (sem perda de viabilidade), *BL4* utiliza esta solução como uma nova semente. Para cada nova semente uma lista de possíveis trocas de arestas é gerada e testada. O critério de parada ocorre quando todas as possíveis trocas de arestas são esgotadas e não há melhoria da semente atual.

6.2 Metaheurísticas Propostas

A literatura apresenta um número razoável de metaheurísticas para a solução de problemas de otimização combinatória como é o caso do *PCVBP*. Dentre elas, destacamos os métodos : *GRASP*, *VNS*, e *Busca Tabu*. Em particular alguns trabalhos da literatura mostram que versões híbridas de *GRASP* são muito competitivas para problemas combinatórios [8].

Nesta seção propomos as seguintes combinações entre o procedimento construtivo e as buscas locais discutidas acima. A *Tabela 1* ilustra cada versão proposta, onde a coluna Busca Local representa para cada versão as estruturas de vizinhança utilizadas no *VNS* e *VND*.

O procedimento *VND'* (*Algoritmo 4*) se diferencia da metaheurística *VND* pelo fato de que é acrescido a este uma perturbação da solução quando as buscas locais não são capazes de melhorar a solução corrente. A idéia contida neste procedimento de perturbação (*Algoritmo 5*) é fazer com que através de movimentos de trocas aleatórias de posições de vértices distintos pertencentes à solução (mesmo sendo movimentos de piora), seja gerada uma nova solução viável em que as buscas locais possam explorar uma nova vizinhança, permitindo assim sair de ótimos locais ainda distantes de um ótimo global.

No *Algoritmo 4* considere NT uma constante que indica o número de trocas a serem efetuadas no procedimento pertuba (*Algoritmo 5*). Para o *Algoritmo 5* considere *L* como sendo uma lista de vértices pertencente à solução corrente.

id	Metaheurística	Construtivo	Busca Local
GRASP_1	GRASP + VND	C3_R + F	BL1 + BL2 + BL3 + *2-OPT* + BL4
GRASP_2	GRASP + VND' + VND	C3_R + F	BL1 + BL2 + BL3 + *2-OPT* + BL4
VNS_1	VNS	C1 + F	BL1 + BL2 + BL3 + *2-OPT* + BL4
VNS_2	VNS	C2 + F	BL1 + BL2 + BL3 + *2-OPT* + BL4
VNS_3	VNS	C3 + F	BL1 + BL2 + BL3 + *2-OPT* + BL4

Tabela 1: Metaheurísticas utilizadas

Nos procedimentos *GRASP_1* e *GRASP_2* utilizamos o procedimento *GRASP* como descrito na seção 5.1, o procedimento de construção *C3_R* (seção 6.1.1) e as buscas locais *VND* e *VND'* respectivamente.

Algoritmo 4 – VND'(s)

```
1.  s*← s; k ← 1;
2.  Enquanto (k ≤ 6) Faça
3.      1 : s' ← BL1(s*);
4.      2 : s' ← BL2(s*);
5.      3 : s' ← BL3(s*);
6.      4 : s' ← 2-OPT(s*);
7.      5 : s' ← BL4(s*);
8.      6 : s' ← Pertuba(s*, NT);   s' ← VND(s');
9.    Se (custo(s') < custo(s*)) Então
10.     s* ← s';  k ← 1;
11.   Senão
12.     k ← k + 1;
13.   Fim Se
14. Fim Enquanto
15. Retorne s*;
```

Algoritmo 5 – Pertuba(s, NT)

```
1.  L ← {v0, v1 ... vn / ∀ v ∈ V}; i ← 0;
2.  Enquanto (i ≤ NT) Faça
3.    troca_efetuada ← false;
4.    Enquanto (troca_efetuada = false) Faça
5.      Escolher dois vértices a e b pertencentes a L;
6.      s ← Trocar de posição a e b;
7.      Se (viavel (s) = true ) Então
8.        troca_efetuada ← true;  L ← L – {a,b};   i ←  i + 1;
9.      Senão
10.       s ← Trocar de posição a e b;
11.     Fim Se
12.   Fim Enquanto
13. Fim Enquanto
14. Retorne s;
```

Para as propostas *VNS_1*, *VNS_2* e *VNS_3* usamos uma versão básica do *VNS*, descrito na seção 5.2.

7 Resultados Computacionais

De nosso conhecimento só existe o trabalho proposto em [2] para o *PCVBP*. Neste trabalho os autores não disponibilizam as instâncias por eles utilizadas. Desta forma no nosso artigo tivemos que gerar instâncias de diferentes tipos e dimensões para a avaliação dos algoritmos aqui propostos.

As instâncias utilizadas em nosso trabalho foram geradas como sugeridos em [2]: escolhe-se um quadrante no plano cartesiano, sorteia-se aleatoriamente o número de vértices desejados dentro deste quadrante. Depois destes criados, uma porcentagem dos vértices é classificada como vértices pretos. A escolha dos vértices a serem classificados como pretos se dá aleatoriamente e obedece as seguintes

porcentagens 10%, 20% ou 30% do total de vértices. Os valores de *Q* e *L* são propostos da seguinte maneira:

- **Valores de Q:** $Q = |B| / |P|$, $Q = (|B| / |P|) + 5$, $Q = (|B| / |P|) + 10$;
- **Valores de L:** $L = z\,\tilde{a} / |P|$, onde *z* é o custo encontrado pela execução do algoritmo GENIUS para a instância levando em consideração valores iniciais de $Q = L =$ e $\tilde{a}$ assume valores que variam de [1,75 a 4].

Para melhor identificarmos as instâncias, seus nomes (coluna 2 da *Tabela 2*) obedecem a seguinte estrutura, *[quadrante]-total de vértices_total de vértices pretos;* a coluna 1 apresenta a identificação das instâncias (ID); os valores sugeridos para y, z, *Q* e *L* (colunas 3, 4, 5 e 6) e a porcentagem das arestas eliminadas pelas regras de redução (coluna 7).

ID	Nome da Instância	ã	z	Q	L	% das arestas eliminadas
01	[0-100]-50_5	1,75	578	9	203	0
02	[0-100]-50_5	2	578	14	232	0
03	[0-100]-50_5	2,25	578	19	261	0
04	[20-80]-100_20	2.5	484	9	61	16,03
05	[20-80]-100_20	2.5	484	14	61	16,03
06	[40-60]-200_60	4	217	3	15	28,4
07	[40-60]-200_60	4	217	8	15	28,4
08	[0-400]-5_2	-	-	2*	380*	20
09	[0-400]-7_3	-	-	2*	400*	0
10	[0-400]-10_4	-	-	2*	500*	4,5
11	[0-400]-19_5	-	-	4*	480*	11,69

Tabela 2 – Descrição das Instâncias geradas. (os valores de Q e L sugeridos não obedecem as propriedades citadas anteriormente)*

Para cada algoritmo proposto na *Tabela 1* formam realizadas dez execuções de cada instância descrita na *Tabela 2* O critério de parada foi um número máximo de iterações igual a 100. A formulação matemática foi utilizada para gerar através do software *GLPK* soluções exatas para o *PCVBP*. Infelizmente como já era esperado, o uso de métodos exatos só foi possível para instâncias de dimensões muito pequenas (instâncias 8, 9,10 e 11 da *Tabela 2*). Para instâncias de dimensões um pouco maiores (de até 30 vértices) o *GLPK* não consegui obter uma solução exata, sendo abortado pelo sistema operacional devido o grande consumo de memória.

Os resultados obtidos através dos testes são expressos através das *Tabelas 3, 4, 5* e *6*. Nestas tabelas, a coluna 1 (ID) identifica as instâncias; a solução exata (se conhecida) é ilustrada na coluna 2. As colunas 3 e 6 identificam o valor de *a* utilizado na metaheurística *GRASP*. Para cada um dos 5 algoritmos propostos são mostrados a melhor solução (MS) e solução média (SM) obtida em 10 execuções. Nas colunas MS, o valor da célula $X_{(y)}$ indica que o valor X foi encontrado y vezes em 10 execuções. Para as colunas SM, o valor y indica o número de execuções onde uma solução viável foi encontrada e X indica a média em y execuções. Isso porque existem execuções onde nenhuma solução válida é encontrada. Para tabelas 3 e 4, os valores em negrito representam ou a solução ótima (se esta for conhecida) ou a melhor solução encontrada considerando todos os algoritmos aqui analisados.

		Nossas Propostas											
ID	**Exato**	***GRASP_1***			***GRASP_2***			***VNS_1***		***VNS 2***		***VNS_3***	
		a	**MS**	**SM**	**a**	**MS**	**SM**	**MS**	**SM**	**MS**	**SM**	**MS**	**SM**
01	-	10%	**639$_{(10)}$**	**639$_{(10)}$**	10%	**639$_{(6)}$**	640,5$_{(10)}$	692$_{(1)}$	735$_{(5)}$	-	-	642$_{(3)}$	649,5$_{(10)}$
02	-	5%	**609$_{(10)}$**	**609$_{(10)}$**	5%	**609$_{(10)}$**	**609$_{(10)}$**	612$_{(10)}$	612$_{(10)}$	**609$_{(10)}$**	**609$_{(10)}$**	**609$_{(10)}$**	**609$_{(10)}$**
03	-	5%	**582$_{(10)}$**	**582$_{(10)}$**	5%	**582$_{(10)}$**	**582$_{(10)}$**	**582$_{(10)}$**	**582$_{(10)}$**	**582$_{(10)}$**	**582$_{(10)}$**	**582$_{(10)}$**	**582$_{(10)}$**
04	-	5%	**487$_{(1)}$**	495,7$_{(10)}$	5%	488$_{(1)}$	**491,3$_{(10)}$**	**487$_{(1)}$**	495,2$_{(9)}$	-	-	491$_{(2)}$	494$_{(10)}$
05	-	5%	**483$_{(1)}$**	488,9$_{(10)}$	5%	484$_{(2)}$	**488,6$_{(10)}$**	491$_{(2)}$	494,5$_{(8)}$	-	-	486$_{(1)}$	492,5$_{(10)}$
06	-	5%	**255$_{(2)}$**	261,6$_{(10)}$	5%	257$_{(1)}$	262,8$_{(9)}$	256$_{(1)}$	**259,2$_{(10)}$**	-	-	258$_{(1)}$	262,2$_{(8)}$
07	-	5%	223$_{(1)}$	224,8$_{(4)}$	5%	221$_{(1)}$	234,1$_{(9)}$	221$_{(1)}$	**222,5$_{(10)}$**	-	-	222$_{(1)}$	224,6$_{(8)}$
08	467	10%	**467$_{(10)}$**	**467$_{(10)}$**	10%	**467$_{(10)}$**	**467$_{(10)}$**	**467$_{(10)}$**	**467$_{(10)}$**	**467$_{(10)}$**	**467$_{(10)}$**	**467$_{(10)}$**	**467$_{(10)}$**
09	768	10%	**768$_{(10)}$**	**768$_{(10)}$**	10%	**768$_{(10)}$**	**768$_{(10)}$**	**768$_{(10)}$**	**768$_{(10)}$**			**768$_{(10)}$**	**768$_{(10)}$**
10	1080	20%	**1080$_{(5)}$**	1083$_{(10)}$	10%	**1080$_{(10)}$**	**1080$_{(10)}$**	**1080$_{(10)}$**	**1080$_{(10)}$**	**1080$_{(10)}$**	**1080$_{(10)}$**	**1080$_{(10)}$**	**1080$_{(10)}$**
11	1474 *	20%	**1455$_{(10)}$**	**1455$_{(10)}$**	10%	**1455$_{(10)}$**	**1455$_{(10)}$**	**1455$_{(10)}$**	**1455$_{(10)}$**	-	-	**1455$_{(10)}$**	**1455$_{(10)}$**

Tabela 3 – Resultados obtidos pelos algoritmos propostos

Os resultados da tabela 3 mostram que em relação a melhor solução obtida (maior número de vitórias), os melhores resultados foram obtidos respectivamente por: *GRASP_1* (10 vitórias em 11 instâncias); *GRASP_2* (7 em 11); *VNS_1* e *VNS_3* (6 em 11); *VNS_2* (4 em 11). Em relação as melhores médias obtidas em 10 execuções para cada instância, tivemos a seguinte classificação do melhor para o pior: *GRASP_2* e *VNS_1* (7 em 11); *GRASP_1* e *VNS_1* (6 em 11); e novamente em último lugar o *VNS_2* com (4 em 11). Ainda na tabela 3, observamos que a versão *VNS_2*, não conseguiu obter uma solução viável em 7 das 11 instâncias analisadas. Comparando os resultados do exato com os das heurísticas propostas verificamos que as heurísticas *GRASP_1*, *GRASP_2*, *VNS_1* e *VNS_3* conseguem sempre atingir a solução ótima quando este é conhecido, e na instância 11 o exato foi abordado pelo sistema operacional. Neste caso a melhor solução obtida até então pelo *GLPK* é ilustrado na célula associada. Note que esta solução foi superada pelas heurísticas propostas *GRASP_1*, *GRASP_2*, *VNS_1*, *VNS_3* e pela heurística da literatura *C1-F-2OPT*.

		Algoritmos Propostos em [2]					
ID	**Exato**	***C1 – F – 2OPT***		***C2 – F – 2OPT***		***C3– F – 2OPT***	
		MS	**SM**	**MS**	**SM**	**MS**	**SM**
01	-	-	-	-	-	$690_{(1)}$	$793,5_{(15)}$
02	-	$623_{(4)}$	$683,6_{(10)}$	$656_{(1)}$	$796,9_{(16)}$	$754_{(1)}$	$837,7_{(17)}$
03	-	$599_{(2)}$	$619,9_{(10)}$	$702_{(1)}$	$838,8_{(2)}$	$687_{(1)}$	$746,3_{(2)}$
04	-	$707_{(1)}$	$707_{(1)}$	-	-	$784_{(1)}$	$784_{(1)}$
05	-	-	-	-	-	$663_{(1)}$	$708,5_{(2)}$
06	-	$442_{(1)}$	$443,5_{(2)}$	-	-	-	-
07	-	$260_{(1)}$	$260_{(1)}$	-	-	$329_{(1)}$	$329_{(1)}$
08	467	**467**$_{(2)}$	$530,6_{(5)}$	**467**$_{(6)}$	$527,57_{(14)}$	**467**$_{(6)}$	$515,18_{(11)}$
09	768	**768** $_{(10)}$	**768** $_{(10)}$	-	-	**768** $_{(6)}$	$783,36_{(11)}$
10	1080	$1167_{(3)}$	$1188,7_{(10)}$	$1138_{(1)}$	$1269,64_{(14)}$	$1086_{(1)}$	$1257,9_{(16)}$
11	1474 *	$1456_{(1)}$	$1795,8_{(6)}$	-	-	$1604_{(1)}$	$1883_{(6)}$

Tabela 4 – Resultados computacionais para as heurísticas propostas por [2]

A tabela 4 ilustra os resultados obtidos pelos algoritmos propostos em [2]. Observamos que o desempenho dos mesmos fica muito abaixo dos algoritmos propostos neste trabalho. De fato pela tabela 4, pode-se notar que o melhor desempenho em termos de número de vitórias foi obtido pela versão C1-F-2OPT e C3-F-2OPT com 2 vitórias.

	Nossas Propostas					**Algoritmos Propostos em [2]**			**Exato**
ID	***GRASP_1***	***GRASP_2***	***VNS_1***	***VNS_2***	***VNS_3***	***C1-F-2OPT***	***C2-F-2OPT***	***C3-F-2OPT***	
01	134,3	122,9	31,16	21,83	78,82	0.025	0,025	0,023	-
02	98,10	142,8	121,4	79,7	87,04	0,31	0,24	0,23	-
03	73,72	122,1	69,83	81,9	79,17	0,29	0,23	0,22	-
04	208,8	300,0	278,65	76,86	196,2	0,31	0,16	0,35	-
05	242,7	260,9	255,9	84,16	188,7	1,95	0,61	1,3	-
06	948,8	1139,5	5121,0	195,9	811,1	2,12	0,0007	2,25	-
07	280,5	422,0	1964,4	235,0	421,8	10,14	3,54	3,27	-
08	0,055	0,06	0,03	0,03	0,01	0,003	0,003	0,002	0,0
09	0,31	0,56	0,57	0,026	0,17	0,006	0,001	0,002	2,0
10	1,86	2,54	2,83	1,16	1,75	0,08	0,001	0,001	1223,0
11	3,46	4,9	4,41	0,59	2,87	0,021	0,006	0,005	*

*Tabela 5 – Tempos de execução em segundos. Os valores das colunas 2 a 9, representam tempos médios em 10 execuções e na coluna 10 o tempo exigido pelo método exato (GLPK). * significa que a execução foi abortada.*

Na tabela 5 é descrito o tempo médio exigidos pelos algoritmos propostos e os da literatura além dos tempos exigidos pelo método exato quando este foi utilizado. Como era de se esperar os tempos das metaheurísticas *GRASP* e *VNS* ficam muito acima das heurísticas propostas em [2] pelo fato destes últimos não serem iterativos, gerando assim um número muito reduzido de soluções. Em relação nas heurísticas propostas observamos que os tempos são similares na maioria das instâncias analisadas e que nos testes realizados estes tempos são relativamente pequenos (menos que 2 horas). Comparando os tempos do exato com os das heurísticas nota-se uma diferença brutal a favor das heurísticas e este diferença tende a ser cada vez maior a medida que as instâncias crescem.

8 Conclusões

Este trabalho apresenta propostas para resolver de forma exata e aproximada uma variante do Problema do Caixeiro Viajante - *PCV* aqui denominada Problema do Caixeiro Branco e Preto - *PCVBP*. Embora esta variante seja aplicável em problemas de roteamento e escalonamento tais como escalonamento de aeronaves e tripulações aéreas, configurações de redes de telecomunicações [1] entre outros; identificamos na literatura apenas um único artigo sobre este problema [2]. Como contribuições são propostos: uma formulação matemática do *PCVBP* (caso assimétrico) descrevendo-o como um problema de programação linear inteira, heurísticas do tipo *GRASP*, *VND* e *VNS*. Os testes computacionais efetuados mostram o potencial destes algoritmos, que nas instâncias pequenas, conseguem sempre atingir a solução ótima para os poucos casos onde esta é conhecida. Nas instâncias de maior porte, é verificado que versões híbridas conjugando *GRASP* com busca local *VND* se mostraram mais promissoras.

Como trabalhos futuros sugerimos elaboração de um modelo de Programação Linear Inteira para o *PCVBP* simétrico, visando desta forma, uma redução do número de variáveis associadas. Isto irá possibilitar a resolução de instâncias maiores para o caso simétrico. Outra possibilidade é a utilização das metaheurísticas *GRASP* com re-conexão de caminhos e *GRASP* com filtro.

Agradecimentos

Agradecemos aos revisores pelas valiosas sugestões e comentários.

Referencias Bibliográfica:

- [1]Bastos, L.; Ochi, L. Satoru e Macambira, E. M. (2005) *"A relative neighbourhood GRASP for the SONET Ring Assignment Problem*". Proc. of the INOC 2005, pp. 833-838, Book 3, INFORMS - Porto – Portugal.
- [2]Bourgeois, M.; Laporte, G. e Semet, F; (2001) *"Heuristic for the black and white traveling salesman problem*". Computers & Operations Research, v. 30, p. 75–85.
- [3]F Feo, T. A. e Resende, M. G. C.; (1995) *"Greedy randomized adaptive search procedures"*. Journal of Global Optimization, v. 6, p. 109-133.
- [4] Garey, M. R., Graham, R. L. e Johnson, D. S.; (1976) *"Some NP- Complete Problems"*, Eigth Annual Symp, on Theory of Computation, p. 10-22.
- [5]Gendered, M.; Hertz, A. e Laporte, G.; (1992) "*New insertion and post optimization procedures for the traveling salesman problem*". Operations Research, v. 40, n. 6, p. 1086–1094.
- [6] Hansen, P. e Mladenovic, N.; (2003) "*Handbook of Metaheuristic*". F. Glover and Kochenagen, Kluwer, p. 145-184.
- [7] Lin S.; (1965) "*Computer solutions of the traveling salesman problem*". Bell System Technical Journal, v .44, p. 2245-2269.
- [8] Silva, G. C.; Martins, S. L. e OCHI, L. Satoru (2004) "*Experimental comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem"*. In Lecture Notes on Computer Science (LNCS) 3059, pp. 498-512, Springer.