Capítulo 10

Métodos Penalizados e Não-Penalizados para o Problema do Caixeiro Viajante com Grupamentos

Mário Mestria*

Resumo: Este capítulo propõe diversos métodos heurísticos para resolver o Problema do Caixeiro Viajante com Grupamentos (PCVG). No PCVG os vértices são divididos em grupos e todos os vértices de cada grupo devem ser visitados de forma contígua. As abordagens desenvolvidas foram baseadas no GRASP, *Iterated Local Search*, Reconexão de Caminhos, Método de Descida em Vizinhança Variável e Inserção mais Próxima. Os resultados mostraram que os métodos heurísticos utilizando a penalização nas arestas intergrupos apresentam os melhores resultados. O desempenho dos métodos heurísticos foi comparado com um algoritmo exato usando o *software* CPLEX paralelo e um Algoritmo Genético da literatura.

Palavras-chave: Meta-heurísticas, Reconexão de caminhos, Método de descida em vizinhança variável, Problema do caixeiro viajante com grupamentos.

Abstract: *This chapter proposes several heuristic methods to solve the Clustered Traveling Salesman Problem (CTSP). In the CTSP the vertices are partitioned into clusters and all vertices of each cluster must be visited continuously. The developed approaches were based on the GRASP, Iterated Local Search, Path Relinking, Variable Neighborhood Descent, and Nearest Insertion. The results showed that the heuristic methods using a penalization for inter-clusters edges achieve the best results. The performance of the heuristic methods was compared with an exact algorithm using the Parallel CPLEX software and a Genetic Algorithm of literature.*

Keywords: *Metaheuristics, Path relinking, Variable neighborhood descent, Clustered traveling salesman problem.*

1. Introdução

O Problema do Caixeiro Viajante (PCV) é um problema clássico mais estudado na área de pesquisa operacional. O Problema do Caixeiro Viajante com Grupamentos (PCVG) é uma extensão do PCV. Desta forma, podemos afirmar que o PCVG também pertence à classe $\mathcal{NP}$-difícil e limita o uso exclusivo de métodos exatos. Neste contexto a metodologia para solucionar o PCVG neste trabalho será através de métodos heurísticos.

O PCVG aparece na literatura em duas versões. Na primeira mais explorada, supõe-se que a ordem de visitas aos grupos já é previamente definida (Anily et al., 1999; Gendreau et al., 1994; Laporte et al., 1996; Potvin & Guertin, 1995), informação esta, que geralmente não é conhecida em muitas aplicações. Na segunda versão, para o qual existe reduzido número de trabalhos, esta ordem de visitas não é definida previamente e esta escolha é deixada para o algoritmo. Para este caso são encontrados na literatura métodos utilizando Algoritmos Genéticos (Ding et al., 2007; Potvin & Guertin, 1996) e Algoritmos α-aproximados (Arkin et al., 1997; Guttmann-Beck et al., 2000). A segunda versão é mais complexa, pois na busca da melhor sequência de visitas dos vértices de cada grupo, deve-se também analisar a melhor ordem de visitas aos grupos que correspondem a encontrar os vértices de entrada e saída de cada grupo. O capítulo deste livro tem como foco apresentar métodos heurísticos para este segundo caso.

O PCVG foi proposto por Chisman (1975) para resolver um problema real de roteamento automático em sistemas de armazenagem. Outras aplicações do PCVG podem ser encontradas em planejamento da produção (Lokin, 1979), despacho de veículos de emergência numa empresa de eletricidade (Weintraub et al., 1999), desfragmentação de discos rígidos, sistemas de manufaturas (Laporte & Palekar, 2002) e em transações comerciais envolvendo supermercados, lojas e fornecedores de mercadorias (Ghaziri & Osman, 2003).

*Autor para contato: mmestria@ifes.edu.br; mmestria@ic.uff.br

Na literatura encontramos alguns trabalhos para o PCVG. Limites inferiores utilizando Relaxação Lagrangeana foram desenvolvidos em (Jongens & Volgenant, 1985) para o PCVG. Os valores médios dos limites inferiores alcançaram *gap* igual a 0,35%. Em (Laporte et al., 1996) uma Busca Tabu combinada com uma fase de diversificação usando um Algoritmo Genético foi proposta para o PCVG, mas a ordem de visita dos grupos já era definida *a priori*. Um Algoritmo Genético (AG) básico (sem componentes que realizam uma busca mais intensiva, tais como, Reconexão de Caminhos (RC) e/ou *Variable Neighborhood Descent* (VND)) foi proposto para o PCVG por Potvin & Guertin (1996). No artigo apresentado por Ding et al. (2007) um outro AG, também básico, usa o critério de visita aos vértices de cada grupo de forma aleatória sem estabelecer uma relação com as distâncias entre os vértices. A escolha da ordem de visita de cada grupo também é aleatória. Observamos nestes trabalhos da literatura que a maioria parte do princípio que já existe uma ordem pré-fixada de visita dos grupos e com isso limita problemas reais que necessitam encontrar a melhor ordem de visitas dos grupos.

Neste capítulo foram desenvolvidos diversos métodos heurísticos utilizando *Greedy Randomized Adaptive Search Procedures* (GRASP) e Reconexão de Caminhos (Resende et al., 2010), *Iterated Local Search* (ILS) (Lourenço et al., 2002) e Método de Descida em Vizinhança Variável (Hansen & Mladenović, 2003) para resolver o PCVG sem estabelecer *a priori* a ordem de visita dos grupos. Foram desenvolvidos dois algoritmos construtivos baseado na heurística Inserção mais Próxima (IMP), um que penaliza as arestas intergrupos e outro que não penaliza. Estes construtivos serão inseridos nos métodos heurísticos para avaliar o impacto da penalização na solução do PCVG.

Na Seção 2 apresenta-se a definição do problema e a formulação matemática para o PCVG. A metodologia para resolver o PCVG é apresentada na Seção 3. Os resultados computacionais, as análises de desempenho dos métodos heurísticos, as comparações entre os métodos penalizados e não-penalizados são apresentados na Seção 4. Finalmente na Seção 5 descrevem-se as conclusões e sugestões dos trabalhos futuros.

2. Problema do Caixeiro Viajante com Grupamentos

O PCV é um problema $\mathcal{NP}$-difícil (Garey & Johnson, 1979). Neste contexto, para o PCV não são conhecidos algoritmos que o resolvam em tempo polinomial. Admitindo que um conjunto de vértices represente as cidades, necessita-se construir um ciclo passando por todos os vértices uma única vez. Quantas soluções são possíveis nesta sequência de vértices? A resposta no caso simétrico é que o número total de soluções possíveis é $n!/2$, onde n é o total de vértices.

2.1 Definição do problema

O Problema do Caixeiro Viajante com Grupamentos (PCVG) é uma extensão do PCV. O PCV é um caso particular do PCVG, onde existe somente um grupo ou analogamente um vértice em cada grupo. Por ser uma generalização do PCV torna-o também $\mathcal{NP}$-difícil (Guttmann-Beck et al., 2000) e (Ding et al., 2007).

Uma descrição do PCVG (Laporte & Palekar, 2002) na estrutura de um grafo é dado a seguir: seja G=(V, A) um grafo completo, direcionado, simétrico e ponderado com um conjunto de vértices $V = \{v_1, v_2, ..., v_n\}$ e um conjunto de arcos $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ em ambas as direções com mesmo peso. O conjunto de vértices V é particionado em m grupos $V_1, V_2, ..., V_m$, onde: $V = \bigcup_{i=1}^{m} V_i$, $\forall\ i$ e $V_i \cap V_j = \emptyset$ para todos i e j, $i \neq j$. Assumindo que um custo não-negativo c_{ij} é associado com o arco $(v_i, v_j) \in A$, o PCVG consiste em determinar um ciclo Hamiltoniano de custo mínimo em G, tal que os vértices de cada grupo são visitados contiguamente. O custo c_{ij} é a distância Euclidiana que corresponde, neste trabalho, a distância geométrica d_{ij} entre dois vértices v_i e v_j no plano bidimensional. Neste contexto, instâncias Euclidianas significam que quando os métodos utilizam estas instâncias para resolução do PCVG fazem o uso das distâncias Euclidianas.

Para exemplificar o PCVG, mostra-se nas Figuras 1 e 2 os vértices e os grupos a que estes pertencem. O Grupo 1 contém os seguintes vértices: 1, 4 e 6. O Grupo 2 os vértices 2, 3 e 7 e o Grupo 3 os vértices 5 e 8.

Uma solução viável para este exemplo é mostrado pela Figura 1, onde a solução para o PCVG forma um ciclo Hamiltoniano que passa por todos os vértices, visitando todos os vértices de cada grupo contiguamente. As arestas (linha cheia) (4,1) e (1,6) mostram as arestas do Grupo 1, para o Grupo 2 (3,2) e (2,7) e a aresta (5,8) no Grupo 3. Por último, as linhas tracejadas mostram as ligações entre os grupos. Outra solução viável, utilizando os mesmos vértices e grupos da Figura 1, mas de custo maior é mostrado pela Figura 2.

As Figuras 1 e 2 mostram que a escolha dos vértices extremos de entrada e de saída para cada grupo é um fator importante para encontrar soluções de melhor qualidade para o PCVG. Através deste exemplo é demonstrada a dificuldade em resolver o PCVG, porque não são conhecidos estes vértices extremos de cada grupo, principalmente na versão onde não é pré-definida a visita dos grupos. Pois, nesta versão mais complexa para o PCVG, os algoritmos devem buscar a melhor sequência de visitas aos vértices de cada grupo e também analisar a melhor sequência de visitas aos grupos.

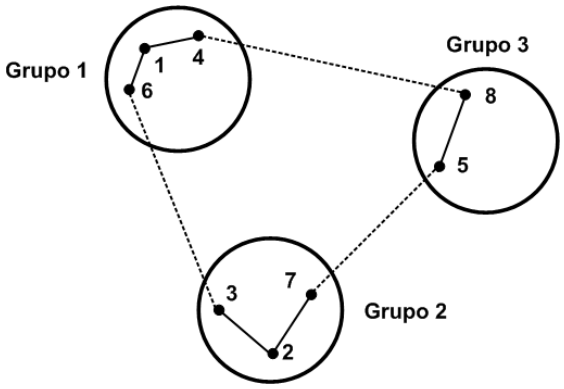

Figura 1. Uma solução viável para o PCVG.

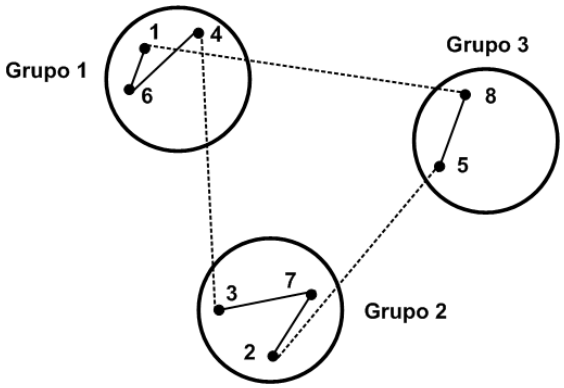

Figura 2. Uma outra solução viável do PCVG de custo maior.

2.2 Formulação matemática

Diversas formulações do PCV podem ser encontradas em (Dantzig et al., 1954; Miller et al., 1960; Lawler et al., 1985). Muitas destas, podem ser adaptadas para o PCVG. Para isto, basta incorporarem-se as restrições que mantém os vértices dos grupos a serem visitados de uma forma contígua. Em Chisman (1975) especificam-se as restrições a serem incorporadas.

Uma formulação para o PCVG utilizando programação inteira `0-1` está descrita em Chisman (1975) e em Miller et al. (1960), e é denominada "Miller/Chisman". Esta formulação é um problema de programação inteira `0-1` através de um grafo completo, direcionado, simétrico e ponderado $G = (V, A)$, onde $V = \{v_1, v_2, ..., v_n\}$ representa o conjunto de vértices e A o conjunto de arcos, sendo $(v_i, v_j) \in A$, $v_i, v_j \in V$ e $i \neq j$. Seja um caixeiro viajante partindo da origem da cidade 1 sem perda de generalidade. A seguir, mostra-se a formulação dada por Miller/Chisman:

$$\text{Min} \qquad z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

s.a.

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \in V \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in V \tag{3}$$

$$u_i - u_j + (n-1)x_{ij} \leq (n-2), \quad 2 \leq i \neq j \leq n \tag{4}$$

$$\sum_{i \in V_k} \sum_{j \in V_k} x_{ij} = \mid V_k \mid -1, \qquad \forall V_k \subset V, \mid V_k \mid \geq 1, k = 1, ..., m \tag{5}$$

$$u_i \geq 0 \qquad 2 \leq i \leq n \tag{6}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{7}$$

A função objetivo 1 requer que se minimize o custo total percorrido pelo caixeiro viajante. As restrições 2 garantem que de cada cidade i só pode sair para uma única cidade j. As restrições 3 significam que para cada cidade j há uma única origem em i. Estes dois conjuntos de restrições implicam que somente duas arestas podem ser conectadas a cada vértice formando um conjunto de ciclos possivelmente disjuntos.

As restrições 4 garantem não haver subciclo na solução e que o caixeiro viajante parte da cidade de número 1 (vértice origem). As variáveis u_i, $i = 2, ..., n$ representam a sequência das cidades i a serem visitadas tendo a relação u_j - $u_k \leq$ -1, $\forall j, k \in V$ sendo k e j cidades contíguas na solução, $x_{jk} = 1$. As restrições 4 utilizam variáveis contínuas u_i que representam o número de arcos menos uma unidade (1) entre o vértice origem e o vértice i. São necessários num ciclo que começa na cidade 1 e termina na última cidade a ser visitada, no máximo $(n-1)$ arcos. Se uma cidade j for visitada após a cidade k, a diferença entre os valores das variáveis contínuas (u_k e u_j) é da ordem de uma unidade (1). Assim, a diferença entre quaisquer duas cidades, exceto a cidade origem, adquire um *limite superior* igual a $(n-2)$. Este limite superior é mostrado nas restrições 4. Observe que no caso trivial com x_{ij}=0, as restrições 4 são satisfeitas utilizando este limite superior.

As restrições 5 indicam que a soma das variáveis x_{ij} na partição do grupo V_k, $k = 1, ..., m$, sendo m o número total de grupos, alcançam o valor da cardinalidade $\mid V_k \mid$ deste grupo menos 1, forçando em conjunto com as restrições 2 e 3, a formação de um caminho dentro deste grupo. As variáveis binárias x_{ij} assumem o valor igual a 1 se a cidade i for conectada a cidade j na solução. Caso contrário, será igual a 0. Esta formulação envolve $n^2 - n + 2 + m$ restrições e n^2 variáveis inteiras e $n - 1$ variáveis contínuas.

3. Metodologia

Para solucionar o PCVG adotamos uma metodologia que utiliza heurísticas construtivas e métodos heurísticos baseados nos *frameworks* das meta-heurísticas. Estes últimos métodos são constituídos de forma básica de três fases: na primeira fase é construída uma solução, na segunda fase uma busca local é realizada e na terceira fase um mecanismo de memória é utilizado. Neste capítulo de livro são mostrados diversos métodos heurísticos baseados no GRASP, RC, ILS e método de descida em vizinhança variável, além da heurística de inserção mais próxima, para solucionar o PCVG genérico onde a sequência de visitas aos grupos de vértices não é pré-fixada. Através dos algoritmos construtivos baseados na heurística de inserção mais próxima serão estabelecidos os métodos Penalizados (P) e os não-Penalizados (nP).

3.1 Heurística de inserção mais próxima

O Algoritmo 1 foi desenvolvido para construir soluções ao PCVG utilizando-se o conceito da heurística de inserção mais próxima. O Algoritmo 1 penaliza as arestas intergrupos tratando o PCVG como um PCV. No Algoritmo 1, o conjunto de vértices $V = \{v_1, v_2, ..., v_n\}$ é definido para representar os elementos pertencentes a solução S.

Algoritmo 1: Algoritmo inserção mais próxima com as arestas penalizadas.

1: Escolher três vértices iniciais v_l, v_{l+1} e v_{l+2}, sendo o primeiro aleatório e os outros dois serão os vizinhos mais próximos;
2: $S \leftarrow \{v_l, v_{l+1}, v_{l+2}\}$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_l, v_{l+1}, v_{l+2}\}$;
4: `penalizar_arestas( );`
5: **enquanto** $C \neq \emptyset$ **faça**
6: Encontrar os vértices mais próximos contidos em C em relação aos vértices pertencentes ao ciclo já formado e calcular o custo incremental $c(v_k)$;
7: $c_{min} \leftarrow min\{c(v_k) : v_k \in C\}$;
8: $c_{max} \leftarrow max\{c(v_k) : v_k \in C\}$;
9: $LRC \leftarrow \{v_k \in C : c(v_k) \leq c_{min} + \alpha * (c_{max} - c_{min})\}$;
10: Selecionar um elemento v_k da LRC aleatório;
11: $S \leftarrow S \cup \{v_k\}$; (conectar v_k aos vértices do ciclo (i e $i+1$), cujo custo (*price*) $p(v_k)$=$\{p_{i,k}$+$p_{k,i+1}$-$p_{i,i+1}\}$ seja mínimo e atualizar o ciclo);
12: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_k\}$;)
13: **fim enquanto**
14: Retornar S

No início do Algoritmo 1 são escolhidos três vértices, sendo o primeiro aleatório e os dois outros usando o conceito de vizinho mais próximo. No passo 4 o procedimento `penalizar_arestas( )` é utilizado para alterar os valores das arestas intergrupos. A alteração penaliza os custos entre os vértices que pertencem a grupos distintos somando um valor constante igual a L, onde $L >> \max\{c_{ij}\}$. Em seguida, os vértices pertencentes ao conjunto de candidatos são avaliados pelos custos incrementais (passo 6), computando-se as distâncias destes vértices contidos no conjunto em relação aos vértices da solução parcial. O valor de c_{min}, menor custo incremental e o valor de c_{max}, maior custo incremental são armazenados para calcular a LRC

(Lista Restrita de Candidatos) (Seção 3.3.1). A seguir, no passo 10, seleciona-se aleatoriamente um vértice da LRC que é conectado aos vértices do ciclo da solução parcial, onde o custo de inserção seja mínimo, passo 11. Atualiza-se o conjunto de candidatos e se repetem os passos 6 até 12. O algoritmo termina quando o conjunto de candidatos estiver vazio.

Um segundo algoritmo construtivo, foi desenvolvido para o PCVG que o trata na forma original, no qual não penaliza as arestas intergrupos (*não realiza o passo 4 do Algoritmo 1*) na formação do conjunto de candidatos. Neste segundo algoritmo, *semelhante o passo 6 do Algoritmo 1*, primeiro são escolhidos os vértices que irão compor o conjunto de candidatos sendo aqueles que pertencem ao mesmo grupo corrente dos vértices contidos na solução parcial. Quando todos os vértices do grupo corrente já estão na solução parcial, todos os outros vértices restantes que ainda não foram selecionados são considerados candidatos.

3.2 Método de descida em vizinhança variável

Vários métodos heurísticos híbridos utilizam como busca local o método de Descida em Vizinhança Variável (VND – *Variable Neighborhood Descent*) (Hansen & Mladenović, 2003) para solucionar problemas de otimização combinatória tais como o problema do caixeiro viajante com coleta e entrega de uma comodidade (Hernández-Pérez et al., 2009). Para o PCVG, propõe-se verificar o impacto de incluir nas meta-heurísticas (GRASP e ILS) módulos de buscas locais mais eficientes utilizando os conceitos do VND. Neste contexto foram desenvolvidas várias estruturas de vizinhanças para realizar uma fase de busca local baseadas no VND.

Algoritmo 2: *Variable neighborhood descent.*

1: **repita**
2: $k \leftarrow 1$
3: **enquanto** $k \leq k'_{max}$ **faça**
4: Encontrar a melhor vizinhança x' de x, ($x' \in N'_k(x)$);
5: **se** $f(x') < f(x)$ **então**
6: $x \leftarrow x'$;
7: $k \leftarrow 1$;
8: **senão**
9: $k \leftarrow k + 1$;
10: **fim se**
11: **fim enquanto**
12: **até** nenhum melhoramento é obtido

O Algoritmo 2 descreve o VND e define-se o conjunto de vizinhanças N'_k, k=1, 2, ..., k'_{max} e uma solução inicial x.

Algoritmo 3: Procedimento *Drop*.

1: **para** k **de** 1 **até** n **faça**
2: g_k=c_{a_k,v_k} + c_{v_k,s_k} - c_{a_k,s_k} (computar todos os ganhos de economia para cada vértice v_k);
3: **fim para**
4: $\bar{k} = argmax\{g_k\colon k \in G\}$;
5: Remover $v_{\bar{k}}$ do ciclo;

Para realizar o método de Descida em Vizinhança Variável propõem-se neste trabalho, quatro tipos de estruturas de vizinhanças. A primeira N'_1 é obtida através de um movimento de deslocamento do vértice semente no ciclo de uma posição para outra, 1-*Shift*. Para definir a segunda estrutura N'_2 precisa-se definir dois tipos de procedimentos: *Drop* e *Cheap*. O procedimento *Drop* varre o ciclo de uma solução e retira deste o vértice que proporcionará a maior economia com sua retirada.

O procedimento *Cheap* insere um vértice não pertencente a um ciclo parcial entre dois vértices adjacentes pertencentes ao ciclo, cuja inserção permita um custo adicional mínimo a solução. O ciclo construído após a inserção do vértice deverá permanecer viável. Esse procedimento está relacionado à heurística de construção inserção mais barata desenvolvida para o PCV (Reinelt, 1994).

Diante destas duas definições de procedimentos pode-se estabelecer a segunda estrutura de vizinhança N'_2, 1-*DropCheap*. Uma sequência de procedimentos *Drop* e *Cheap* é realizada sobre uma solução até que nenhum melhoramento seja encontrado. Estruturas de vizinhanças semelhantes a do tipo 1-*DropCheap* são vistas em (Gomes et al., 2000; de Assumpção Drummond et al., 2002). O Algoritmo 3 mostra o procedimento *Drop*

Algoritmo 4: Procedimento *Cheap*.

1: **para** i **de** 1 **até** m **faça**
2: $h(k)_i$=$c_{i,k}$ + $c_{k,i+1}$ - $c_{i,i+1}$, (computar todos os custos de inserção do vértice v_k entre os vértices i e $i+1$ no ciclo C);
3: **fim para**
4: $\bar{k} = argmin\{h(k)_i\colon i \in H\}$;
5: Inserir $v_{\bar{k}}$ no ciclo C entre os vértices i e $(i+1)$;

considerando-se: $V = \{v_1, v_2, ..., v_n\}$ o conjunto de vértices de um ciclo; g_k o ganho da economia pela retirada do vértice v_k do conjunto V; a_k e s_k o antecessor e sucessor, respectivamente do vértice v_k; e o conjunto de ganhos g_k para cada vértice k, $G = \{g_1, g_2, ..., g_n\}$.

O Algoritmo 4 apresenta o procedimento *Cheap* com as seguintes definições: v_k o vértice a ser inserido; n o total de vértices; o conjunto de vértices $V = \{v_1, v_2, ..., v_n\}$ e $W = \{v_1, v_2, ..., v_m\}$, sendo W=$(V \backslash v_k)$; C o ciclo formado pelos vértices em W; $h(k)_i$ o custo de inserção do vértice v_k entre os vértices i e $(i+1)$ em C; e $H = \{h(k)_1, h(k)_2, ..., h(k)_m\}$ o conjunto de custos $h(k)_i$ de inserção do vértice k entre os vértices i e $(i+1)$.

Algoritmo 5: *Variable neighborhood descent* para o PCVG.

1: $k \leftarrow 1$
2: **enquanto** $k \leq 4$ **faça**
3: **se** $k = 1$ **então**
4: $x^{'} \leftarrow$ 1-*Shift(x)*
5: **senão se** $k = 2$ **então**
6: $x^{'} \leftarrow$ 1-*DropCheap(x)*
7: **senão se** $k = 3$ **então**
8: $x^{'} \leftarrow$ 2-*Swap(x)*
9: **senão**
10: $x^{'} \leftarrow$ 2-*Optimal(x)*
11: **fim se**
12: **se** $f(x^{'}) < f(x)$ **então**
13: $x \leftarrow x^{'}$;
14: $k \leftarrow 1$;
15: **senão**
16: $k \leftarrow k + 1$;
17: **fim se**
18: **fim enquanto**
19: Retornar x;

A terceira vizinhança $N_3^{'}$ é obtida através de uma permuta entre os vértices de um mesmo grupo (*cluster*), 2-*Swap*. A 2-*Swap* foi aplicada em outras variantes do PCV (de Assumpção Drummond et al., 2002; Hernández-Pérez et al., 2009). A quarta vizinhança $N_4^{'}$ utiliza a heurística 2-*Opt* (Reinelt, 1994) para trocar as arestas intragrupos e as intergrupos. Nesta quarta vizinhança quando são retiradas as arestas intragrupos e é reconstruída uma nova solução, somente a sequência dos vértices dentro do grupo é alterada. Quando as arestas intergrupos são trocadas, somente a sequência dos grupos é alterada. Diante das definições anteriores, o método de descida VND proposto neste trabalho a ser incluído nos métodos heurísticos baseados no GRASP e ILS como módulo de busca local é descrito pelo Algoritmo 5.

3.3 GRASP - *Greedy Randomized Adaptive Search Procedure*

Dentre as meta-heurísticas existentes na literatura, uma que tem se destacado é o GRASP. GRASP é um algoritmo iterativo no qual cada iteração consiste de duas fases: uma de construção e outra de busca local. Na fase de construção, uma solução viável é construída, e na fase de busca local sua vizinhança é investigada até um mínimo local ser encontrado. A melhor solução obtida ao final de um determinado número de iterações é retornada como a solução do algoritmo. Neste trabalho utiliza-se memória adaptativa no GRASP através do uso e atualização dinâmica de um conjunto elite (CE), uma técnica de busca denominada Reconexão de Caminhos (RC) (Resende et al., 2010).

Na fase de construção, a cada iteração, o conjunto de elementos candidatos é formado por todos os elementos que podem ser incorporados à solução parcial sem destruir a sua viabilidade. A seleção do próximo

elemento a ser incorporado é determinada pela avaliação de todos os elementos candidatos, através de uma função de avaliação gulosa. Essa função gulosa usualmente representa, num problema de minimização, o aumento incremental na função custo devido à incorporação desse elemento dentro da solução construída (aspecto guloso do algoritmo). A avaliação dos elementos por essa função conduz a criação da Lista Restrita de Candidatos (LRC) formada por um subconjunto dos elementos melhor avaliados. O elemento a ser incorporado à solução parcial é selecionado aleatoriamente da LRC (aspecto probabilístico do algoritmo). O elemento selecionado é incorporado à solução parcial, a lista de candidatos é atualizada e os custos incrementais são reavaliados (aspecto adaptativo do algoritmo).

As soluções geradas pela construção aleatória gulosa do GRASP não representam necessariamente ótimos locais. A fase de busca local visa tentar melhorar a solução construída. Um algoritmo de busca local trabalha em um modo iterativo substituindo sucessivamente a solução corrente por uma solução melhor na vizinhança da solução corrente. O algoritmo termina quando nenhuma solução melhor é encontrada na vizinhança.

3.3.1 Construção da lista restrita de candidatos

Uma característica atraente do GRASP é a facilidade para a implementação e a existência de poucos parâmetros a serem ajustados. GRASP possui basicamente dois parâmetros principais, um relacionado ao critério de parada e outro à cardinalidade da LRC. O critério de parada pode ser determinado pelo número máximo de iterações ou um tempo limite de processamento.

O parâmetro α relacionado a construção da LRC usada na primeira fase do GRASP é descrito a seguir. Considere o problema como sendo de minimização sem perda de generalidade e um conjunto V, $V = \{v_1, v_2, ..., v_n\}$, composto de elementos a serem inseridos numa solução. Designa-se por $c(v_i)$ o custo incremental associado com a incorporação do elemento $v_i \in V$ dentro da solução construída. Representa-se por c_{min} e c_{max}, o menor e o maior custo incremental, respectivamente.

A LRC é construída com elementos $v_i \in V$ com os melhores custos incrementais $c(v_i)$. Esta lista pode ser limitada pelo número de elementos ou por sua qualidade. No primeiro caso, a LRC é construída de l elementos com os melhores custos incrementais, onde l é um parâmetro. No segundo caso, considera-se a LRC associada com um parâmetro limiar $\alpha \in [0, 1]$. A LRC é formada por todos os elementos viáveis $v_i \in V$, os quais podem ser inseridos dentro da solução parcial construída sem destruir a viabilidade e cuja qualidade é superior ao valor limiar. Assim, verifica-se: $\{c(v_i) \in [c_{min}, c_{min} + \alpha(c_{max} - c_{min})]\}$. O caso $\alpha = 0$ corresponde ao algoritmo puramente guloso, enquanto $\alpha = 1$ equivale ao algoritmo totalmente aleatório.

A versão reativa do GRASP tenta determinar dinamicamente os valores de α a serem utilizados. A seguir, a estratégia da escolha do parâmetro α de modo reativo (Resende & Ribeiro, 2002) é descrita sucintamente. Esta estratégia procura incorporar um mecanismo de aprendizado usando os valores das soluções produzidas ao longo das iterações. Neste caso, o valor de α não será fixo. Seja, $\Psi = \{\alpha_1, ..., \alpha_m\}$, o conjunto dos possíveis valores para α de tamanho m. A probabilidade inicial associada com a escolha de cada valor α_i será igual a p_i=$1/m$, com $i = 1, ..., m$. Seja também z^* a melhor solução encontrada até o momento e z_{m_i} o valor médio de todas as soluções encontradas usando $\alpha = \alpha_i, i = 1, ..., m$. As probabilidades de seleção serão periodicamente reavaliadas, com p_i=$q_i / \sum_{j=1}^{m} q_j$, q_i=z^*/z_{m_i} para $i = 1, ..., m$. O valor de cada q_i será maior quando o valor médio z_{m_i} diminuir. Os maiores valores de q_i correspondem aos valores mais adequados para os α_i. As probabilidades associadas com estes valores mais *apropriados* aumentam à medida que os valores de α são reavaliados.

3.3.2 Reconexão de caminhos

A Reconexão de Caminhos (RC) foi originalmente proposta por Glover (1996) como uma estratégia de intensificação que explora trajetórias conectando soluções elite obtidas pela Busca Tabu e *Scatter Search*. O uso da Reconexão de Caminhos com o GRASP foi primeiro proposto por Laguna & Martí (1999) como uma estratégia de intensificação. Partindo de uma ou mais soluções elites, a Reconexão de Caminhos permite que outras soluções sejam geradas e exploradas percorrendo-se novos caminhos no espaço de solução. Isso é efetuado selecionando-se os movimentos que introduzem atributos contidos nas soluções guias a serem inseridos nas soluções base. Basicamente *duas estratégias* são usadas com o uso da Reconexão de Caminhos com o GRASP: uma aplicada numa etapa de pós-otimização para todos os pares de soluções elite e outra, aplicada num processo de intensificação para cada ótimo local obtido depois da fase de busca local.

Na segunda estratégia, a Reconexão de Caminhos é aplicada ao pares (x_1, x_2) de soluções extremos, onde x_1 é a solução obtida usualmente por uma busca local e x_2 é uma das soluções elite (*conjunto das k melhores soluções obtidas até o momento pelo GRASP*) escolhida aleatoriamente deste conjunto construído ao longo das iterações (Resende & Ribeiro, 2002). A cardinalidade deste conjunto é limitada a um valor máximo (`Max_Elite`).

Algoritmo 6: Reconexão de Caminhos(S_b, S_g).

1: $S_{ini} \leftarrow S_b$;
2: $S_{rc} \leftarrow S_b$;
3: **enquanto** $Novos\_Sub\_Conj(S_{ini}, S_g) \neq \emptyset$ **faça**
4: `sel_diferencas`(S_{ini}, S_g, S_{int});
5: $S_{ini} \leftarrow S_{int}$;
6: **se** S_{ini} for melhor que S_{rc} **então**
7: $S_{rc} \leftarrow S_{ini}$;
8: **fim se**
9: **fim enquanto**
10: Retornar a solução S_{rc};

O Algoritmo 6 mostra a Reconexão de Caminhos realizada entre duas soluções elites, solução base S_b e solução guia S_g. O procedimento $Novos\_Sub\_Conj(S_k, S_j)$ estabelece os subconjuntos novos entre as soluções S_k e S_j definidos pelas diferenças entre estas duas soluções. O Algoritmo 6 estabelece no passo 1 que S_b será a solução inicial (S_{ini}). Em seguida, o Algoritmo 6 analisa todas as diferenças existentes entre a solução base e a solução guia (passos 3 a 9). Depois de escolhido o par de soluções, S_{ini} e S_g, percorre o caminho de S_{ini} a S_g e uma diferença entre S_{ini} a S_g é retornada pela solução S_{int} (solução intermediária) através do procedimento `sel_diferencas`(S_{ini}, S_g, S_{int}). A solução S_{int} será a solução inicial da próxima etapa, passo 5. A melhor de todas as soluções intermediárias nas etapas da Reconexão de Caminhos será a solução de saída, passos 6 a 8.

Na versão proposta para a RC foi escolhida como solução base S_b, a solução cujo custo seja menor entre as duas soluções extremos (S_b e S_g). A solução de pior qualidade será a solução guia S_g.

Uma das principais características para realização da Reconexão de Caminhos é a construção de um conjunto elite (CE). A cardinalidade do conjunto elite é formada por um número fixo pré-definido de soluções e a estratégia para escolher as soluções que irão pertencer ao conjunto elite é a seguinte: *primeiro*, se o conjunto elite estiver incompleto, a solução gerada na iteração do GRASP simplesmente é inserida neste e *segundo*, se o conjunto elite já estiver completo, a solução gerada na iteração do GRASP é inserida no conjunto elite, caso seu custo da função objetivo seja menor que o custo da solução de pior qualidade do conjunto elite e a solução apresente uma diferença mínima na sua estrutura para cada solução presente no CE. Depois de verificada as duas condições, necessita-se saber qual será a solução que deve ser retirada do conjunto elite. A solução que irá sair deverá ter em relação à solução que entra no conjunto elite um custo maior e uma menor diferença na estrutura entre as duas soluções. A ideia da construção do conjunto elite é ter um conjunto de soluções de boa qualidade.

A diferença na estrutura é calculada pelo número de arestas. Assim, dada duas soluções (S_i e S_j) computa-se esta diferença pelo número de arestas necessárias a serem inseridas em uma solução S_i para se transformar em outra solução S_j. A versão proposta para a RC realiza uma busca mais intensiva, onde a reconexão é feita entre cada par de soluções elite. Caso surjam novas soluções nesta recombinação que atualizem o CE, uma nova etapa será realizada entre as novas soluções obtidas. O processo termina quando o CE não for mais atualizado. Esta RC é ativada durante as iterações do GRASP sempre que o CE for totalmente renovado. A RC é realizada em cada iteração do GRASP somente quando muda todas as soluções do CE. Esta versão combina as *duas estratégias* básicas expostas anteriormente: etapa de **pós-otimização** e processo de **intensificação**.

3.3.3 Método heurístico baseado no GRASP para o PCVG

O GRASP desenvolvido para o PCVG utiliza parte da estrutura do GRASP tradicional (Resende & Ribeiro, 2002), onde em cada iteração executa-se uma fase de construção de uma solução e uma fase de busca local. Na etapa de construção, utiliza-se o método de Inserção mais Próxima com as arestas penalizadas ou Inserção mais Próxima com as arestas não penalizadas, ambos com valor de α calculado pela versão reativa e na busca local o VND. No GRASP será inserido ainda uma Reconexão de Caminhos durante suas iterações. O GRASP será denominado de G-IMPP quando utiliza o algoritmo construtivo (Algoritmo 1) que penalizada as arestas intergrupos e de G-IMPnP quando não penaliza tais arestas.

Durante a execução do GRASP é gerado e atualizado um CE que armazena as *num_eli* melhores soluções distintas encontradas pelo GRASP. Tal procedimento equivale a usar uma memória que armazene no CE informações relevantes encontradas em iterações passadas. Como este conjunto sofre atualizações sempre que uma nova solução de boa qualidade é encontrada entre as iterações do GRASP, esta *memória é adaptativa*.

O GRASP utiliza para a busca local a meta-heurística VND (Hansen & Mladenović, 2003). No VND são utilizados quatro tipos de estruturas de vizinhanças definidas anteriormente na Seção 3.2. A primeira $N_1^{'}$ denominada de 1-*Shift*, a segunda $N_2^{'}$ (1-*DropCheap*), a terceira $N_3^{'}$ (2-*Swap*) e a quarta $N_4^{'}$ utiliza a heurística 2-Opt. Nesta proposta de trabalho foram adotadas as seguintes estratégias ao VND para reduzir o esforço computacional decorrente do uso das estruturas de vizinhanças. Quando são utilizadas as vizinhanças $N_1^{'}$ e $N_3^{'}$, adota-se a estratégia *first improvement* (ou seja, na primeira melhora ocorre o movimento para esta solução aprimorante) (Hansen & Mladenović, 2003), enquanto que, quando são utilizadas $N_2^{'}$ e $N_4^{'}$, adota-se o método *best improvement* (são analisadas todas as soluções vizinhas e ocorre o movimento para a melhor solução).

O método heurístico baseado no GRASP para o PCVG está representado no Algoritmo 7 com todos os componentes: construção da solução, busca local e Reconexão de Caminhos. Para o Algoritmo 7 considera-se: S^* a melhor solução; *maxprob* o n^{o} máximo de iterações para mudar a distribuição de probabilidade dos alfas; *num_iter* o n^{o} máximo de iterações; e *num_eli* o n^{o} máximo da cardinalidade do conjunto elite; O procedimento $Max\_Sol\_Elite(S^{'}, iter)$ (Algoritmo 8) representa a construção do conjunto de soluções elites e o $Rec\_Cam(U^{elite}, S^{'})$ representa a Reconexão de Caminhos durante as iterações do GRASP. O Algoritmo 7 mostra ainda os procedimentos `calcularalfa( )`, utilizado para selecionar o alfa a ser utilizado pela versão reativa e `atualizardisalfas( )`, para atualizar a distribuição de probabilidade dos alfas.

Algoritmo 7: Algoritmo GRASP para o PCVG.

1: Início;
2: $iterprob \leftarrow 1$; (incremento para atingir o valor *num_prob*)
3: $iter \leftarrow 1$; (incremento para atingir o valor *num_iter*)
4: **enquanto** $iter <= num\_iter$ **faça**
5: α= `calcularalfa( )`; (calcular alfa pela versão reativa)
6: $S^{'} \leftarrow Construcao\_Aleatoria\_Gulosa$;
7: $S^{'} \leftarrow Busca\_Local\ (S^{'})$;
8: **se** $iter > num\_eli$ **então**
9: $S^{'} \leftarrow Rec\_Cam(U^{elite}, S^{'})$;
10: **fim se**
11: **se** $S^{'}$ for melhor que S^* **então**
12: $S^* \leftarrow S^{'}$;
13: **fim se**
14: $U^{elite} \leftarrow Max\_Sol\_Elite(S^{'}, iter)$; (conjunto de soluções elites)
15: **se** $iterprob = maxprob$ **então**
16: `atualizardisalfas( )`; (atualizar a distribuição de probabilidade os alfas)
17: $iterprob \leftarrow 0$;
18: **fim se**
19: $iterprob++$;
20: $iter++$;
21: **fim enquanto**
22: Retornar solução S^*;
23: Fim.

O Algoritmo 8 descreve o procedimento para construção do conjunto elite. Os parâmetros de entrada são: $S^{'}$ a solução do GRASP e *num* é uma variável que contabiliza a ordem no qual as soluções são geradas. O retorno deste algoritmo é o conjunto de soluções elites (U^{elite}=$\{S_1, ..., S_{maxelite}\}$). Ainda para o Algoritmo 8 necessita-se definir os parâmetros: *maxelite* o n^{o} máximo de soluções do conjunto elite; *mindif* a mínima diferença permitida entre duas soluções; *valordif* o valor da diferença entre as soluções; *cont* contador das soluções, cuja diferença mínima foi alcançada; e $dif(S_i, S_j)$ o cálculo da diferença entre a solução S_i e S_j.

Ainda, para o Algoritmo 8, caso o conjunto elite (CE) esteja incompleto, a solução ($S^{'}$) é inserida neste (passos de 1 a 4). Caso o CE estiver completo, a solução a ser inserida ($S^{'}$) no conjunto deverá ter o seu custo da função objetivo $f(S^{'})$ menor do que o custo da solução de pior qualidade ($S_{maxelite}$) do CE e deve apresentar uma diferença mínima ($mindif$) na estrutura de seus elementos para cada solução presente no conjunto U^{elite}=$\{S_1, ..., S_{maxelite}\}$ (passos de 5 a 12).

Após as duas condições terem sido atendidas, necessita-se encontrar a solução a ser retirada S_{out} do conjunto elite. A solução que irá ser retirada deve ter em relação à solução a ser inserida no CE um custo maior e a menor diferença ($valordif$) nos elementos entre estas duas soluções (passos 13 a 26). O Algoritmo 8 no final retorna o conjunto U^{elite}, passo 29.

Algoritmo 8: $Max\_Sol\_Elite(S^{'}, num)$ – Construção do CE.

se $num <= maxelite$ **então**
  $U^{elite} \leftarrow U^{elite} \cup \{S^{'}\}$;
  Reordenar U^{elite}; (reordenar o conjunto elite)
senão
  se $f(S^{'}) < f(S_{maxelite})$ **então**
    $cont \leftarrow 0$;
    para j **de** 1 **até** $maxelite$ **faça**
      Seleciona S_j em U^{elite};
      se $dif(S^{'}, S_j) > mindif$ **então**
        $cont++$;
      fim se
    fim para
    se $cont{=}maxelite$ **então**
      $valordif \leftarrow dif(S^{'}, S_{maxelite})$;
      $S_{out} \leftarrow S_{maxelite}$;
      para j **de** 1 **até** $maxelite$ **faça**
        Seleciona S_j em U^{elite};
        se $(f(S^{'}) < f(S_j))$ **and** $(dif(S^{'}, S_j) < valordif)$ **então**
          $valordif \leftarrow dif(S^{'}, S_j)$;
          $S_{out} \leftarrow S_j$; (solução a ser retirada do conjunto elite)
        fim se
      fim para
      $U^{elite} \leftarrow U^{elite} \setminus \{S_{out}\}$;
      $U^{elite} \leftarrow U^{elite} \cup \{S^{'}\}$;
      Reordenar U^{elite};
    fim se
  fim se
fim se
Retornar o conjunto U^{elite};

3.4 ILS - *Iterated Local Search*

Outra meta-heurística que vem se mostrando eficiente para obter soluções aproximadas de boa qualidade para problemas de otimização combinatória é o método ILS (Lourenço et al., 2002). As características da meta-heurística ILS são simplicidade, eficiência, facilidade de implementação e robustez. Sua estrutura canônica é formada por quatro módulos principais: construção da solução, busca local, perturbação e critério de aceitação. Cada módulo pode ser tratado de forma independente possibilitando maior facilidade em sua atualização. É claro que o sucesso do ILS depende exclusivamente do projeto de cada módulo.

A maneira como ILS explora o espaço de soluções de ótimos locais é definido a seguir: primeiro aplica-se uma perturbação à solução corrente S que permite um estado intermediário S'. Sobre S', aplica-se uma busca local obtendo-se S''. Se a solução S'' for considerada pelo critério de aceitação, esta será a próxima solução a sofrer a perturbação, caso contrário é retornada a solução S (Lourenço et al., 2002).

Como as perturbações dependem sensivelmente das soluções obtidas nas etapas anteriores guiadas pelo critério de aceitação, pode-se dizer que ILS incorpora mecanismo de memória. Uma das grandes vantagens do ILS está na estrutura modular de cada componente. Assim, pode-se desenvolver os módulos independentes e incorporar complexidades maiores a cada um com a finalidade de tentar obter soluções de alta qualidade. ILS é descrito pelo Algoritmo 9 baseada no *framework* de Lourenço et al. (2002). No Algoritmo 9, a solução S^* representa a melhor solução encontrada e o módulo *Construcao_Aleatoria_Gulosa* representa a construção de solução. No Algoritmo 9 são mostrados ainda, os módulos perturbação, busca local, e critério de aceitação, passos 6, 7 e 8, respectivamente.

3.4.1 Método heurístico baseado no ILS para o PCVG

Um tipo de perturbação utilizada no PCV com sucesso foi o movimento *double-bridge* (Martin et al., 1992). Este tipo de perturbação remove quatro arestas na solução e reconstrói uma nova solução substituindo as arestas removidas por outras arestas para obter um movimento não sequencial nas novas arestas inseridas. Para o PCVG a perturbação utilizada foi realizada sob duas formas: a primeira denominada (*Perttc*) realiza

Algoritmo 9: Procedimento *Iterated Local Search*.

```
1: Início;
2: S^0 ← Construcao_Aleatoria_Gulosa;
3: S ← Busca_Local(S^0);
4: S* ← S
5: repita
6:    S' ← Perturbacao(S,história);
7:    S'' ← Busca_Local(S');
8:    S ← Criterio_Aceitacao(S,S'',história);
9:    se S for melhor que S* então
10:      S* ← S;
11:   fim se
12: até critério de parada não satisfeito
13: Retornar Ssolução S*;
14: Fim.
```

uma troca aleatória entre dois grupos (*two clusters*) sem alterar os vértices internos dos grupos. A segunda (*Pert*) realiza o movimento *double-bridge* aplicado aos vértices em um grupo escolhido aleatório. A sequência dos grupos não é alterada, somente a sequência dos vértices do grupo escolhido é modificada.

O critério de aceitação (*CritAceit*) construído na etapa seguinte da perturbação foi escolhido entre as duas soluções ótimas locais, S e $S^{''}$, descritas anteriormente pelo Algoritmo 9 e a solução S^*, a melhor solução até a etapa corrente do algoritmo. Para o critério de aceitação (*CritAceit*) a solução que tiver o menor custo entre a três soluções (S, $S^{''}$, S^*) será escolhida como solução para a etapa seguinte da perturbação. Assim, tem-se:

$$CritAceit(S, S^{''}, história) = \begin{cases} S & \text{se} \quad c(S) < c(S^{''}) \quad \text{e} \quad c(S) < c(S^*), \\ S^{''} & \text{se} \quad c(S^{''}) < c(S) \quad \text{e} \quad c(S^{''}) < c(S^*), \\ S^* & \text{caso contrário,} \end{cases} \tag{8}$$

O método heurístico baseado no ILS para o PCVG é representado pelo Algoritmo 10. Neste algoritmo, o número máximo de iterações para mudar a distribuição de probabilidade dos alfas é representado pelo parâmetro *maxprob*, o parâmetro *iterprob* representa o incremento para atingir o valor *maxprob*, o *iterext* denota o número máximo de iterações externas e o *maxiter* o número máximo de iterações internas.

No Algoritmo 10, a solução S^0 representa uma solução construída pelo módulo *Construcao_Aleatoria_Gulosa*, no qual é utilizada para iniciar o algoritmo e para gerar soluções diversificadas a cada iteração mais externa. A solução S corresponde a solução corrente no qual é aplicada uma perturbação, a solução S' é a solução que está num estado intermediário, onde é aplicada uma busca local obtendo a solução S'' e a solução S^* representa a melhor solução encontrada. No Algoritmo 10 são mostrados ainda os módulos: de perturbação (*Pert*), de busca local (*VND*) e critério de aceitação (*CritAceit*). Estes módulos foram os que permitiram os métodos heurísticos utilizando o ILS a encontrar as melhores soluções para o PCVG. O procedimento `calcularalfa( )` seleciona o alfa (α) na versão reativa a ser utilizado pelo módulo construtivo e o procedimento `atualizardisalfas( )` atualiza a distribuição de probabilidade dos alfas, sempre que *iterprob* atingir seu valor máximo igual a *maxprob*.

No Algoritmo 10, no passo 3, inicialmente é obtida uma solução S^0. Para cada iteração externa do algoritmo, passos 6 a 23, tem-se que: no passo 7 é selecionado um valor para α através de uma distribuição de probabilidade, uma nova solução S^0 é construída no passo 8 e, no passo 9, a busca local é executada pelo VND obtendo a solução S. Para cada iteração interna do algoritmo, passos 10 a 17, a perturbação é aplicada gerando a solução S' (passo 11) e a busca local é executada de novo pelo VND produzindo a solução S'', passo 12. No passo 13, é executado o critério de aceitação e a melhor solução S^* é atualizada no passo 15. No passo 19, os alfas são atualizados pela distribuição de probabilidade na versão reativa e, finalmente, no passo 24, a melhor solução S^* é retornada.

4. Resultados Computacionais

Para avaliar os métodos heurísticos e um método exato ao PCVG foram utilizadas as instâncias descritas em (Mestria, 2011), por apresentar uma biblioteca pública com instâncias para este problema. Todas as instâncias são Euclidianas e estão disponíveis na internet [1]. O CPLEX paralelo foi executado num computador com 4 núcleos Intel Core 2 Quad, 2,83 GHz com 8 GB de RAM. O sistema operacional utilizado foi Linux Ubuntu

[1] http://labic.ic.uff.br/Instance/index.php

Algoritmo 10: Algoritmo ILS-VND.

```
1: Início;
2: iterprob ← 1;
3: S^0 ← Construcao_Aleatoria_Gulosa;
4: S ← VND(S^0);
5: S* ← S
6: para i=1 to iterext faça
7:    α= calcularalfa( );
8:    S^0 ← Construcao_Aleatoria_Gulosa;
9:    S ← VND(S^0);
10:   para i=1 to maxiter faça
11:      S' ← Pert(S);
12:      S'' ← VND(S');
13:      S ← CritAceit(S, S'',história);
14:      se S for melhor que S* então
15:         S* ← S;
16:      fim se
17:   fim para
18:   se iterprob = maxprob então
19:      atualizardisalfas( );
20:      iterprob ← 0;
21:   fim se
22:   iterprob++;
23: fim para
24: Retornar Solução S*;
25: Fim.
```

versão 4.3.2-1. Os métodos heurísticos foram codificados em linguagem de programação `C` e executados no mesmo computador descrito anteriormente, mas utilizando somente um núcleo.

4.1 Testes utilizando um método exato

A Tabela 1 mostra as instâncias (coluna 1) com os seus identificadores (coluna 2), seguido do número de vértices (nós), número de grupos e tipo. Na Tabela 1 são mostradas ainda, as soluções e os limites inferiores gerados pelo método exato através do software CPLEX. Na última coluna é mostrado o percentual do gap_{li} (%) entre valor da solução e limite inferior calculado da seguinte forma:

$$gap_{li} = 100 * \left(\frac{best - li}{best + \epsilon} \right), \tag{9}$$

onde, *best* é o melhor valor encontrado pelo CPLEX, *li* o limite inferior dado pelo CPLEX e ϵ igual 10^{-10}.

Devido às longas iterações ocorridas no CPLEX para a maioria das instâncias, um tempo limite de 7200s foi determinado para sua execução. Para a instância I_3, a solução ótima foi encontrada pelo CPLEX em 442s. Todas as outras execuções utilizaram 7200s, sendo, portanto abortadas antes de se encontrar ou confirmar uma solução ótima.

4.2 Testes dos métodos construtivos e do VND

A Figura 3 mostra os *gaps* alcançados pelas heurísticas IMPP, IMPnP e pelo método VND utilizando a heurística IMPP, denominado de VND-IMPP, além do VND combinado com a IMPnP (VND-IMPnP). Foram escolhidas várias instâncias de diversos *tipos* variando o número de vértices entre 200 e 1000 e o número de grupos entre cinco e 144 vértices, totalizando 53 instâncias. Para cada execução dos métodos heurísticos foram utilizadas 200 iterações com o parâmetro α na versão reativa que gerencia o tamanho da LRC na etapa de construção. Os valores de α variam no intervalo de [0,1]. O valor para atualizar periodicamente as probabilidades dos αs foi igual a 50. Todos os métodos foram executadas dez vezes. Assim, tem-se 2120 testes computacionais. O desempenho dos algoritmos construtivos (IMPP e IMPnP) e com buscas locais (VND-IMPP e VND-IMPnP) para os valores médios são mostrados na Figura 3 através do gap_h médio (eixo y), Equação 10, para cada *tipo* de instância. Observa-se que os algoritmos (eixo x) que utilizaram penalização mostram-se com melhores resultados na média do que os algoritmos que não aplicaram a estratégia de penalização.

Tabela 1. Instâncias com seus identificadores (J_k), número de nós, número de grupos, tipo e valores alcançados pelo CPLEX com limite de duas horas.

Instâncias	**Id.**	**#nós**	**#V_i**	**Tipo**	**CPLEX**		
					Valor	**Lim. Inf.**	**(%)**
i-50-gil262	J_1	262	50	1	135529	135374,68	0,11
10-lin318	J_2	318	10	1	534640	526412,07	1,54
10-pcb442	J_3	442	10	1	547152	536478,33	1,95
C1k.0	J_4	1000	10	2	134025123	131354923,50	1,99
C1k.1	J_5	1000	10	2	130750874	128540131,50	1,69
C1k.2	J_6	1000	10	2	144341485	141501445	1,97
300-6	J_7	300	6	3	8969	8915,18	0,60
400-6	J_8	400	6	3	9117	9021,51	1,05
700-20	J_9	700	20	3	41638	41274,00	0,87
200-4-h	J_{10}	200	4	4	63429	62244,84	1,87
200-4-x1	J_{11}	200	4	4	**60797**	60242,96	**0,91**
600-8-z	J_{12}	600	8	4	132897	127901,75	3,76
600-8-x2	J_{13}	600	8	4	132228	127901,75	3,27
300-5-108	J_{14}	300	5	5	68361	67128,93	1,80
300-20-111	J_{15}	300	20	5	311286	308595,45	0,86
500-15-306	J_{16}	500	15	5	196001	193522,8	1,26
500-25-308	J_{17}	500	25	5	367586	364108,13	0,95
25-eil101	J_{18}	101	25	6	**23671**	23668,63	**0,01**
42-a280	J_{19}	280	42	6	130043	129560,53	0,37
144-rat783	J_{20}	783	144	6	916174	913715,52	0,27
						gap_{li} médio =	1,36

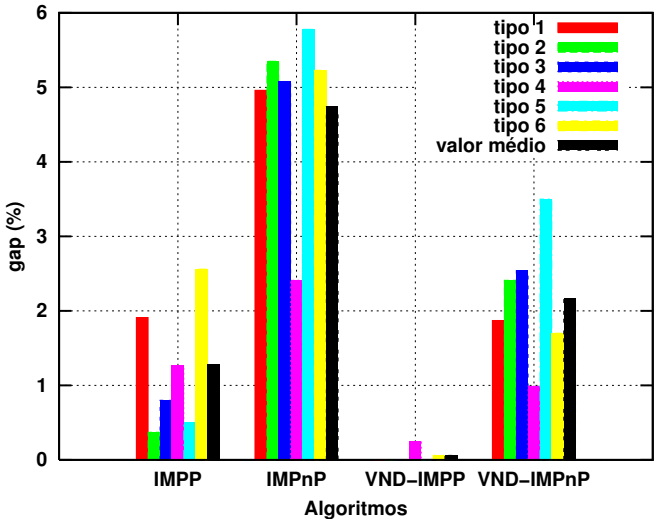

Figura 3. *Gap* médio para cada *tipo* de instância alcançados pelos algoritmos: IMPP, IMPnP, VND-IMPP e VND-IMPnP, sobre os valores médios.

O percentual do gap_h utilizado na Figura 3 é cálculado conforme a Equação 10 descrita abaixo:

$$gap_h = 100 * \left(\frac{vh - mv}{mv} \right), \tag{10}$$

onde, vh é o valor médio encontrado pela heurística e mv é o melhor valor médio alcançado entre as heurísticas IMPP, IMPnP, VND-IMPP e VND-IMPnP.

Na Figura 3 temos, por exemplo, que para cada algoritmo e para as instâncias do *tipo 1* (primeiros dados em barras no gráfico na cor **vermelha**), o gap_h médio dos valores médios do algoritmo IMPP foi igual a 1,91%, de IMPnP com o valor de 4,96%, de VND-IMPP com ***0,00%*** e VND-IMPnP com 1,87%. Observe que o gap_h médio do algoritmo VND-IMPP foram todos iguais a ***zero***, exceto para as instâncias do *tipo 4* e do *tipo 6*, cujos valores foram iguais a 0,24% e 0,06%, respectivamente. Na Figura 3 também são mostrados em barras,

na cor **preta**, os valores médios alcançados pelos algoritmos para todos os tipos de instâncias representando o gap_h médio individual de cada algoritmo.

4.3 Testes dos métodos heurísticos utilizando GRASP e ILS

Os principais parâmetros do GRASP são descritos a seguir. O valor para penalizar os custos das arestas (L) foi de 10*max$\{c_{ij}\}$. A expressão max$\{c_{ij}\}$ significa o maior custo entre todos os custos das arestas (v_i,v_j). O valor de 10*max$\{c_{ij}\}$ adotado para o parâmetro L não ocasionou nenhum problema para os algoritmos gerar as soluções. Todas as soluções geradas foram viáveis. O parâmetro α para gerenciar a LRC na etapa de construção utiliza a versão reativa, onde os seus valores variam no intervalo de [0,1].

O número de iterações do GRASP foi fixado em 200 (critério de parada limitado pelas iterações). A cardinalidade do conjunto de soluções elites, *num_eli*, foi igual a 10 e o valor de *maxprob* com valor igual a 10. Para o critério de parada do GRASP limitado pelo tempo foi estabelecido 720 segundos a cada execução. Todos os valores dos parâmetros do GRASP foram estabelecidos após experimentos preliminares utilizando as instâncias da Tabela 1.

Para o ILS-VND, os parâmetros que foram ajustados após experimentos preliminares são mostrados, a seguir: total de soluções novas geradas iguais a 40 (*iterext*=40), este parâmetro designa o número máximo de iterações externas. O número máximo de iterações internas foi igual a 35 (*maxiter*=35). O valor de *maxprob* foi igual a 10. Para o critério de parada do ILS-VND limitado pelo tempo, também foi estabelecido 720 segundos a cada execução.

Devido à natureza aleatória do GRASP e do ILS foram executadas dez (10) vezes cada heurística aqui proposta para cada instância. Na Tabela 2 é mostrada a comparação entre os métodos G-IMPP e G-IMPnP e entre ILS-VND-P e ILS-VND-nP com critério de parada estabelecido pelo limite de tempo. A primeira coluna da Tabela 2 mostra os identificadores das instâncias da Tabela 1, na segunda coluna o gap_{li}, calculado pela Equação 9, do melhor valor encontrado por G-IMPP e na terceira o gap_{li} do melhor valor encontrado por G-IMPnP.

Tabela 2. Gap_{li} dos valores dos métodos GRASP e ILS com limite de tempo.

Id.	Melhores Valores		Valores Médios		Melhores Valores		Valores Médios	
	G-IMPP	G-IMPnP	G-IMPP	G-IMPnP	ILS-VND-P	ILS-VND-nP	ILS-VND-P	ILS-VND-nP
J_1	**0,07**	0,08	**0,28**	0,32	0,11	**0,10**	**0,15**	**0,15**
J_2	**0,76**	0,81	**1,08**	1,64	**0,73**	**0,73**	**0,82**	0,83
J_3	**0,70**	0,76	**1,08**	1,63	0,56	**0,46**	0,79	**0,65**
J_4	**1,38**	1,47	**1,58**	1,94	**1,30**	1,45	**1,52**	1,81
J_5	1,12	**1,07**	**1,24**	1,6	**0,88**	0,99	**1,06**	1,28
J_6	**1,48**	**1,48**	**1,82**	1,83	1,29	**1,24**	1,46	**1,41**
J_7	**0,52**	0,53	**0,83**	1,32	0,24	**0,21**	0,35	**0,31**
J_8	0,82	**0,79**	**1,39**	1,53	**0,33**	**0,33**	0,52	**0,46**
J_9	**0,42**	0,63	**0,50**	0,82	0,51	**0,43**	0,55	**0,51**
J_{10}	**1,35**	1,36	**2,46**	3,45	**0,89**	**0,89**	**1,16**	1,29
J_{11}	1,04	1,04	**2,97**	3,21	**0,55**	1,13	**0,92**	1,85
J_{12}	1,83	**1,67**	**2,29**	2,63	**1,05**	1,17	**1,30**	1,56
J_{13}	1,83	**1,81**	2,81	**2,70**	1,43	**1,04**	1,79	**1,38**
J_{14}	1,47	**1,33**	**2,18**	2,46	1,03	**1,01**	1,23	**1,18**
J_{15}	**0,43**	0,47	**0,64**	1,02	**0,46**	0,52	**0,54**	0,59
J_{16}	**0,95**	1,01	**1,33**	1,49	**0,80**	0,86	1,02	**0,98**
J_{17}	**0,56**	0,57	**0,74**	0,9	0,51	**0,47**	0,59	**0,54**
J_{18}	**0,01**	0,04	**0,06**	0,36	0,02	0,04	**0,04**	0,09
J_{19}	**0,17**	**0,17**	**0,45**	0,48	**0,12**	**0,12**	**0,13**	0,21
J_{20}	**0,14**	0,15	**0,17**	0,24	**0,16**	**0,16**	**0,16**	0,18
gap_{li} médio	**0,85**	0,86	**1,30**	1,58	***0,65***	0,67	***0,80***	0,86

Nas duas colunas seguintes são mostrados os gap_{li}s dos valores médios alcançados por G-IMPP e G-IMPnP. Na sexta e sétima colunas os gap_{li}s dos melhores valores de ILS-VND-P e ILS-VND-nP, respectivamente. Nas duas últimas colunas os valores médios obtidos por ILS-VND-P e ILS-VND-nP calculados através do gap_{li}. Valores em negrito mostram o melhor desempenho comparando cada método Penalizado e não-Penalizado.

O gap_{li} médio dos melhores valores de G-IMPP ficou em **0,85%** e de G-IMPnP igual a 0,86%. O gap_{li} do CPLEX para estas instâncias com o critério de parada estabelecido de duas horas foi igual 1,36%. G-IMPP

obteve o melhor desempenho obtendo as melhores soluções em quatorze de um total de vinte instâncias, G-IMPnP com sete e CPLEX com duas. O gap_{li} médio dos valores médios de G-IMPP foi igual a **1,30%** e de G-IMPnP em 1,58%. O ILS-VND-P alcançou onze melhores soluções, enquanto que o ILS-VND-nP alcançou treze melhores soluções. O CPLEX alcançou uma única solução melhor quando comparado ao ILS. O gap_{li} médio do ILS-VND-P foi igual a 0,65%, do ILS-VND-nP ficou em 0,67% e do CPLEX em 1,36%. O gap_{li} médio dos valores médios do ILS-VND-P alcançou o valor de 0,80% e do ILS-VND-nP foi igual a 0,86%.

Estes resultados mostram a uma leve superioridade da versão G-IMPP em relação a G-IMPnP e de ILS-VND-P sobre ILS-VND-nP. No entanto, ressalta-se o bom comportamento médios dos quatros métodos heurísticos propostos mostrando que ambos possuem robustez necessária para fins práticos.

A Tabela 3 mostra os resultados obtidos com os algoritmos ILS-VND-P, ILS-VNP-nP e o método exato CPLEX, restritos às instâncias menores sem limitação de tempo para o CPLEX. Os algoritmos ILS foram executados 10 vezes. Na primeira coluna da Tabela 3 encontram-se as instâncias, na segunda o valor obtido pelo CPLEX, na terceira, o tempo demandado pelo CPLEX em segundos t_{CPLEX}(s) e na quarta o gap_{hc} (Equação 11) do melhor valor obtido por ILS-VND-P (%) em relação ao valor obtido pelo CPLEX. Na quinta coluna o tempo médio de execução do ILS-VND-P t_{ILS-P}(s) medido em segundos. Na sexta o gap_{hc} do melhor valor obtido por ILS-VND-nP (%) e sétima coluna o tempo médio de execução do ILS-VND-nP t_{ILS-nP}(s). Na última coluna, é mostrada a perturbação utilizada pelos algoritmos ILS-VND. A razão para executar a perturbação *Perttc* em algumas instâncias deve-se ao fato que estas não têm vértices suficientes nos grupos para realizar a perturbação *Pert*, pois se deve ter oito vértices no mínimo em um grupo para realizar esta perturbação. A *Perttc* foi realizada em 11 instâncias do total de 27.

O percentual do gap_{hc} é calculado da seguinte forma:

$$gap_{hc} = 100 * \left(\frac{vh - vcplex}{vcplex} \right), \tag{11}$$

onde, vh é o melhor valor alcançado pela heurística e $vcplex$ o valor da solução ótima encontrada pelo CPLEX.

Para todas as instâncias de pequeno porte (Tabela 3), o CPLEX encontrou soluções ótimas. ILS-VND-P encontrou soluções ótimas em 21 de um total de 27 instâncias e o gap_{hc} médio total dos valores obtidos pelo

Tabela 3. Comparação de ILS-VND-P e ILS-VNP-nP com o CPLEX para instâncias do *tipo 1* de pequeno porte.

Instâncias	**CPLEX**	t_{CPLEX}**(s)**	**ILS-P(%)**	t_{ILS-P}**(s)**	**ILS-nP(%)**	t_{ILS-nP}**(s)**	**perturbação**
5-eil51	437	12,31	**0,00**	5,20	**0,00**	3,80	Pert
10-eil51	440	74,38	**0,00**	4,40	**0,00**	5,50	Perttc
15-eil51	437	2,04	**0,00**	4,90	**0,00**	5,00	Perttc
5-berlin52	7991	201,80	**0,00**	2,90	0,11	4,50	Pert
10-berlin52	7896	89,17	**0,00**	2,50	0,52	5,10	Pert
15-berlin52	8049	75,93	**0,00**	4,30	**0,00**	4,00	Pert
5-st70	695	13790,11	**0,00**	5,40	**0,00**	6,90	Pert
10-st70	691	4581,00	**0,00**	6,60	0,43	3,70	Pert
15-st70	692	883,50	**0,00**	5,40	0,87	6,40	Pert
5-eil76	559	83,70	**0,00**	6,70	**0,00**	5,10	Pert
10-eil76	561	254,30	0,36	5,50	1,07	8,60	Pert
15-eil76	565	49,66	**0,00**	6,90	0,88	6,80	Pert
5-pr76	108590	99,29	**0,00**	8,20	**0,00**	6,30	Pert
10-pr76	109538	238,13	**0,00**	8,40	**0,00**	8,30	Pert
15-pr76	110678	261,94	0,49	10,00	**0,00**	9,70	Pert
10-rat99	1238	650,67	**0,00**	13,40	**0,00**	12,60	Pert
25-rat99	1269	351,15	**0,00**	23,80	**0,00**	21,40	Perttc
50-rat99	1249	2797,58	**0,00**	23,40	**0,00**	18,60	Perttc
25-kroA100	21917	3513,57	**0,00**	18,40	**0,00**	21,70	Perttc
50-kroA100	21453	947,55	**0,00**	19,90	**0,00**	21,70	Perttc
10-kroB100	22440	4991,44	0,16	9,40	0,91	12,40	Pert
50-kroB100	22355	2579,22	**0,00**	17,70	**0,00**	22,40	Perttc
25-eil101	663	709,45	0,45	21,20	1,06	21,10	Perttc
50-eil101	644	275,33	1,09	21,10	0,16	20,80	Perttc
25-lin105	14438	6224,55	**0,00**	8,10	1,38	16,70	Pert
50-lin105	14379	1577,21	**0,00**	21,50	**0,00**	22,60	Perttc
75-lin105	14521	15886,77	0,15	21,90	**0,00**	23,70	Perttc
val. médios	-	2266,73	***0,10***	***11,37***	0,27	12,05	-

ILS-VND-P em relação ao ótimo foi de 0,10% com tempo médio igual a 11,37s. O ILS-VND-nP conseguiu 17 soluções ótimas. O gap_{hc} médio total dos valores obtidos pelo ILS-VND-nP em relação ao ótimo foi de 0,27% com tempo médio igual a 12,05s. Observa-se que o tempo médio de 12,05s do ILS-VND-nP foi um pouco maior do que o tempo médio de 11,37s do ILS-VND-P. Isto ocorre porque em IMPnP, a Lista de Candidatos que auxilia a elaboração da solução parcial considera em cada etapa todos os vértices pertencentes ao grupo dos vértices contidos na solução parcial. Em oposição, IMPP considera somente os k=10 vértices mais próximos pertencentes ou não ao mesmo grupo dos vértices da solução parcial já construída.

Pelos resultados apresentados nas Tabelas 2 e 3 constata-se que o algoritmo construtivo IMPP quando utilizado para construir soluções para o algoritmo ILS-VND-IMPP permite alcançar resultados melhores, mas bem próximos do algoritmo ILS-VND-IMPnP. Neste sentido, verifica-se que ao tratar o PCVG transformando-o em PCV é uma alternativa eficaz para conseguir bons resultados.

4.4 Comparações entre os métodos heurísticos propostos com um algoritmo genético

Os métodos heurísticos propostos foram comparados com Algoritmo Genético (AG) da literatura (Ding et al., 2007) que resolve o PCVG sem a prefixação da sequência dos grupos visitados. O AG foi implementado neste trabalho pois não foi possível acesso ao código original e seguiu todas as sugestões de parâmetros dado pelos autores.

O AG é denominado de (*Two-Level Genetic Algorithm* – TLGA) (Ding et al., 2007) e constrói soluções para o PCVG em dois níveis, um denominado de nível baixo e outro de nível alto. No nível baixo um método evolutivo utilizando operadores de cruzamento e mutação produz um ciclo T_k, k=1,...,m, em cada grupo utilizando uma população de cromossomos. No nível baixo os operadores de cruzamento e mutação são aplicados somente aos vértices e não operam sobre os grupos. No nível alto uma sequência dos grupos (V_1, V_2, ..., V_m) é gerada aleatoriamente. Em seguida, um caminho P_k sobre o ciclo T_k para cada grupo é encontrado quebrando o ciclo T_k através da escolha de dois vértices aleatórios consecutivos. Um vértice *start* v_{s_k} representa o início do caminho no grupo k e um outro *end* v_{e_k} o fim. O ciclo Hamiltoniano C para o PCVG é encontrado designando cada caminho P_k ao grupo correspondente V_k, iniciando o caminho em cada grupo pelo vértice v_{s_k} e finalizando no vértice v_{e_k}. O algoritmo TLGA gera uma população inicial de cromossomos nos quais cada um representa um ciclo C para o PCVG. O TLGA termina se após um número máximo consecutivos de gerações (*maxger*), a melhor solução da população não é atualizada. Mais detalhes sobre o TLGA mostrado em Ding et al. (2007).

Os resultados computacionais, apresentados a seguir, comparando o TLGA com os métodos heurísticos foram obtidos com o mesmo *software* e *hardware* descritos no início da Seção 4.

Os primeiros testes foram efetuados entre G-IMPP, ILS-VND-P e TLGA, restritos às instâncias menores, mostrados na Tabela 4, vistos que os dois primeiros métodos obtiveram melhor desempenho entre todos os métodos. Nestes testes foram utilizados como critério de parada o número máximo de iterações dos métodos heurísticos. No G-IMPP o número de iterações foi igual a 200 e no ILS-VND-P para o número de iterações internas *maxiter* foi igual a 35 e o parâmetro *iterext* (número máximo de iterações externas) igual a 40. No TLGA, o critério de parada é o número máximo consecutivos de gerações onde a melhor solução da população não é atualizada (*maxger*). O valor de *maxger* foi igual 10. O número de execuções para cada instância foi de 10 para G-IMPP e ILS-VND-P e cinco para TLGA (valor utilizado pelos autores Ding et al. (2007)).

Na primeira coluna da Tabela 4 são mostradas as instâncias, na segunda coluna encontra-se o gap_{hc} do melhor valor obtido por TLGA(%). Na terceira coluna o tempo médio demandado pelo TLGA em segundos t_{TLGA}, na quarta o gap_{hc} do melhor valor obtido por G-P(%) (G-IMPP), na quinta o tempo médio de execução t_{G-P}(s) de G-P medido em segundos. Na sexta e sétima colunas são mostradas o gap_{hc} ILS-P (%) e o tempo médio de execução t_{ILS-P}(s) do ILS-P, respectivamente. O percentual do gap_{hc} é calculado de acordo a Equação 11, onde os valores das soluções ótimas são apresentados na Tabela 3.

Observa-se que na Tabela 4, o TLGA produz soluções de baixa qualidade quando comparado ao G-IMPP e ILS-VND-P. O gap_{hc} médio das soluções produzidas pelo TLGA ficou em 12,29%, enquanto G-P alcançou 0,21% e ILS-VND-P com 0,10%. O TLGA nunca alcançou uma solução ótima. O tempo médio total avaliado sobre todas as instâncias do TLGA foi melhor do que G-IMPP e ILS-VND-P alcançando 1,8s, G-IMPP ficou em 6,05s e ILS-VND-P com 11,37s. Mesmo que o tempo médio do TLGA seja menor, a opção pela heurística G-IMPP torna-se mais promissora, porque aglutina pequeno esforço computacional com soluções de alta qualidade.

Nos resultados apresentados até o momento, onde o critério de parada é o número de iterações, observa-se que G-IMPP e ILS-VND-P obtiveram os melhores resultados, mas exigiram tempos computacionais maiores que TLGA, mesmo para instâncias de pequeno porte. Novos testes computacionais para instâncias maiores foram executados.

O critério de parada estabelecido para os três métodos heurísticos neste novo teste foi definido por um tempo máximo de 720 segundos a cada execução, porque, para cada instância, os métodos são executados

Tabela 4. Comparação do TLGA com G-IMPP e ILS-VND-P utilizando como critério de parada estabelecido pelo número de iterações.

Instâncias	**TLGA (%)**	t_{TLGA}**(s)**	**G-P(%)**	t_{G-P}**(s)**	**ILS-P(%)**	t_{ILS-P}**(s)**
5-eil51	8,47	**0,4**	**0,00**	1,50	**0,00**	5,20
10-eil51	2,73	**0,4**	**0,00**	1,30	**0,00**	4,40
15-eil51	7,78	**0,4**	**0,00**	1,40	**0,00**	4,90
5-berlin52	1,44	**0,6**	**0,00**	1,80	**0,00**	2,90
10-berlin52	10,18	**0,4**	**0,00**	1,60	**0,00**	2,50
15-berlin52	14,69	**0,4**	**0,00**	1,60	**0,00**	4,30
5-st70	0,86	**1,6**	**0,00**	3,40	**0,00**	5,40
10-st70	1,88	**1,0**	**0,00**	2,50	**0,00**	6,60
15-st70	7,23	**0,8**	**0,00**	2,90	**0,00**	5,40
5-eil76	3,94	**1,8**	0,54	3,80	**0,00**	6,70
10-eil76	9,45	**1,2**	0,71	3,20	**0,36**	5,50
15-eil76	2,48	**1,2**	0,53	3,60	**0,00**	6,90
5-pr76	1,78	**2,0**	**0,00**	5,50	**0,00**	8,20
10-pr76	1,02	**1,2**	**0,00**	4,00	**0,00**	8,40
15-pr76	5,95	**1,2**	**0,15**	4,40	0,49	10,00
10-rat99	6,70	**3,4**	**0,00**	9,70	**0,00**	13,40
25-rat99	23,48	**2,4**	**0,00**	9,80	**0,00**	23,80
50-rat99	37,15	**2,4**	0,80	10,50	**0,00**	23,40
25-kroA100	5,69	**2,2**	**0,00**	8,60	**0,00**	18,40
50-kroA100	22,23	**2,8**	**0,00**	9,60	**0,00**	19,90
10-kroB100	2,49	**3,8**	**0,00**	9,30	0,16	9,40
50-kroB100	25,36	**2,2**	0,54	9,50	**0,00**	17,70
25-eil101	6,33	**2,2**	1,21	7,80	**0,45**	21,20
50-eil101	20,34	**2,2**	**1,09**	9,10	**1,09**	21,10
25-lin105	19,39	**2,0**	**0,00**	10,50	**0,00**	8,10
50-lin105	26,29	**3,2**	**0,00**	12,40	**0,00**	21,50
75-lin105	56,62	**4,6**	0,23	14,10	**0,15**	21,90
valores médios	12,29	**1,8**	0,21	6,05	**0,10**	11,37

10 vezes. Mesmo para o TLGA que no algoritmo original executa cinco vezes, nessa nova bateria de testes, o algoritmo foi executado 10 vezes. A melhor solução encontrada e a solução média nestas execuções são retornadas. Somente a execução da instância J_{18} não foi limitada por duas horas, devido à solução ótima ser encontrada antes pelo CPLEX.

Os melhores valores e valores médios obtidos pelas heurísticas com o critério de parada limitado pelo tempo são mostrados na Tabela 5, onde na primeira coluna apresentam-se os identificadores das instâncias. Na segunda coluna o gap_{li} alcançado pelo TLGA, onde este *gap* é calculado pela Equação 9. Na terceira coluna o gap_{li} alcançado por G-IMPP e na quarta coluna o gap_{li} de ILS-VND-P. Nas três últimas colunas temos gap_{li} para os valores médios.

Na Tabela 5 foi observado que mesmo estabelecendo tempos limites iguais para os três métodos heurísticos, o algoritmo TLGA não consegue melhorar a qualidade da solução para o PCVG. O TLGA continua produzindo soluções de baixa qualidade comparadas ao G-IMPP e ILS-VND-P, onde o gap_{li} médio ficou em 9,31%, enquanto G-IMPP alcançou 0,85% e ILS-VND-P com 0,65%. Para algumas instâncias, o algoritmo TLGA obteve gap_{li} próximos de G-IMPP e ILS-VND-P. Por exemplo, para a instância J_{18} o gap_{li} de TLGA foi de 0,05%, enquanto G-IMPP e ILS-VND-P obtiveram 0,01% e 0,02%, respectivamente. Para as instâncias J_1, J_{15} e J_{19}, o gap_{li} de TLGA alcançou 0,12%, 0,56% e *0,14%*, respectivamente, enquanto que G-IMPP obteve para J_1 o valor de 0,07%, para J_{15} igual a 0,43% e para J_{19} o valor 0,17%. O valor em itálico para a instância J_{19} mostra que TLGA foi melhor do que G-IMPP. Para ILS-VND-P o gap_{li} da instância J_1 foi igual a 0,11%, para J_{15} igual a 0,46% e J_{19} em 0,12%. As melhores soluções sempre são encontradas por G-IMPP ou ILS-VND-P. O gap_{li} médio dos valores médios de TLGA ficou em 12,30%, enquanto G-IMPP alcançou 1,30% e ILS-VND-P com 0,80%. Em algumas instâncias os gap_{li}s dos valores médios dos custos das soluções encontradas pelo algoritmo TLGA são menores do que 1%. Por exemplo, o TLGA obteve gap_{li} dos valores médios para as instâncias J_1, J_{18} e J_{19}, iguais a 0,32%, 0,33% e 0,82%, respectivamente.

Tabela 5. Gap_{li} dos melhores valores e valores médios de TLGA, G-IMPP e ILS-VND-P com limite de tempo.

Id.	Melhores Valores			Valores Médios		
	TLGA (%)	**G-P(%)**	**ILS-P(%)**	**TLGA (%)**	**G-P(%)**	**ILS-P(%)**
J_1	0,12	**0,07**	0,11	0,32	0,28	**0,15**
J_2	1,08	0,76	**0,73**	6,27	1,08	**0,82**
J_3	9,07	0,70	**0,56**	16,86	1,08	**0,79**
J_4	26,32	1,38	**1,30**	27,26	1,58	**1,52**
J_5	25,84	1,12	**0,88**	26,85	1,24	**1,06**
J_6	23,61	1,48	**1,29**	24,72	1,82	**1,46**
J_7	0,81	0,52	**0,24**	4,72	0,83	**0,35**
J_8	4,71	0,82	**0,33**	15,01	1,39	**0,52**
J_9	6,99	**0,42**	0,51	8,22	**0,50**	0,55
J_{10}	1,72	1,35	**0,89**	4,34	2,46	**1,16**
J_{11}	1,46	1,04	**0,55**	4,01	2,97	**0,92**
J_{12}	33,71	1,83	**1,05**	38,66	2,29	**1,30**
J_{13}	33,34	1,83	**1,43**	38,57	2,81	**1,79**
J_{14}	1,61	1,47	**1,03**	6,60	2,18	**1,23**
J_{15}	0,56	**0,43**	0,46	1,84	0,64	**0,54**
J_{16}	7,93	0,95	**0,80**	11,15	1,33	**1,02**
J_{17}	4,39	0,56	**0,51**	6,08	0,74	**0,59**
J_{18}	0,05	**0,01**	0,02	0,33	0,06	**0,04**
J_{19}	*0,14*	0,17	**0,12**	0,82	0,45	**0,13**
J_{20}	2,70	**0,14**	0,16	3,28	0,17	**0,16**
gap_{li} médio	9,31	0,85	**0,65**	12,30	1,30	**0,80**

4.5 Análise dos métodos penalizados e não-penalizados

Para analisar os resultados dos métodos heurísticos que penalizam ou não, as arestas intergrupos, são classificados como métodos Penalizados (P) aqueles que utilizam a penalização intergrupos e de métodos não-Penalizados (nP) os que não penalizam. Dentre os métodos Penalizados são destacados: IMPP, VND-P, G-IMPP e ILS-VND-P. Para os métodos não-Penalizados são apresentados: IMPnP, VND-nP, G-IMPnP e ILS-VND-nP.

A Tabela 6 mostra para cada tipo de instância uma comparação do desempenho de cada método (Penalizado ou não-Penalizado). Foi realizado um total de 173 experimentos. Para cada experimento realizado com uma instância e os dois métodos, foram atribuídos um valor igual a um (1) se um método obteve melhor desempenho, zero (0) se obteve pior desempenho e meio (1/2) a cada método se o desempenho for igual em ambos. Para os métodos construtivos, VND, GRASP e ILS-VND é considerado de melhor desempenho, o *gap* da melhor solução encontrada por estes métodos.

Tabela 6. Desempenho (%) dos métodos penalizados e não-penalizados para cada *tipo* de instância.

Tipo	IMPP	IMPnP	VND-P	VND-nP	G-IMPP	G-IMPnP	ILS-P	ILS-nP
1	**77,8**	22,2	33,3	**66,7**	**100,0**	0,0	**63,0**	37,0
2	**100,0**	0,0	**80,0**	20,0	50,0	50,0	**66,7**	33,3
3	**90,0**	10,0	**55,0**	45,0	**66,7**	33,3	16,7	**83,3**
4	40,0	**60,0**	40,0	**60,0**	37,5	**62,5**	**62,5**	37,5
5	**90,0**	10,0	50,0	50,0	**75,0**	25,0	50,0	50,0
6	**77,8**	22,2	**66,7**	33,3	**83,3**	16,7	**66,7**	33,3

Na primeira coluna da Tabela 6 é apresentado o *tipo* de instância analisada, na segunda coluna a soma dos valores (em porcentagem) obtidos pelo algoritmo construtivo IMPP para todas as instâncias deste tipo, na terceira a soma dos valores para o algoritmo construtivo IMPnP, na quarta e quinta colunas as somas dos valores dos método VND com IMPP (VND-P) e do VND com IMPnP (VND-nP), respectivamente. Na sexta coluna a soma dos valores obtidos para G-IMPP, na sétima a soma obtida pelo G-IMPnP e nas últimas colunas os valores de somas do ILS-VND-IMPP (ILS-P) e ILS-VND-IMPnP (ILS-nP), respectivamente.

São observados na Tabela 6 para as instâncias do *tipo 2*, *5* e *6* que os Métodos Penalizados sempre ganham. Na maioria dos casos, as instâncias do *tipo 3* apresentam porcentagens melhores com os Métodos Penalizados, exceto ao utilizar o algoritmo ILS-nP, já as instâncias do *tipo 4* com os não-Penalizados. Para as instâncias

do *tipo 1*, em algumas situações, são obtidas de melhor desempenho para alguns Métodos Penalizados e em outras situações aos Métodos não-Penalizados.

5. Discussão e Conclusões

Neste capítulo foram propostos diversos métodos heurísticos para resolver o PCVG. Todos os métodos heurísticos encontram soluções de boa qualidade num tempo computacional baixo. Nas comparações dos métodos heurísticos com o CPLEX foi verificado que na maior parte das instâncias de pequeno porte, estes métodos alcançaram soluções ótimas. Ao estabelecer limites de tempo para o CPLEX, os valores dos gap_{li}s alcançados pelos métodos heurísticos (G-IMPP, G-IMPnP, ILS-VND-P e ILS-VND-nP) para as instâncias de porte maior, foram melhores do que os obtidos pela formulação exata, em sua grande maioria. Ao comparar os métodos heurísticos com um Algoritmo Genético (AG) da literatura, verifica-se que tanto o G-IMPP quanto o ILS-VND-P obtiveram melhor desempenho do que o AG.

Ao analisar os Métodos Penalizados (IMPP, VND-P, G-IMPP, ILS-VND-P) com os Métodos não-Penalizados (IMPnP, VND-nP, G-IMPnP e ILS-VND-nP) verificou-se que para as instâncias do *tipo 2*, *5* e *6* foi obtido melhor desempenho quando as arestas intergrupos são penalizadas.

Foi observado também, nos resultados dos Métodos Penalizados e não-Penalizados, que para diversas instâncias, alguns métodos heurísticos obtêm melhor desempenho quando a estratégia que *não penaliza as arestas intergrupos* é aplicada. Assim, pode-se afirmar que a penalização não é uma estratégia boa, principalmente as do *tipo 1*, *tipo 3* e do *tipo 4* que necessitam utilizar algoritmos não-Penalizados. As características das instâncias do *tipo 1*, *3* e *4* são: *primeiro*, os vértices não são dispersos e *segundo*, em cada grupo os vértices são agrupados com melhor distribuição geométrica em torno de um centro.

Trabalhos futuros incluem a adaptação dos módulos deste trabalho como RC e VND para outras metaheurísticas tais como os Algoritmos Evolutivos e *Variable Neighborhood Search*, bem como outras formas híbridas envolvendo GRASP, RC, VND e ILS combinando-os em diversas estratégias.

Novas alternativas poderão ser estudadas para utilizar novos algoritmos construtivos com penalização nas arestas e sem penalização, ajustando-os com as buscas locais (2-Optimal, 3-Optimal e VND) e novas formas de Reconexão de Caminhos para formar módulos que se adaptem aos métodos heurísticos baseados no GRASP e ILS a serem implementados.

6. Agradecimentos

Este trabalho tem o apoio parcial da CAPES (Programa PIQDTec).

Referências

Anily, S.; Bramel, J. & Hertz, A., A 5/3-approximation algorithm for the clustered traveling salesman tour and path problems. *Operations Research Letters*, 24(1-2):29–35, 1999.

Arkin, E.M.; Hassin, R. & Klein, L., Restricted delivery problems on a network. *Networks*, 29(4):205–216, 1997.

Chisman, J.A., The clustered traveling salesman problem. *Computers & Operations Research*, 2(2):115–119, 1975.

Dantzig, G.; Fulkerson, R. & Johnson, S., Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2:393–410, 1954.

Ding, C.; Cheng, Y. & He, M., Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale TSPs. *Tsinghua Science and Technology*, 12(4):459–465, 2007.

de Assumpção Drummond, L.M.; Vianna, L.S.; da Silva, M.B. & Ochi, L.S., Distributed parallel metaheuristics based on GRASP and VNS for solving the traveling purchaser problem. In: *Proceedings of the Ninth International Conference on Parallel and Distributed Systems*. Piscataway, USA: IEEE Press, v. 1, p. 257–263, 2002.

Garey, M.R. & Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, USA: W. H. Freeman, 1979.

Gendreau, M.; Laporte, G. & Potvin, J.Y., *Heuristics for the Clustered Traveling Salesman Problem*. Technical Report CRT-94-54, Centre de Recherche sur les Transports, Université de Montréal, Montréal, Canada, 1994.

Ghaziri, H. & Osman, I.H., A neural network for the traveling salesman problem with backhauls. *Computers & Industrial Engineering*, 44(2):267–281, 2003.

Glover, F., Tabu search and adaptive memory programing - advances, applications and challenges. In: Barr, R.S.; Helgason, R.V. & Kennington, J.L. (Eds.), *Interfaces in Computer Science and Operations Research*. Dordrecht, Netherlands: Kluwer Academic Publishers, p. 1–75, 1996.

Gomes, L.M.; Diniz, V.B. & Martinhon, C.A., A hybrid GRASP+VNS metaheuristic for the prize collecting traveling salesman problem. In: *Anais do XXXII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*. N. 32, p. 1656–1666, 2000.

Guttmann-Beck, N.; Hassin, R.; Khuller, S. & Raghavachari, B., Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000.

Hansen, P. & Mladenović, N., Variable neighborhood search. In: Glover, F.W. & Kochenberger, G.A. (Eds.), *Handbook of Metaheuristics*. Boston, USA: Kluwer Academic Publishers, p. 145–184, 2003.

Hernández-Pérez, H.; Rodríguez-Martín, I. & Salazar-González, J.J., A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36(5):1639–1645, 2009.

Jongens, K. & Volgenant, T., The symmetric clustered traveling salesman problem. *European Journal of Operations Research*, 19(1):68–75, 1985.

Laguna, M. & Martí, R., GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

Laporte, G. & Palekar, U., Some applications of the clustered travelling salesman problem. *Journal of the Operational Research Society*, 53(9):972–976, 2002.

Laporte, G.; Potvin, J.Y. & Quilleret, F., A tabu search heuristic using genetic diversification for the clustered traveling salesman problem. *Journal of Heuristics*, 2(3):187–200, 1996.

Lawler, E.L.; Lenstra, J.K.; Kan, A.H.G.R. & Shmoys, D.B., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. New York, USA: J. Wiley & Sons, 1985.

Lokin, F.C.J., Procedures for travelling salesman problems with additional constraints. *European Journal of Operations Research*, 3(2):135–141, 1979.

Lourenço, H.R.; Martin, O.C. & Stützle, T., Iterated local search. In: Glover, F. & Kochenberger, G. (Eds.), *Handbook of Metaheuristics*. Boston, USA: Kluwer Academic Publishers, p. 321–353, 2002.

Martin, O.; Otto, S.W. & Felten, E.W., Large-step markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224, 1992.

Mestria, M., Proposta e avaliação de heurísticas GRASP para o problema do caixeiro viajante com grupamentos. In: *Anais do X Congresso Brasileiro de Inteligência Computacional*. Fortaleza, CE: SBRN, v. 1, 2011.

Miller, C.E.; Tucker, A.W. & Zemlin, R.A., Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

Potvin, J.Y. & Guertin, F., *A Genetic Algorithm for the Clustered Traveling Salesman Problem with an A Priori Order on the Clusters*. Technical Report CRT-95-06, Centre de Recherche sur les Transports, Université de Montréal, Montréal, Canada, 1995.

Potvin, J.Y. & Guertin, F., The clustered traveling salesman problem: A genetic approach. In: Osman, I.H. & Kelly, J. (Eds.), *Meta-heuristics: Theory & Applications*. Norwell, USA: Kluwer Academic Plublishers, p. 619–631, 1996.

Reinelt, G., *The Traveling Salesman: Computational Solutions for TSP Applications*. v. 840 de *Lecture Notes in Computer Science*. Heidelberg, Germany: Springer-Verlag, 1994.

Resende, M.G.C.; Martí, R.; Gallego, M. & Duarte, A., GRASP and path relinking for the max-min diversity problem. *Computers & Operations Research*, 37(3):498–508, 2010.

Resende, M.G.C. & Ribeiro, C.C., Greedy randomized adaptive search procedures. In: Glover, F. & Kochenberger, G. (Eds.), *Handbook of Metaheuristics*. Dordrecht, Netherlands: Kluwer Academic Publishers, p. 219–249, 2002.

Weintraub, A.; Aboud, J.; Fernandez, C.; Laporte, G. & Ramirez, E., An emergency vehicle dispatching system for an electric utility in Chile. *Journal of the Operational Research Society*, 50:690–696, 1999.

Notas Biográficas

Mário Mestria é graduado e mestre em Engenharia Elétrica (UFES, 1991 e 1995, respectivamente) e doutor em Computação (IC/UFF, 2011). Atualmente é professor da coordenadoria do curso de Eletrotécnica, da Engenharia Elétrica e da Pós-Graduação em Engenharia Elétrica, câmpus Vitória, do Instituto Federal de Educação, Ciência e Teconologia do Espírito Santo. Atua na área de computação e tem interesse em meta-heurísticas, otimização combinatória, aplicações modeladas num PCVG e *layout* de sistemas de manufaturas. Trabalha nas áreas acadêmicas de linguagem de programação, eletricidade, eletrônica, robótica assistiva e educacional.