Capítulo 11

Um Algoritmo Heurístico Baseado em *Iterated Local Search* para Problemas de Roteamento de Veículos

Anand Subramanian, Puca Huachi Vaz Penna*,
Luiz Satoru Ochi e Marcone Jamilson Freitas Souza

Resumo: Este capítulo apresenta um algoritmo heurístico, baseado na meta-heurística *Iterated Local Search*, para resolver diversas variantes do Problema de Roteamento de Veículos. São descritos os procedimentos de construção das soluções iniciais, as estruturas de vizinhança para explorar o espaço de soluções, o esquema de busca local, assim como os mecanismos de perturbação. Foram realizados experimentos computacionais em conjuntos consagrados de problemas-teste da literatura referente às variantes consideradas. Os resultados mostraram que o algoritmo desenvolvido produz soluções finais de alta qualidade e baixa variabilidade. Além disto, considerando o conjunto de problemas-teste usados, ele detém mais de 80% dos melhores resultados da literatura.

Palavras-chave: Problemas de Roteamento de Veículos, *Iterated Local Search*, Meta-heurística.

Abstract: *This work presents a heuristic algorithm, based on the Iterated Local Search (ILS) metaheuristic, for solving different variants of the Vehicle Routing Problem. The construction procedures, the neighborhood structures to explore the solution space, the local search scheme and the perturbation mechanisms are described. Computational experiments were carried out in well-known benchmark instances of the variants considered. The results showed that the developed algorithm produces high quality solutions with small variability. In addition, when considering the set of test-problems used, the proposed algorithm holds more than 80% of the best known results of the literature.*

Keywords: *Vehicle Routing Problem, Iterated Local Search, Metaheuristic.*

1. Contextualização

O Problema de Roteamento de Veículos - PRV (*Vehicle Routing Problem* - VRP) teve sua origem associada ao trabalho desenvolvido por Dantzig & Ramser (1959). Desde então, tem sido, particularmente nas últimas décadas, um dos problemas mais abordados nas áreas de Otimização Combinatória e Pesquisa Operacional. Isso se deve, em parte, ao grande desenvolvimento de métodos de resolução e da enorme variedade de aplicações existentes.

Outro aspecto que tem contribuído para este sucesso é a eficiência destes métodos no sentido operacional, ou seja, as técnicas desenvolvidas têm se mostrado eficientes quando implementadas em situações reais em diferentes empresas da área de transporte.

A importância no desenvolvimento de Sistemas Automatizados, para estas empresas de transporte, se deve também ao fato de eles propiciarem economias médias entre 5% a 10% nos custos finais de transporte (Toth & Vigo, 2002). Em muitas empresas de médio e grande porte, esses percentuais representam valores altamente significativos.

Atualmente, na literatura, existe uma gama enorme de variantes do problema de roteamento que consideram diversas restrições espaciais e/ou temporais encontradas em aplicações reais. Essas variantes têm em comum uma elevada complexidade computacional. Mesmo observando um enorme desenvolvimento de métodos exatos de otimização para a resolução do PRV e suas variantes, a maioria dos problemas, caracterizados por elevadas dimensões, ainda tendem a serem resolvidos por meio de métodos heurísticos.

Na literatura são apresentados diversos algoritmos heurísticos, cada qual destinado, em geral, à resolução de uma única variante do PRV. Neste capítulo apresenta-se um algoritmo unificado, baseado na meta-heurística

*Autor para contato: ppenna@ic.uff.br

Iterated Local Search – ILS (Lourenço et al., 2003), capaz de resolver eficientemente vários problemas de roteamento. Versões parciais do algoritmo proposto foram publicados nos seguintes trabalhos: Subramanian et al. (2010), Souza et al. (2010b), Penna et al. (2011), Silva et al. (2012) e Subramanian & Battarra (2013).

O restante deste capítulo está organizado como segue. Na Seção 2 são descritos os problemas abordados. Na Seção 3 é apresentada a fundamentação teórica da meta-heurística *Iterated Local Search* (ILS). Na Seção 4 é descrito o algoritmo heurístico, baseado em ILS, proposto para resolver os problemas abordados. Na Seção 5 são mostrados os resultados da aplicação do algoritmo proposto a instâncias desses problemas encontradas na literatura. A Seção 6 apresenta as considerações finais.

2. Descrição dos Problemas

Para a definição das diversas variantes dos PRVs, seja $D = \{1, \ldots, d\}$ o conjunto de depósitos, $V' = \{1, \ldots, n\}$ o conjunto de clientes, $G = (V, E)$ um grafo completo com $V = D \cup V'$ representando o conjunto de $n + d$ vértices, $E = \{(i, j) : i, j \in V, i \neq j\}$ o conjunto de arestas, q_i a demanda do cliente $i \in V'$, p_i a quantidade de produtos a serem coletados no cliente $i \in V'$, $T = \{1, \ldots, t\}$ o conjunto de tipos de veículos, m_k o número de veículos do tipo $k \in T$, Q_k a capacidade do veículo do tipo k, f_k o custo fixo do veículo do tipo k, g_k o custo variável do veículo do tipo k, l_{ij} o comprimento da aresta $(i, j) \in E$, $c^k_{ij} = f_k + g_k \times l_{ij}$ o custo de percorrer o caminho $(i, j) \in E$ com o veículo do tipo $k \in T$. Assume-se que a demanda dos depósitos é nula e a distância $l_{ij} = \infty$, $\forall i, j \in D$.

2.1 Problema de roteamento de veículos capacitado

Considerada a versão clássica dos PRVs, no Problema de Roteamento de Veículos Capacitado (PRVC) tem-se um único depósito, um único tipo de veículo (sem custo a ele associado, sem perda de generalidade), demanda de somente coleta ou somente entrega para todos os clientes e matriz de custos simétrica, isto é, $|D| = 1$, $|T| = 1$, $f_k = 0$, $g_k = 1$, $p_i = 0$, $c^k_{ij} = c^k_{ji}$. Por envolver um único tipo de veículo, diz-se que a frota é homogênea e composta por m veículos. Para simplicidade de notação o depósito será referenciado pelo índice 0, c_{ij} representará o custo de transporte entre o depósito e os clientes ou entre os clientes e Q denotará a capacidade dos veículos da frota homogênea.

O PRVC consiste em determinar um conjunto R de rotas para os veículos, de tal modo que: (i) toda rota $r \in R$ começa e termina no depósito, isto é, dada a sequência r de visitas, em que $r = \{i_1, i_2, \ldots, i_{|r|}\}$, tem-se $i_1 = i_{|r|} = 0$ e $\{i_2, \ldots, i_{|r|-1}\} \subseteq V'$; (ii) a demanda de todos os clientes deve ser atendida pelos veículos; (iii) cada cliente é atendido por um único veículo; (iv) a capacidade de cada veículo não pode ser violada, isto é, $\sum_{h=2}^{|r|-1} q_{i_h} \leqslant Q$; (v) a soma total dos custos das operações de transporte deve ser minimizada.

Seja x_{ij} uma variável inteira que armazena o número de vezes que a aresta $\{i, j\} \in E$ aparece em uma rota. Tal variável assumirá valor 2 para rotas que contém apenas um cliente. Dado um conjunto $S \subseteq V'$, defina $q(S)$ como sendo a soma das demandas de todos os clientes em S e $e(S) = \lceil q(S)/Q \rceil$. Finalmente, considere $\bar{S}$ como sendo o conjunto complementar de S, incluindo o depósito $\{0\}$. O PRVC pode ser modelado da seguinte forma:

$$\min \sum_{i \in V} \sum_{j \in V, j > i} c_{ij} x_{ij} \tag{1}$$

$$\text{s.a:} \sum_{i \in V, i < k} x_{ik} + \sum_{j \in V, j > k} x_{kj} = 2 \qquad \forall k \in V' \tag{2}$$

$$\sum_{j \in V'} x_{0j} = 2m \tag{3}$$

$$\sum_{i \in S} \sum_{j \in \bar{S}, i < j} x_{ij} + \sum_{i \in \bar{S}} \sum_{j \in S, i < j} x_{ij} \geq 2e(S) \qquad \forall S \subseteq V' \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \{i, j\} \in E, i > 0 \tag{5}$$

$$x_{ij} \in \{0, 1, 2\} \qquad \forall \{0, j\} \in E. \tag{6}$$

A função objetivo (1) minimiza os custos de viagem. As restrições (2) asseguram que cada cliente possui duas arestas incidentes. As restrições (3) determinam que o número de arestas incidentes no depósito deve ser igual ao dobro da quantidade de veículos m. As restrições (4) impedem a formação de subrotas. Por fim, as restrições (5-6) definem o domínio das variáveis.

Vale ressaltar que as formulações matemáticas das variantes descritas a seguir podem ser derivadas do modelo apresentado nesta seção.

2.2 Problema de roteamento de veículos capacitado assimétrico

O Problema de Roteamento de Veículos Capacitado Assimétrico (PRVCA) difere do PRVC com relação à matriz de custos, que neste caso é assimétrica, ou seja, $\exists\, i, j \in V' \mid c_{ij} \neq c_{ji}$.

2.3 Problema de roteamento de veículos aberto

O Problema de Roteamento de Veículos Aberto (PRVA) é uma variante do PRVC em que o veículo não necessita retornar ao depósito, o que equivale a adotar o custo de retorno do veículo ao depósito nulo. A maioria dos autores da literatura assume que o objetivo primário é minimizar o número de veículos.

2.4 Problema de roteamento de veículos com coleta e entrega simultânea

O Problema de Roteamento de Veículos com Coleta e Entrega Simultânea (PRVCES) é uma variante do PRVC em que cada cliente $i \in V'$ requer a coleta e a entrega simultânea de p_i e q_i unidades de um produto, respectivamente.

2.5 Problema de roteamento de veículos com coleta e entrega mista

Ao contrário do PRVCES, no Problema de Roteamento de Veículos com Coleta e Entrega Mista (PRVCEM) cada cliente tem uma demanda de somente coleta ou somente entrega.

2.6 Problema de roteamento de veículos com múltiplos depósitos

O Problema de Roteamento de Veículos com Múltiplos Depósitos (PRVMD) envolve um conjunto V' de clientes, em que todos demandam operações de somente entrega ou somente coleta de unidades de um produto, um conjunto D de depósitos e uma frota homogênea de veículos para cada depósito. O problema consiste em estabelecer um conjunto de rotas para os veículos, de forma a atender a demanda dos clientes com o menor custo de transporte. Neste problema assume-se que cada veículo está sediado em um dado depósito, previamente conhecido. Desta forma, a rota de cada veículo deve iniciar e terminar no mesmo depósito.

2.7 Problema de roteamento de veículos com múltiplos depósitos com coleta e entrega mista

O Problema de Roteamento de Veículos com Múltiplos Depósitos com Coleta e Entrega Mista (PRVMDCEM) é uma variante do PRVMD em que cada cliente requer a demanda de somente coleta ou somente entrega, tal como no PRVCEM.

2.8 Problema de roteamento de veículos com frota heterogênea

O Problema de Roteamento de Veículos com Frota Heterogênea (PRVFH) envolve um conjunto V' de clientes, em que todos demandam operações de somente entrega ou somente coleta de unidades de um produto, um único depósito ($|D| = 1$) e um conjunto T de tipos diferentes de veículos. Para cada tipo $k \in T$ existem m_k veículos disponíveis no depósito, de capacidade Q_k, custo fixo f_k e custo variável g_k. Há duas variantes consideradas para este problema. A primeira considera conhecido o número de veículos disponíveis e, na segunda, este número é ilimitado. Em ambas, procura-se estabelecer um conjunto de rotas para os veículos, de forma a atender a demanda dos clientes no menor custo de transporte. Observa-se que na segunda variante, além do roteamento, é feito o dimensionamento da frota.

3. A Meta-Heurística *Iterated Local Search*

A meta-heurística *Iterated Local Search* – ILS (Lourenço et al., 2003) é um método de busca local que explora o espaço de soluções por meio de perturbações em ótimos locais gerados durante a busca.

Para descrever o ILS, considere um ótimo local obtido por um método de busca local. Em seguida, uma solução intermediária é obtida por meio da aplicação de uma perturbação na solução ótima local previamente visitada. A ideia principal do ILS está em utilizar ótimos locais como novas soluções de partida, ao invés de começar de uma solução completamente nova. Desta forma, o ILS concentra sua busca em um conjunto reduzido de soluções, no caso, apenas ótimos locais, em vez de considerar todo o espaço de busca.

Para aplicar um algoritmo ILS, quatro elementos devem ser especificados: (i) Procedimento `GeraSoluçãoInicial()`, que constrói uma solução inicial para o problema; (ii) Procedimento `BuscaLocal()`, que recebe uma solução e retorna um ótimo local; (iii) Procedimento `Perturbação()`, que modifica, ou seja, perturba a solução corrente gerando, assim, uma solução intermediária e (iv) Procedimento `CritérioAceitação()`, que decide a partir de qual solução será aplicada a próxima perturbação. O Algoritmo 1 descreve como combinar estes elementos para utilizar a meta-heurística ILS.

Algoritmo 1: ILS

```
1 Início
2     s_0 ← GeraSoluçãoInicial()
3     s* ← BuscaLocal(s_0)
4     enquanto (os critérios de parada não estiverem satisfeitos) faça
5         s' ← Perturbação(s*, histórico)
6         s*' ← BuscaLocal(s')
7         s* ← CritérioAceitação(s*, s*')
8 fim
```

Pelo Algoritmo 1 gera-se, na linha 2, uma solução inicial s_0. Essa solução é, a seguir, na linha 3, refinada por um método de busca local, conduzindo a um ótimo local s^*. O procedimento iterativo do método ocorre nas linhas 5 a 7. Na linha 5, o ótimo local s^* passa por uma perturbação gerando uma solução intermediária s', a qual, por sua vez, passa por um processo de refinamento (linha 6), resultando em um novo ótimo local $s^{*\prime}$. Na linha 7, verifica-se de qual solução a busca prosseguirá, se do ótimo local s^* corrente ou do novo ótimo local $s^{*\prime}$. O critério de aceitação mais utilizado é partir do ótimo local de melhor qualidade, ou seja, $s^{*\prime}$ é aceita como nova solução de partida se $f(s^{*\prime}) < f(s^*)$ para um problema de minimização, sendo $f(s)$ o custo associado à solução s.

Para obter sucesso com o ILS, uma atenção especial deve ser dada na escolha do método de busca local, das perturbações e do critério de aceitação. Em princípio, qualquer método de busca local pode ser utilizado. Entretanto, o desempenho do ILS, em termos de qualidade da solução e esforço computacional, dependem fortemente do método adotado. A perturbação por sua vez, consiste em um conjunto de modificações feitas no ótimo local para gerar soluções progressivamente mais "distantes" desse ótimo local. A ideia principal é aumentar o nível de perturbação se a busca local não for bem sucedida, isto é, não estiver gerando ótimos locais melhores. A variável `histórico` controla o nível de perturbação e estabelece um balanço entre intensificação e diversificação. Quando o nível de perturbação é baixo, há uma exploração mais efetiva da região do espaço de busca em que se encontra o ótimo local corrente, isto é, aplica-se a intensificação. Por outro lado, à medida que esse nível de perturbação é aumentado a busca se direciona para outras regiões do espaço de soluções, ou seja, aplica-se a estratégia de diversificação.

4. Descrição do Algoritmo Proposto

Esta seção descreve o algoritmo desenvolvido para resolver uma ampla classe de Problemas de Roteamento de Veículos. O algoritmo proposto, denominado MILS-RVND, é uma heurística *multi-start* baseada na metaheurística *Iterated Local Search* – ILS, tendo como busca local o método *Randomized Variable Neighborhood Descent* – RVND. Seu pseudocódigo é apresentado no Algoritmo 2.

Contrariamente ao método *Variable Neighborhood Descent* (VND) clássico (Hansen et al., 2010), que trabalha com um conjunto de vizinhanças previamente ordenadas, o RVND utiliza uma ordem aleatória de vizinhanças a cada chamada. Essa é uma vantagem, uma vez que não é necessário fazer experimentos para descobrir qual a melhor ordem (Penna et al., 2011; Souza et al., 2010a).

Observa-se, pelo Algoritmo 2, que o MILS-RVND começa calculando, na linha 4, a frota de veículos necessária para resolver o problema (veja Seção 4.1). A seguir, a heurística *Multi-start* é executada *MaxIterMS* vezes (linhas 6 – 19), onde a cada iteração, na linha 7, uma solução inicial é construída por um método construtivo (veja Seção 4.2). O parâmetro *MaxIterILS* representa o número máximo de perturbações consecutivas sem melhora. O laço principal do ILS (linhas 10 – 16) envolve três procedimentos. No primeiro, linha 11, busca-se aprimorar a solução corrente pela aplicação do algoritmo RVND (veja Seção 6) como método de busca local. No segundo, linha 12, é verificado se o ótimo local gerado pelo RVND será aceito como novo ótimo local. Uma solução é aceita se houver melhora na função de avaliação ou se houver redução no número de veículos, dependendo de qual for o objetivo do problema proposto. No terceiro procedimento do laço do ILS, linha 15, aplica-se um mecanismo de perturbação (veja Seção 4.4). Finalizado o ILS, atualiza-se a melhor solução gerada até então (linha 17).

Nas seções seguintes os componentes principais do MILS-RVND são descritos.

4.1 Estimativa do número de veículos

Em alguns casos, o número de veículos disponíveis não é especificado no problema. Como o método construtivo depende dessa informação, um algoritmo de estimativa do número inicial de veículos foi desenvolvido. Seja LC a Lista de Candidatos composta pelos clientes que não foram adicionados à solução parcial. Inicialmente,

Algoritmo 2: MILS-RVND(*MaxIterMS*, *MaxIterILS*, v)

```
1  Início
2      CarregaDados( )
3      se v não for definido então
4          v ← EstimaNumeroDeVeiculos(semente)
5      f* ← ∞
6      para i ← 1 até MaxIterMS faça
7          s ← GeraSolucaoInicial(v, semente)
8          s' ← s
9          iterILS ← 0
10         enquanto (iterILS ≤ MaxIterILS) faça
11             s ← RVND(s)
12             se (f(s) < f(s') ou Redução Número Veículos = true) então
13                 s' ← s
14                 iterILS ← 0
15             s ← Perturba(s', semente)
16             iterILS ← iterILS + 1
17         se (f(s') < f*) então
18             s* ← s'
19             f* ← f(s')
20     retorna s*
21 fim
```

somente um veículo é considerado. Um cliente $i \in$ LC é escolhido aleatoriamente e inserido nesta única rota. Em seguida, enquanto a LC não for vazia, um critério de inserção, descrito na Seção 4.2.1, é selecionado aleatoriamente para calcular a inclusão do cliente $i \in$ LC em cada posição da rota e a inserção de menor custo é aplicada. Se o veículo estiver cheio um novo veículo é adicionado.

O número de veículos usados no algoritmo de construção varia de acordo com a variante do PRV. No PRVFH limitado, esse número é a soma da quantidade de veículos disponíveis de cada tipo; nos PRVs com múltiplos depósitos, ele é obtido somando-se a quantidade de veículos disponíveis em cada depósito; no PRVFH ilimitado, esse número inicia-se com um veículo de cada tipo e é acrescido durante o algoritmo de construção, conforme descrito na próxima seção. Finalmente, no PRVA é utilizado um limite inferior para o número de veículos ($v_{\min}$), sendo ele o menor inteiro maior ou igual ao resultado da divisão entre a soma das demandas dos clientes e a capacidade do veículo.

4.2 Construção da solução inicial

O algoritmo de construção da solução inicial foi adaptado de métodos existentes na literatura. Duas estratégias de inserção foram adotadas: a Estratégia de Inserção Sequencial (EIS) e a Estratégia de Inserção Paralela (EIP). Com a finalidade de gerar soluções iniciais diversificadas, para cada uma dessas estratégias, dois critérios de inserção foram utilizados, a saber, o Critério de Inserção Mais Barata Viável Modificada (CIMBVM) e o Critério de Inserção Mais Próxima Viável (CIMPV).

O pseudocódigo do algoritmo `GeraSoluçãoInicial()` é apresentado no Algoritmo 3. Inicialmente, a Lista de Candidatos (LC) é preenchida com todos os clientes (linha 3). Em seguida, cada rota r' é preenchida com um cliente i selecionado, aleatoriamente, da LC (linhas 5 – 7). Também aleatoriamente são selecionados, a estratégia e o critério de inserção (linhas 8 – 9). A solução inicial é gerada (linhas 10 – 13) usando o critério e a estratégia previamente selecionados.

Se a solução s for inviável e a frota for conhecida (isto é, limitada) o algoritmo tenta obter uma nova solução que seja viável (linha 20). Caso a solução s seja inviável e a frota seja ilimitada, somente após um certo número de tentativas sem sucesso é que o algoritmo adiciona um novo veículo à frota e tenta reconstruir uma solução viável (linhas 15 – 20).

Ao final do algoritmo, para as variantes com frota heterogênea e ilimitada, onde é necessário dimensioná-la, para cada tipo de veículo, é adicionada à solução s uma rota vazia (linhas 21 e 22). Este artifício permite o redimensionamento da frota durante a fase de busca local.

Algoritmo 3: GeraSoluçãoInicial(v, semente)

```
1  Início
2      TentativasConsecutivas ← 0
3      Inicializa LC
4      Seja s = {s^1, ..., s^{v-1}} o conjunto composto por v − 1 rotas vazias
5      para r' ← 1 até v − 1 faça
6          s^{r'} ← i ∈ LC selecionado aleatoriamente
7          Atualize a LC                                  // LC ← LC \ {i}
8      EstratégiaInserção ← EIS ou EIP                    // seleção aleatória
9      CritérioInserção ← CIMBVM ou CIMPV                 // seleção aleatória
10     se (EstratégiaInserção = EIS) então
11         s ← InserçãoSequencial(s, v, LC, CritérioInserção)
12     senão
13         s ← InserçãoParalela(s, v, LC, CritérioInserção)
14     se (s for inviável) então
15         se v não foi definido então
16             TentativasConsecutivas ← TentativasConsecutivas + 1
17             se TentativasConsecutivas foi alcançado então
18                 v ← v + 1
19             TentativasConsecutivas ← 0
20         Vá para a linha 3
21     se (PRVDFH com frota ilimitada) então
22         Adicione em s uma rota vazia associada a cada tipo de veículo
23     retorna s
24 fim
```

4.2.1 Critérios de inserção

No critério CIMBVM, o custo de inserção de um cliente não roteado $h \in \text{LC}$ em uma dada rota é expresso pela Equação (7), na qual a função $g(h)$ representa o custo de inserção. O valor de $g(h)$ é calculado pela soma de dois termos. O primeiro computa o custo de inserção do cliente h entre cada par clientes adjacentes i e j, enquanto o segundo corresponde à penalização usada para desestimular inserções tardias de clientes localizados distantes do depósito. O fator γ define o peso do custo de ida e volta ao depósito.

$$g(h) = \left(c^k_{ih} + c^k_{hj} - c^k_{ij}\right) - \gamma\left(c^k_{0h} + c^k_{h0}\right) \tag{7}$$

O critério CIMPV calcula a distância entre um cliente $h \in \text{LC}$ e todos os clientes que já tenham sido incluídos na solução parcial, ou seja $g(h) = c^k_{ih}$ (Equação (8)). Assume-se que a inserção de h é sempre realizada após i.

$$g(h) = c^k_{ih} \tag{8}$$

Nos dois critérios adotados, o cliente $i_{\min}$ inserido é aquele associado ao menor custo, isto é, $i_{\min} \leftarrow \arg\min\{g(h) \mid h \in \text{LC}\}$.

4.2.2 Estratégias de inserção

São duas as estratégias de inserção utilizadas para adicionar um cliente a uma rota: a Estratégia de Inserção Sequencial (EIS) e a Estratégia de Inserção Paralela (EIP).

Na EIS, cujo pseudocódigo é apresentado no Algoritmo 4, uma única rota é considerada a cada iteração. Se o critério de inserção selecionado for o CIMBVM, um valor de γ é escolhido, aleatoriamente, no conjunto $\{0,00, 0,05,\ 0,10, \ldots, 1,65, 1,70\}$ (linha 4), conforme experimentos empíricos de Subramanian et al. (2010). Em seguida, enquanto a LC não estiver vazia e existir pelo menos um cliente $i \in \text{LC}$ que possa ser adicionado à solução parcial corrente, sem violar qualquer restrição (linhas 7-15), cada rota é preenchida com um cliente selecionado pelo critério de inserção previamente definido (linhas 8-14). Se ao final do laço anterior ainda restar cliente não atendido, dois casos podem ocorrer. Se a frota for heterogênea e ilimitada (linhas 21-23),

Algoritmo 4: InserçãoSequencial(s, v, LC, CritérioInserção)

```
1  Início
2      clienteInserido ← falso
3      γ ← 0
4      se (CritérioInserção = CIMBVM) então
5          γ ← valor aleatório dentro de um dado conjunto
6      r_0 ← 1
7      enquanto (LC ≠ ∅ e pelo menos um cliente i ∈ LC pode ser adicionado a s) faça
8          para r' ← r_0 até v e LC ≠ ∅ faça
9              se (pelo menos um cliente i ∈ LC pode ser adicionado no veículo r') então
10                 Calcule o valor de cada custo g(i) para i ∈ LC
11                 i' ← arg min{g(i) | i ∈ LC}
12                 s^{r'} ← s^{r'} ∪ {i'}
13                 Atualize LC
14                 clienteInserido ← verdadeiro
15         Atualize r_0
16     se (|LC| > 0 e houver múltiplos depósitos e restrição de duração de rota e clienteInserido =
       verdadeiro ) então
17         s ← RVND(s)
18         clienteInserido ← falso
19         Vá para a linha 7
20     se (|LC| > 0 e a frota for heterogênea e ilimitada) então
21         Adicione um novo veículo selecionado aleatoriamente
22         Atualize r_0                                   // r_0 ← v
23         Vá para linha 7
24     retorna s
25 fim
```

um novo veículo, selecionado aleatoriamente entre os tipos existentes, é adicionado e o algoritmo é reiniciado a partir da linha 7; em qualquer outro caso, é realizado um novo procedimento de construção.

A Figura 1 ilustra um exemplo de uma iteração do procedimento construtivo utilizando a EIS com CIMPV. Neste caso, considere $r_1 = 0 \rightarrow 1 \rightarrow 2 \rightarrow 0$ como sendo a rota na qual a melhor inserção deverá ser avaliada. No exemplo em questão, o algoritmo avalia a melhor possibilidade de inserção dentre todas as possíveis, conforme pode ser observado nas Figuras 1.b – 1.g. A inserção de menor custo ocorre quando o cliente 3 é inserido após o cliente 2 (veja Figura 1.d)

O funcionamento da EIP é similar a da EIS, porém ela difere da anterior por considerar todas as rotas durante o processo de avaliação da inserção de um cliente. O Algoritmo 5 ilustra o seu pseudocódigo. Neste caso, um cliente é adicionado à rota r de menor custo de inserção, segundo o critério de inserção selecionado (linhas 7 – 10). Este procedimento é repetido enquanto a LC não estiver vazia e existir pelo menos um cliente $k \in$ LC que possa ser incluído em s (linhas 6 – 12). O restante do código é semelhante ao EIS.

4.3 Busca local

A busca local é feita pelo algoritmo *Randomized Variable Neighborhood Descent* – RVND (Subramanian et al., 2010). O RVND é uma variação do método *Variable Neighborhood Descent* – VND (Hansen et al., 2010), que ao invés de usar uma ordem pré-definida de vizinhanças para explorar o espaço de soluções, utiliza uma ordem aleatória a cada chamada. Mais especificamente, sempre que em uma determinada vizinhança não for possível melhorar a solução corrente, o RVND seleciona, aleatoriamente, outra vizinhança para continuar a busca pelo espaço de soluções. Esta abordagem, utilizada em Subramanian et al. (2010), Penna et al. (2011), Silva et al. (2012) e Subramanian & Battarra (2013), produz, em média, resultados melhores do que a versão com ordem determinística das vizinhanças e tem a vantagem de não necessitar de um estudo de qual a melhor ordem de vizinhanças.

O pseudocódigo do algoritmo RVND é apresentado no Algoritmo 6. Para explicar seu funcionamento, seja LV uma Lista de Vizinhanças inter-rotas. A LV é inicializada com vizinhanças associadas aos movimentos descritos na Seção 4.3.1 (linha 3). A cada iteração (linhas 4 – 16) uma vizinhança $\mathcal{N}^{(\eta)} \in LV$ é selecionada aleatoriamente (linha 5) e o melhor movimento viável é determinado (linha 6). Em caso de melhora da solução

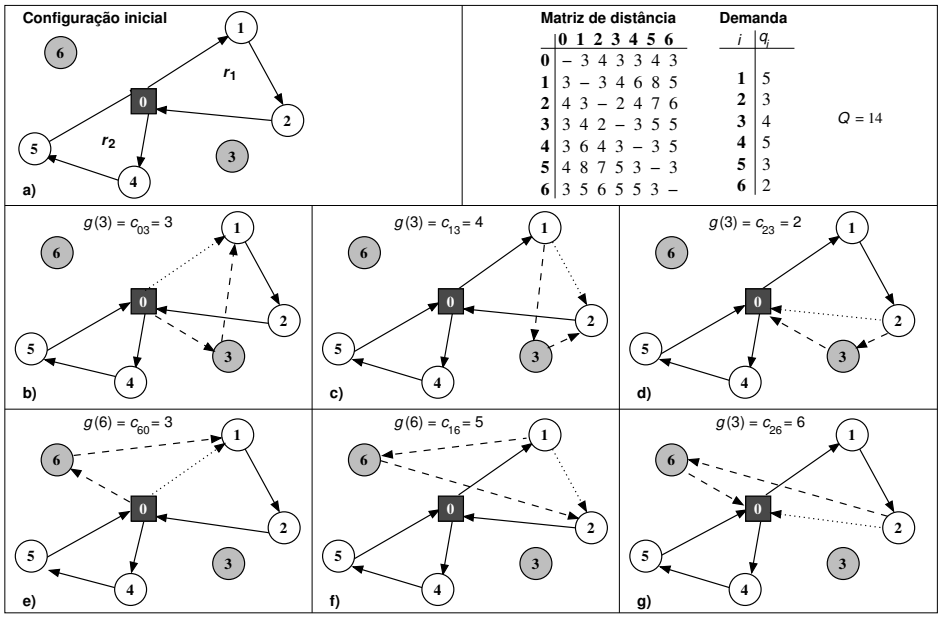

Figura 1. Exemplo de uma iteração do EIS com CIMPV.

Algoritmo 5: InserçãoParalela(s, v, LC, CritérioInserção)

1 **Início**
2 $clienteInserido \leftarrow$ **falso**
3 $\gamma \leftarrow 0$
4 **se** *(CritérioInserção = CIMBVM)* **então**
5 $\gamma \leftarrow$ valor aleatório dentro de um dado conjunto
6 **enquanto** ($LC \neq \emptyset$ **e** *pelo menos um cliente* $i \in LC$ *pode ser adicionado a* s) **faça**
7 Calcule o custo $g(i)$ para cada $i \in$ LC
8 $i' \leftarrow \arg\min\{g(i) \mid i \in \text{LC}\}$
9 $r' \leftarrow$ rota associada ao i'
10 $s^{r'} \leftarrow s^{r'} \cup \{i'\}$
11 Atualize LC
12 $clienteInserido \leftarrow$ **verdadeiro**
13 **se** ($|LC| > 0$ **e** *houver múltiplos depósitos* **e** *restrição de duração de rota* **e** $clienteInserido =$ **verdadeiro**) **então**
14 $s \leftarrow \text{RVND}(s)$
15 $clienteInserido \leftarrow$ **falso**
16 Vá para a linha 6
17 **se** ($|LC| > 0$ **e** *a frota for heterogênea e ilimitada*) **então**
18 Adicione um novo veículo selecionado aleatoriamente
19 Vá para linha 6
20 **retorna** s
21 **fim**

corrente, uma busca local intra-rota é executada (Algoritmo 7), a frota é atualizada e LV é reinicializada com todas as vizinhanças (linhas 7 – 13). Caso contrário, $\mathcal{N}^{(\eta)}$ é removida da LV (linha 14). Se o problema possuir como objetivo principal a minimização do número de veículos, linha 16, como é o caso do PRVA, então um algoritmo específico é aplicado para tentar esvaziar rotas (veja Algoritmo 9). Um conjunto de Estruturas de Dados Auxiliares (EDAs), implementadas com a finalidade de acelerar a busca e evitar a avaliação desnecessária de movimentos nas vizinhanças, é atualizado no início (linha 2) e durante a execução do método (linha 15). Uma descrição detalhada dessas estruturas e suas aplicações em cada vizinhança pode ser encontrada em Subramanian (2012).

Algoritmo 6: RVND(s)

```
1  Início
2      Atualize as EDAs
3      Inicialize a Lista de Vizinhanças (LV) Inter-Rota
4      enquanto (LV ≠ ∅) faça
5          Escolha uma vizinhança N^(η) ∈ LV aleatoriamente
6          Encontre o melhor vizinho s' de s ∈ N^(η)
7          se (f(s') < f(s)) então
8              s ← s'
9              s ← BuscaIntra-rota(s)
10             se (a frota for heterogênea e ilimitada) então
11                 Atualize a frota de modo que exista um veículo vazio de cada tipo
12             Atualize a LV
13         senão
14             Remova N^(η) da LV
15         Atualize as EDAs
16         TentaEsvaziarRota(s)                    // somente para o PRVA
17     retorna s
18 fim
```

Algoritmo 7: BuscaIntra-rota(s)

```
1  Início
2      Inicialize Lista de Vizinhanças Intra-rotas (LV')
3      enquanto (LV' ≠ ∅) faça
4          Escolha uma vizinhança N'^(η) ∈ LV' aleatoriamente
5          Encontre o melhor vizinho s' de s ∈ N'^(η)
6          se (f(s') < f(s)) então
7              s ← s'
8          senão
9              Remova N'^(η) da LV'
10     retorna s
11 fim
```

4.3.1 Estruturas de vizinhança inter-rotas

Um conjunto $\mathcal{N}$ de nove diferentes tipos de movimentos é usado para fazer modificações entre clientes de rotas distintas. Esses movimentos definem as estruturas de vizinhanças inter-rotas, sendo que as seis primeiras vizinhanças foram aplicadas em todas as variantes (Figura 2) e as três restantes em variantes específicas. Entre as seis estruturas, cinco delas são baseadas em movimentos λ-*interchanges* (Osman, 1993), que consiste na troca de até λ clientes consecutivos entre duas rotas. Para minimizar o esforço computacional λ foi limitado a 2. De acordo com Cordeau & Laporte (2005), essas trocas são melhores explicadas como duplas (λ_1, λ_2) (com $\lambda_1 \leqslant \lambda$ e $\lambda_2 \leqslant \lambda$), sendo que os λ_1 clientes são transferidos da rota r_1 para a rota r_2 e os λ_2 clientes da rota r_2 para a rota r_1. Portanto, desconsiderando as simetrias, são possíveis as seguintes combinações de movimentos 2-*interchanges*: $((1,0),(1,1),(2,0),(2,1),(2,2))$. Observe que tais combinações incluem movimentos de realocação $((1,0),(2,0))$ e de troca $((1,1),(2,1),(2,2))$. A sexta vizinhança considerada é baseada no operador *Cross-exchange* (Taillard et al., 1997), que consiste em trocar dois segmentos de diferentes rotas. Estas vizinhanças são apresentadas na Figura 2. Para as variantes do PRVFH ilimitada, uma sétima vizinhança implementada é a denominada τ-*Shift*, que consiste em transferir um conjunto de τ clientes consecutivos de uma rota para outra (Figura 3). Por fim, duas vizinhanças para as variantes contendo múltiplos depósitos, denominadas *ShiftDepot* e *SwapDepot*, foram utilizadas (Figura 4). As estruturas de vizinhanças inter-rotas estão descritas a seguir.

***Shift*(1,0)** – $\mathcal{N}^{(1)}$: Um cliente i é transferido de uma rota r_1 para outra rota r_2. A Figura 2.b, que exemplifica o movimento, mostra que o cliente 7 retirado de uma rota é reinserido em outra. A carga do

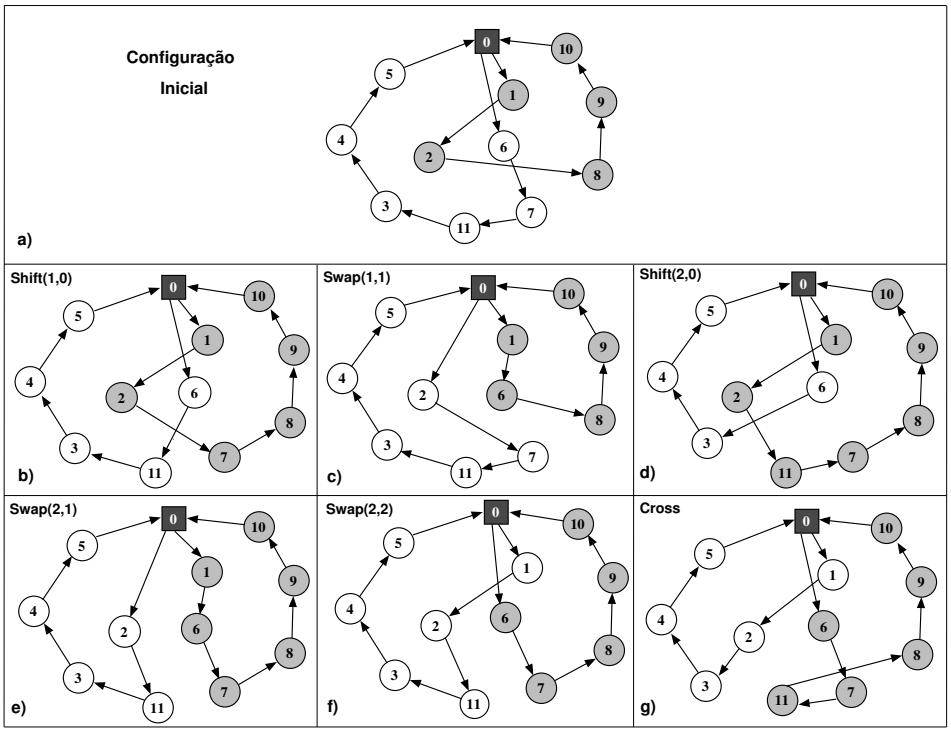

Figura 2. Vizinhanças inter-rotas.

veículo é verificada como segue. Todos os clientes localizados antes da posição de inserção têm suas cargas acrescidas em q_i unidades, enquanto que os localizados depois possuem suas cargas adicionadas em p_i unidades.

***Swap*(1,1)** – $\mathcal{N}^{(2)}$: Executa uma permutação entre um cliente i de uma rota r_1 e um cliente j pertencente a uma outra rota r_2. Na Figura 2.c, o cliente 2 é trocado com o cliente 6. As cargas dos veículos das duas rotas são examinadas de modo similar. Por exemplo, no caso de r_2, todos os clientes, situados antes da posição que j foi encontrado (agora substituído por i), têm suas cargas acrescidas de q_i e diminuídas de q_j unidades, enquanto que as cargas dos clientes posicionados após i aumentarão de p_i e diminuirão de p_j unidades.

***Shift*(2,0)** – $\mathcal{N}^{(3)}$: Dois clientes consecutivos i e j são transferidos de uma rota r_1 para outra r_2. Na Figura 2.d, os clientes adjacentes 7 e 11 são movidos de uma rota para outra. Todos os clientes localizados antes da posição de inserção em r_2 têm suas cargas somadas em $q_i + q_j$, enquanto que aqueles localizados depois têm suas cargas adicionadas em $p_i + p_j$. A nova carga do arco (i, j) também deve ser verificada.

***Swap*(2,1)** – $\mathcal{N}^{(4)}$: Dois clientes consecutivos, i e j, pertencentes a uma rota r_1 são permutados com um cliente i' de outra rota r_2. Na Figura 2.e, os clientes adjacentes 6 e 7 de uma rota são trocados com o cliente 2 de outra rota. A carga é verificada de modo similar às abordagens usadas nas vizinhanças *Shift*(2,0) e *Swap*(1,1).

***Swap*(2,2)** – $\mathcal{N}^{(5)}$: Dois clientes consecutivos, i e j, pertencentes a uma rota r_1 são permutados com outros dois clientes consecutivos i' e j' de outra rota r_2. Na Figura 2.f, os clientes adjacentes 6 e 7 de uma rota são trocados com os clientes adjacentes 1 e 2 de outra rota. A carga é verificada como na vizinhanças *Swap*(2,1).

Cross – $\mathcal{N}^{(6)}$: O arco entre os clientes adjacentes i e $i+1$ de uma rota r_1, e o arco entre os clientes adjacentes j e $j+1$ de outra rota r_2 são removidos. Em seguida, para se inserir novos arcos, os clientes i e $j+1$ são conectados, assim como os clientes j e $i+1$. Na Figura 2.g, os arcos $(2, 8)$ e $(11, 3)$ são removidos e os arcos $(11, 8)$ e $(2, 3)$ inseridos. O algoritmo para testar a carga do veículo de cada rota é feita como segue. A carga inicial $(\mathcal{L}_0)$ e final $(\mathcal{L}_f)$ dos veículos das duas rotas é calculado em tempo constante utilizando-se as EDAs. Se os valores de $\mathcal{L}_0$ e $\mathcal{L}_f$ não excederem a capacidade do veículo Q_k, então as cargas intermediárias são verificadas pela seguinte expressão: $\mathcal{L}_i = \mathcal{L}_{i-1} + p_i - q_i$. Consequentemente, se $\mathcal{L}_i$ ultrapassar Q_k, o movimento é inviável.

τ-Shift – $\mathcal{N}^{(7)}$: um subconjunto de τ clientes consecutivos é transferido de uma rota r_1 de maior capacidade para outra rota r_2 de menor capacidade. É assumido que os custos fixos e variáveis de r_2 são menores que aqueles de r_1. O valor de τ é iniciado por 3 e varia progressivamente até alcançar o número de clientes que compõem a rota r_1. Esse movimento é específico para problemas de roteamento com frota heterogênea e tem o objetivo de esvaziar as rotas feitas por veículos de custo mais elevado. Ele é aplicado mesmo que a rota r_2 esteja vazia. Na Figura 3, os clientes consecutivos 1, 2 e 7 são transferidos de uma rota para outra.

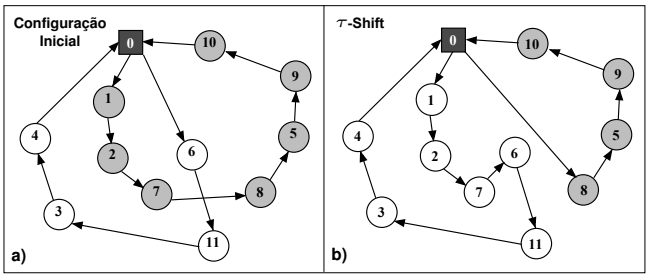

Figura 3. Vizinhança inter-rota τ-*Shift*.

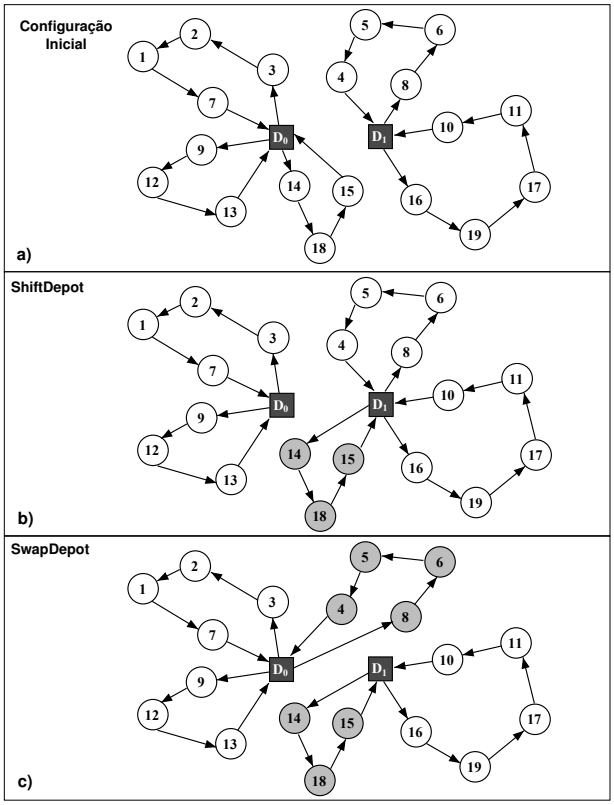

Figura 4. Vizinhanças inter-rotas para multi-depósitos.

ShiftDepot – $\mathcal{N}^{(8)}$: Uma rota r é transferida de um depósito d_1 para um outro depósito d_2, desde que exista um veículo disponível no segundo. Em princípio, qualquer movimento é considerado viável. Na Figura 4.b, a rota composta pelos clientes 14, 18 e 15 é transferida de um depósito para outro.

SwapDepot – $\mathcal{N}^{(9)}$: Permutação entre uma rota r_i de um depósito d_1 e uma rota r_j de um depósito d_2. Assim como no caso anterior, qualquer movimento é considerado viável. Na Figura 4.c, a rota composta pelos clientes 14, 18 e 15, pertencente ao depósito D_0 é trocada com a rota composta pelos clientes 4, 5, 6 e 8, pertencente ao depósito D_1.

O pseudocódigo do algoritmo geral das estruturas de vizinhanças $\mathcal{N}^{(1)}$ a $\mathcal{N}^{(7)}$ é descrito no Algoritmo 8. Para cada par de rotas r_1 e r_2 (linhas 2 e 3), o algoritmo verifica se há movimentos viáveis ou que não foram examinados previamente (linha 4). Se uma das duas condições for satisfeita, um cliente (ou subconjunto de clientes consecutivos, dependendo da vizinhança) da rota r_1 é selecionado para ter seu movimento avaliado no caso de ser transferido ou permutado por outro cliente (ou subconjunto de clientes consecutivos, dependendo da vizinhança), desde que as condições de viabilidade sejam atendidas (linhas 5 – 8). Se o movimento for viável e, ao mesmo tempo, conduzir a uma solução de melhor qualidade em relação à solução incumbente ($\bar{s}$) da iteração corrente do MILS-RVND, então a solução $\bar{s}$ é atualizada (linhas 9 – 11).

A busca local em todas as vizinhanças é realizada de forma exaustiva, considerando apenas soluções viáveis. A estratégia de *best improvement* foi utilizada. Em princípio, a checagem de viabilidade das cargas deve ser verificada apenas no(s) depósito(s), com exceção das variantes envolvendo coleta e entrega, em que as cargas devem ser verificadas ao longo da rota.

Algoritmo 8: AlgoritmoGeralParaVizinhanças1–7(s)

```
1  Início
2      para r1 ← 1 até v faça
3          para r2 ← 1 até v (ou r2 ← r1 + 1 até v para Swap(1,1)) faça
4              se (r1 ≠ r2) e condições de viabilidade dadas pelas EDAs forem satisfeitas) então
5                  para todo cliente i ∈ r1 (ou subconjunto de clientes consecutivos de r1) faça
6                      se (condições de viabilidade associadas ao cliente i ou arco (i, j) dadas pelas EDAs
                       forem satisfeitas) então
7                          para cada cliente j' em r2 (ou subconjunto de clientes consecutivos de r2) faça
8                              Calcule o custo da solução f(s') da solução vizinha s' de s usando a
                               vizinhança η
9                              se (f(s') < f(s̄) e s' é viável) então
10                                 f(s̄) ← f(s')
11                                 s̄ ← s'
12     se f(s̄) < f(s) então
13         s ← s̄
14     retorna s
15 fim
```

4.3.2 Estruturas de vizinhança intra-rotas

Um conjunto $\mathcal{N}'$ de seis diferentes tipos de movimentos encontrados na literatura é usado para fazer modificações entre clientes de uma mesma rota. Esses movimentos, baseados em *Reinsertion*, *Or-opt* (Or, 1976), 2-*opt* e *Swap*, definem as estruturas de vizinhanças intra-rotas.

Reinsertion – $\mathcal{N}'^{(1)}$: Um cliente é removido e reinserido em outra posição da rota. Na Figura 5.b, o cliente 3, na rota original $0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 0$ é realocado para outra posição na rota, sendo a nova rota a sequência $0 \to 1 \to 2 \to 4 \to 5 \to 3 \to 6 \to 7 \to 0$.

Or-opt2 – $\mathcal{N}'^{(2)}$: Dois clientes consecutivos são removidos e reinseridos em outra posição da rota. Na Figura 5.c, os clientes adjacentes 2 e 3 são realocados para outra posição na sequência, formando a nova rota $0 \to 1 \to 4 \to 5 \to 2 \to 3 \to 6 \to 7 \to 0$.

Or-opt3 – $\mathcal{N}'^{(3)}$: Três clientes consecutivos são removidos e reinseridos em outra posição da rota. Na Figura 5.d, apresenta a nova rota $0 \to 4 \to 5 \to 1 \to 2 \to 3 \to 6 \to 7 \to 0$, com os clientes adjacentes 1, 2 e 3 realocados em outra posição, em relação a rota original.

2-opt – $\mathcal{N}'^{(4)}$: Dois arcos não-adjacentes são removidos e outros dois são adicionados de modo a formarem uma nova rota. Na Figura 5.e, os arcos $(2,3)$ e $(5,6)$ são excluídos e os novos arcos $(2,5)$ e $(3,6)$ são inseridos, de modo a obter a nova rota $0 \to 1 \to 2 \to 5 \to 4 \to 3 \to 6 \to 7 \to 0$. Esta estrutura de vizinhança não é aplicada na variante assimétrica.

Swap – $\mathcal{N}'^{(5)}$: Consiste em permutar dois clientes de uma mesma rota. Na Figura 5.f, os clientes 2 e 6 são trocados, formando a rota $0 \to 1 \to 6 \to 5 \to 4 \to 3 \to 2 \to 7 \to 0$.

Reverse – $\mathcal{N}'^{(6)}$: Este movimento inverte a sequência de visitas de uma rota se o valor total da carga correspondente à rota for reduzido. Na Figura 5.g, todos os arcos tiveram seus sentidos invertidos, obtendo-se a rota $0 \to 7 \to 6 \to 5 \to 4 \to 3 \to 2 \to 1 \to 0$. Esta vizinhança é aplicada somente aos problemas envolvendo coleta e entrega.

4.3.3 Tentativa de esvaziar rotas

Em algumas variantes do PRV, minimizar o número de veículos é o objetivo principal. Por isso, um algoritmo guloso e randomizado foi desenvolvido para tratar este caso, o qual é apresentado no Algoritmo 9. A ideia é fazer uso da capacidade residual e da duração residual das rotas de uma dada solução s por meio de uma busca local, com a finalidade de diminuir o número de rotas de s. O algoritmo começa criando uma cópia da solução s em s' (linha 2). Na linha 3, a Lista de Rotas (LR) é criada, sendo essa lista composta pelas rotas da solução s. Enquanto |LR| for maior que 1 (linhas 4 – 11), uma tentativa de esvaziar uma rota é realizada. Para tal, uma rota r é selecionada para ser removida de LR, de acordo com um dos seguintes critérios: (i) rota com a maior carga; (ii) rota com a maior duração; (iii) seleção aleatória. O critério de seleção de rota é escolhido aleatoriamente (linha 5). Em seguida, nas linhas 9 – 11, enquanto for possível mover um cliente de qualquer rota $r' \in$ LR para r ou ainda for possível trocar um cliente de qualquer rota $r' \in$ LR com outro em

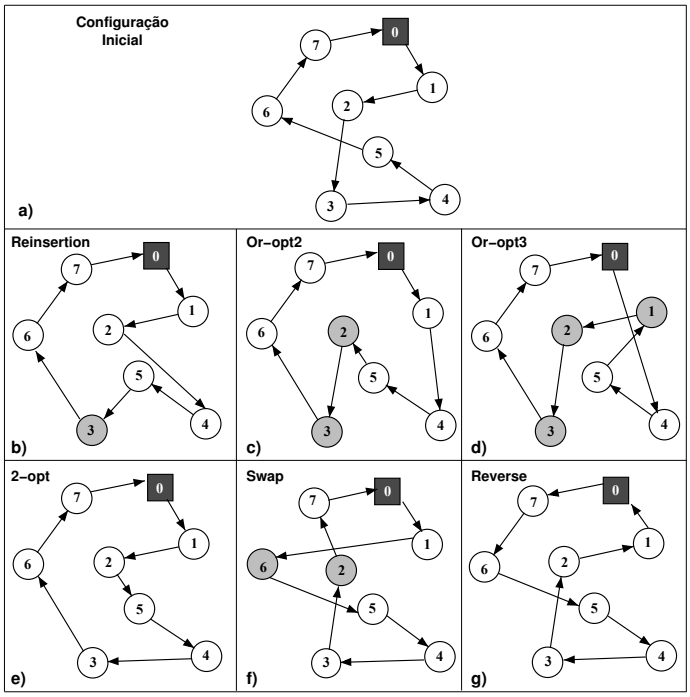

Figura 5. Vizinhanças intra-rotas.

r, de tal modo que a carga de r aumente, uma busca local é feita entre a rota r e aquela em LR usando as estruturas de vizinhanças *Shift*(1,0), *Shift*(2,0) e *Swap*(1,1). O melhor movimento possível é considerado para cada uma dessas três vizinhanças. Além disso, no caso do *Shift*(1,0) e *Shift*(2,0) o movimento é imediatamente aceito se a rota $r' \in$ LR ficar vazia, enquanto que, no caso do *Swap*(1,1), o movimento é aceito somente se a carga do veículo r for incrementada. Uma busca local intra-rota é realizada em cada rota modificada usando as estruturas de vizinhanças *2-opt* e *Shift*. Se o algoritmo não for capaz de esvaziar uma rota, então a solução corrente é restaurada (linhas 12 – 13).

Algoritmo 9: TentaEsvaziarRota(s)

```
1  Início
2      s' ← s
3      Inicialize a Lista de Rotas (LR) com as rotas de s
4      enquanto (|LR| > 1) faça
5          Escolha um critério de seleção de rota aleatoriamente
6          Escolha uma rota r ∈ LR de acordo com o critério selecionado
7          Remova r de LR
8          enquanto (for possível mover um cliente de qualquer rota r' ∈ LR para r ou for possível trocar
               um cliente de qualquer rota r' ∈ LR com um outro em r de tal modo que a carga de r aumente)
           faça
9              s ← Shift(1,0)
10             s ← Shift(2,0)
11             s ← Swap(1,1)
12     se (o número de rotas de s for igual ao número de rotas de s') então
13         s ← s'
14     retorna s
15 fim
```

4.4 Mecanismos de perturbação

Três mecanismos de perturbação $P = \{P^{(1)}, P^{(2)}, P^{(3)}\}$, que aceitam somente modificações que geram soluções viáveis, foram implementados. Quando a função `Perturba()` (linha 15 do Algoritmo 2) é executada, uma

destas perturbações é selecionada aleatoriamente. A perturbação $P^{(3)}$ é aplicada somente na variante com frota heterogênea e ilimitada.

***Multi-Swap*(1,1)** – $P^{(1)}$: Múltiplos movimentos *Swap*(1,1) são executados aleatoriamente. Isto é, sorteia-se um número n_v no conjunto $\{1, \ldots, n_{swap}\}$ e aplica-se esse movimento n_v vezes consecutivas. Nos testes realizados, n_{swap} foi fixado em 2.

***Multi-Shift*(1,1)** – $P^{(2)}$: Múltiplos movimentos *Shift*(1,1) são executados aleatoriamente. O *Shift*(1,1) consiste em transferir um cliente i de uma rota r_1 para outra rota r_2, enquanto que um cliente j de r_2 é transferido para r_1. Esta perturbação é aplicada tal como no caso anterior.

Split – $P^{(3)}$: Consiste em dividir uma rota em rotas menores. Para tal, seja $T' = \{2, \ldots, t\}$ um subconjunto de T composto por todos os veículos, exceto pelo de menor capacidade. Inicialmente, uma rota $r \in s$ associada com um veículo $k \in T'$ é selecionada aleatoriamente. Em seguida, enquanto r não estiver vazia, os clientes de r são transferidos sequencialmente para uma nova rota $r' \notin s$ associada a um veículo $k' \in \{1, \ldots, k-1\}$ sem que a capacidade do veículo k' seja violada. As novas rotas geradas são adicionadas à solução s, enquanto que a rota r é excluída de s. O procedimento descrito é repetido um determinado número de vezes, sendo o número de repetições selecionado aleatoriamente no conjunto $\{1, 2, \ldots, |R|\}$. Esta perturbação, normalmente, aumenta o número de rotas e é aplicada somente ao PRVFH ilimitada.

5. Experimentos Computacionais

O algoritmo proposto foi desenvolvido em C++ (compilador g++ 4.4.3). Para testá-lo foram usadas instâncias clássicas da literatura para cada um dos problemas de roteamento elencados na Seção 2. Os testes relativos às instâncias com frota homogênea foram executados 50 vezes para cada instância em um computador Intel Core 2 Quad 2,4 GHz com 4 GB de RAM, enquanto aquelas com frota heterogênea foram executadas 30 vezes para cada problema em um computador Intel Core i7 2,93 GHz com 8 GB de RAM. Em todas as variantes foram utilizadas instâncias bem conhecidas da literatura.

Uma descrição mais detalhada da calibração dos parâmetros, apresentada na Seção 5.1, e dos resultados do MILS-RVND, contidas na Seção 5.2, são disponibilizadas em Subramanian (2012).

5.1 Calibração dos parâmetros

Um conjunto representativo contendo diferentes tipos de instâncias foi selecionado para calibrar os dois parâmetros do MILS-RVND: *MaxIterMS* e *MaxIterILS*. Foi possível observar, por meio de experimentos empíricos, que o *MaxIterILS* variou de acordo com o tamanho das instâncias, mais precisamente, com o número de clientes e veículos. Com a finalidade de simplificar o cálculo do *MaxIterILS* foi adotada uma expressão linear em função de n e v, dada pela Equação (9).

$$MaxIterILS = n + \beta \times v \tag{9}$$

O Parâmetro β na Equação (9) corresponde a um valor inteiro não-negativo, que indica o nível de influência do número de veículos v no valor de *MaxIterILS*.

Pelos experimentos realizados, verificou-se que os melhores valores para os parâmetros do algoritmo MILS-RVND são os seguintes: $\beta = 5$ e $MaxIterMS = 50$ para as variantes com frota homogênea e $\beta = 5$ e $MaxIterMS = 400$ para as variantes com frota heterogênea.

5.2 Resultados

A Tabela 1 resume o resultado da aplicação do algoritmo MILS-RVND na resolução de diversas variantes do PRV. Nesta tabela, **Desvio Méd.** corresponde ao desvio médio entre os valores médios (obtidos após as execuções do algoritmo) e o melhor valor conhecido para cada uma das instâncias testadas; **#Instâncias** representa o número de problemas-teste usados por um determinado autor, **#Melhoras** indica o número de soluções que o algoritmo proposto conseguiu melhorar e **#Empates** o número de empates.

Pela Tabela 1, verifica-se que, dentre as 508 instâncias testadas, o algoritmo MILS-RVND superou os melhores resultados até então existentes em 43 instâncias e alcançou as melhores soluções em outras 395 instâncias. Além disso, o desvio médio foi sempre menor que 0,66%, exceto no caso das instâncias de Golden et al. (1998), nas quais esse desvio foi 1,03%.

6. Conclusões

Neste capítulo foi apresentado um algoritmo heurístico unificado, chamado MILS-RVND, baseado na metaheurística *Iterated Local Search* (ILS), para resolver uma ampla classe de problemas de roteamento de veículos. O algoritmo tem como procedimento de busca local o método *Randomized Variable Neighborhood Descent* (RVND) e é aplicado em uma estratégia *Multi-Start*. O MILS-RVND tem a vantagem de possuir uma estrutura

Tabela 1. Resumo dos Resultados do MILS-RVND

Variante	#Instâncias	#Melhoras	#Empates	Desvio Méd. (%)	Tempo Méd. (s)
PRVC[1]	[a]90, [b]14,[c]20	[a]0, [b]0, [c]0	[a]87, [b]10, [c]1	[a]0,10, [b]0,26, [c]1,03	[a]8,04, [b]23,56, [c]798,49
PRVCA[1]	[d]24	[d]1	[d]23	[d]0,15	[d]2,54
PRVA[1]	[a]90, [e]16, [f]8	[a]1, [e]2, [f]6	[a]85, [e]12, [f]1	[a]0,09, [e]0,62, [f]0,19	[a]9,31, [e]24,50, [f]843,25
PRVCES[1]	[g]40, [h]28, [i]18	[g]0, [h]3, [i]2	[g]40, [h]20, [i]10	[g]0,02, [h]0,28, [i]0,26	[g]2,04, [h]32,34, [i]433,89
PRVCEM[1]	[h]42	[h]12	[h]26	[h]0,09	30,05
PRVMD[1]	[j]23, [k]10	[j]0, [k]0	[j]16, [k]4	[j]0,47, [k]0,66	[j]225,11, [k]292,16
PRVMDCEM[1]	[h]33	[h]14	[h]17	[h]0,23	[h]188,62
PRVFH[2]	[l]16, [m]36	[l]1, [m]2	[l]14, [m]29	[l]0,23, [m]0,16	[l]32,39, [m]28,51
Total	508	43	395		

[a]: Conjuntos A, B, E, F, M, P; [b]: Christofides et al. (1979); [c]: Golden et al. (1998).

[d]: Fischetti et al. (1994) e Pessoa et al. (2008).

[e]: Christofides et al. (1979) e Fisher (1994); [f]: (Li et al., 2007).

[g]: Dethloff (2001); [h]: (Salhi & Nagy, 1999); [i]: Montané & Galvão (2006);

[j]: Cordeau et al. (1997) (old); [k]: Cordeau et al. (1997) (new).

[l]: Taillard (1999); [m]: Golden et al. (1984).

[1]: Core 2 Quad 2.4 GHz (executado em um único processador).

[2]: Core i7 2.93 GHz (executado em um único processador).

extremamente simples e requerer poucos parâmetros. Os resultados computacionais mostraram sua eficiência, especialmente em termos de qualidade das soluções finais produzidas, assim como de sua flexibilidade em resolver diversas variantes do PRV. De acordo com os resultados obtidos, o algoritmo proposto detém os melhores resultados da literatura em 86,2% das instâncias, sendo que em 8,5% delas, ele foi capaz de apresentar novos melhores resultados. Este desempenho notável merece destaque, uma vez que os melhores resultados conhecidos estão distribuídos entre vários algoritmos de diversos autores e não concentrados em um único.

Agradecimentos

Os autores agradecem à CAPES e ao CNPq pelo apoio no desenvolvimento deste trabalho e aos colegas Marcos Melo Silva, Edcarllos Santos e Renatha Capua pelas correções efetuadas no texto.

Referências

Christofides, N.; Mingozzi, A. & Toth, P., The vehicle routing problem. In: Christofides, N.; Mingozzi, A.; Toth, P. & Sandi, C. (Eds.), *Combinatorial Optimization*. New York, USA: J. Wiley & Sons, p. 315–338, 1979.

Cordeau, J.F.; Gendreau, M. & Laporte, G., A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

Cordeau, J.F. & Laporte, G., New heuristics for the vehicle routing problem. In: Rego, C. & Alidaee, B. (Eds.), *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*. Norwell, USA: Kluwer Academic Publishers, p. 145–163, 2005.

Dantzig, G.B. & Ramser, J.H., The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

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

Fischetti, M.; Toth, P. & Vigo, D., A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42(5):846–859, 1994.

Fisher, M.L., Optimal solutions of vehicle routing problems using minimum K-trees. *Operations Research*, 42(4):626–642, 1994.

Golden, B.L.; Assad, A.A.; Levy, L. & Gheysens, F.G., The feet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984.

Golden, B.L.; Wasil, E.A.; Kelly, J.P. & Chao, I.M., Metaheuristics in vehicle routing. In: Crainic, T.G. & Laporte, G. (Eds.), *Fleet Management and Logistics*. Boston, USA: Kluwer Academic Publishers, p. 33–56, 1998.

Hansen, P.; Mladenović, N. & Moreno Pérez, J., Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

Li, F.; Golden, B. & Wasil, E., The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & Opertions Research*, 34(10):2918–2930, 2007.

Lourenço, H.R.; Martin, O.C. & Stützle, T., Iterated local search. In: Glover, F. & Kochenberger, G.A. (Eds.), *Handbook of Metaheuristics*. Norwell, USA: Kluwer Academic Publishers, p. 321–353, 2003.

Montané, F.A.T. & Galvão, R.D., A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *European Journal of Operational Research*, 33(3):595–619, 2006.

Or, I., Traveling Salesman-type Combinational Problems and their Relation to the Logistics of Blood Banking. PhD Thesis, Northwestern University, Evanston, USA, 1976.

Osman, I.H., Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(1-4):421–451, 1993.

Penna, P.H.V.; Subramanian, A. & Ochi, L.S., An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 2011(Aceito para publicação).

Pessoa, A.; Uchoa, E. & de Aragão, M.P., Robust branch-and-cut-and-price algorithms for vehicle routing problems. In: Golden, B.; Raghavan, S. & Wasil, E. (Eds.), *The Vehicle Routing Problem: Latest Advances and New Challenges*. New York, USA: Springer, p. 297–325, 2008.

Salhi, S. & Nagy, G., A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the Operational Research Society*, 50:1034–1042, 1999.

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

Souza, M.J.F.; Coelho, I.M.; Ribas, S.; Santos, H.G. & Merschmann, L.H.C., A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, 207(2):1041–1051, 2010a.

Souza, M.J.F.; Mine, M.T.; Silva, M.S.A.; Ochi, L.S. & Subramanian, A., A hybrid heuristic, based on iterated local search and GENIUS, for the vehicle routing problem with simultaneous pickup and delivery. *International Journal of Logistics Systems Management*, 10(2):142–156, 2010b.

Subramanian, A., Heuristic, Exact and Hybrid Approaches for Vehicle Routing Problems. Tese de doutorado, Instituto de Computação, Universidade Federal Fluminense, 2012.

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

Subramanian, A.; Drummond, L.M.A.; Bentes, C.; Ochi, L.S. & Farias, R., A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37(11):1899–1911, 2010.

Taillard, E.D., A heuristic column generation method for heterogeneous fleet. *RAIRO*, 33(1):1–14, 1999.

Taillard, E.D.; Badeau, P.; Gendreau, M.; Guertin, F. & Potvin, J.Y., A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

Toth, P. & Vigo, D. (Eds.), *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. Philadelphia, USA: SIAM, 2002.

Notas Biográficas

Anand Subramanian é graduado em Engenharia de Produção Mecânica e mestre em Engenharia de Produção (Universidade Federal da Paraíba – UFPB, 2006 e 2008, respectivamente), e doutor em Computação (Universidade Federal Fluminense – UFF, 2012). Atualmente é professor Adjunto I do departamento de Engenharia de Produção da UFPB e atua nos seguintes temas: otimização combinatória, meta-heurísticas, programação inteira, algoritmos híbridos (*Matheuristics*), problemas de roteamento de veículos e otimização aplicada à gestão de atividades acadêmicas.

Puca Huachi Vaz Penna é graduado em Ciência da Computação e mestre em Engenharia Mineral (UFOP, 1995 e 2009, respectivamente). Atualmente é doutorando em Computação (UFF) e professor Assistente da UFF, atuando em nos seguintes temas: meta-heurística, roteamento de veículos e sequenciamento em uma máquina.

Luiz Satoru Ochi é graduado em Matemática (Universidade Estadual Paulista Júlio de Mesquita Filho, 1977), mestre em Matemática Aplicada (Universidade Estadual de Campinas, 1981) e doutor em Engenharia de Sistemas e Computação (Universidade Federal do Rio de Janeiro, 1989). Atualmente é pesquisador nível 1D do CNPq e professor Titular do Instituto de Computação da UFF, e atua nos seguintes temas: inteligência computacional, programação meta-heurística, programação matemática, algoritmos paralelos e distribuídos em otimização, grafos e algoritmos, inteligência artificial e pesquisa operacional.

Marcone Jamilson Freitas Souza é graduado em Engenharia Metalúrgica (UFOP, 1982), mestre e doutor em Engenharia de Sistemas e Computação (UFRJ, 1989 e 2000, respectivamente) e tem pós-doutorado (UFF, 2009). Atualmente é bolsista de produtividade em pesquisa do CNPq, na área de Engenharia de Produção e Transportes, e professor Associado III da UFOP, atuando principalmente em meta-heurísticas, planejamento da produção, planejamento de lavra, programação de horários (*timetabling*), transporte público e roteamento de veículos.