O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via *Iterated Local Search* e GENIUS

Marcio Tadayuki Mine[1]; Matheus de Souza Alves Silva[2]; Luiz Satoru Ochi[3];
Marcone Jamilson Freitas Souza[4]; Thaís Cotta Barbosa da Silva[5]

Resumo: Este trabalho apresenta o algoritmo GENILS para resolver o Problema de Roteamento de Veículos com Coleta e Entrega Simultânea (PRVCES). GENILS é um algoritmo heurístico baseado nas técnicas heurísticas *Iterated Local Search*, *Variable Neighborhood Descent* e adaptações das heurísticas Inserção Mais Barata e GENIUS. O algoritmo proposto foi testado em três conjuntos consagrados de problemas-teste da literatura e se mostrou superior aos demais algoritmos da literatura com relação à capacidade de encontrar as melhores soluções conhecidas.

Abstract: This work presents GENILS for solving the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD). GENILS is a heuristic algorithm based on Iterated Local Search, Variable Neighborhood Descent and adaptations of the Cheapest Insertion and GENIUS heuristics. The proposed algorithm was tested on three well-known sets of instances found in literature and it overcame other existing algorithms in relation to the ability of finding the best known solutions.

1. INTRODUÇÃO

O Problema de Roteamento de Veículos (PRV), conhecido na literatura como *Vehicle Routing Problem* (VRP), foi originalmente proposto por Dantzig e Ramser (1959) e pode ser definido da seguinte forma: dado um conjunto N de clientes, cada qual com uma demanda d_i e uma frota de veículos homogênea com capacidade Q, tem-se como objetivo estabelecer os trajetos de custo mínimo a serem percorridos pelos veículos, de forma a atender completamente a demanda dos clientes.

Em 1989, Min propôs uma importante variante do PRV: o Problema de Roteamento de Veículos com Coleta e Entrega Simultânea (PRVCES), em que os serviços de entrega e coleta devem ser realizados simultaneamente. Este modelo é um problema básico na área da logística reversa, a qual visa planejar o transporte de produtos aos clientes, bem como o retorno de resíduos ou produtos utilizados por esses para a reciclagem ou depósitos especializados. A logística reversa pode ser observada, por exemplo, na logística postal ou no planejamento da distribuição de indústria de bebidas.

O PRVCES pertence à classe de problemas NP-difíceis, uma vez que ele pode ser reduzido ao PRV clássico quando nenhum cliente necessita de serviço de coleta. Dessa forma, diversos trabalhos na literatura o tratam de forma heurística.

Min (1989) propôs um método de três fases para resolver o planejamento de distribuição de materiais para bibliotecas públicas. A primeira fase do método consiste em agrupar os clientes em *clusters* por meio do Método de Ligação por Médias (*Average Linkage Method*) (Anderberg, 2007). A segunda fase associa os veículos às respectivas rotas e a terceira consiste em resolver cada *cluster* por meio de uma heurística para o Problema do Caixeiro Viajante. Essa heurística atribui, iterativamente, uma penalidade aos arcos em que a carga do veículo foi excedida, procurando, dessa forma, gerar uma solução viável.

Halse (1992) abordou o PRVCES por meio de uma heurística que consiste em, primeiramente, associar os clientes aos veículos e, em seguida, gerar as rotas através de um procedimento baseado na busca local *3-opt*.

Dethloff (2001) desenvolveu uma adaptação do método da Inserção Mais Barata, em que os clientes são adicionados às rotas seguindo três critérios: (*i*) distância; (*ii*) capacidade residual e (*iii*) distância do cliente ao depósito. Nesse trabalho não foi aplicado nenhum método de refinamento da solução.

Vural (2003) desenvolveu duas versões do Algoritmo Genético (Goldberg, 1989). A primeira faz a codificação dos indivíduos através de chaves aleatórias

[1] **Marcio Tadayuki Mine,** Instituto de Computação, Universidade Federal Fluminense, Rio de Janeiro, RJ, Brasil. (e-mail: mmine@ic.uff.br).

[2] **Matheus de Souza Alves Silva,** Instituto de Computação, Universidade Federal Fluminense, Rio de Janeiro, RJ, Brasil. (e-mail: msalves@ic.uff.br).

[3] **Luiz Satoru Ochi,** Instituto de Computação, Universidade Federal Fluminense, Rio de Janeiro, RJ, Brasil. (e-mail: satoru@ic.uff.br).

[4] **Marcone Jamilson Freitas Souza,** Instituto de Ciências Exatas e Biológicas, Universidade Federal de Ouro Preto, Ouro Preto, MG, Brasil. (e-mail: marcone@iceb.ufop.br).

[5] **Thaís Cotta Barbosa da Silva,** Instituto de Ciências Exatas e Biológicas, Universidade Federal de Ouro Preto, Ouro Preto, MG, Brasil. (e-mail: thais_cotta@yahoo.com.br).

Manuscrito recebido em 4/5/2010 e aprovado para publicação em 28/7/2010. Este artigo é parte de TRANSPORTES, volume XVIII, número 3, setembro de 2010. ISSN: 2237-1346 (online).

(*Random Keys*) (Bean, 1994) e a segunda foi implementada como uma heurística de refinamento, baseada na estrutura do AG desenvolvido por Topcuoglu e Sevilmis (2002).

Gökçe (2004) trata o PRVCES com a metaheurística Colônia de Formigas (Dorigo *et al.*, 1996) e utiliza a busca local *2-opt* como procedimento de pós-otimização.

Nagy e Salhi (2005) desenvolveram uma metodologia para a resolução do PRVCES com a restrição de limite de tempo para percorrer cada rota. Essa metodologia reúne diferentes heurísticas para resolver o PRV clássico, tais como as buscas locais *2-opt*, *3-opt*, e as baseadas em realocação e troca, assim como procedimentos para viabilizar a solução.

Dell'Amico *et al.* (2006) utilizaram a técnica *branch-and-price* por meio de duas abordagens: programação dinâmica e relaxação do espaço de estados (*state space relaxation*).

Crispim e Brandão (2005) propuseram uma técnica híbrida, combinando as metaheurísticas Busca Tabu (Glover e Laguna, 1997) e *Variable Neighborhood Descent* – VND (Hansen e Mladenović, 2001). Para gerar uma solução foi utilizado o método da varredura (*sweep method*) e, para refiná-la, um procedimento de busca local que explora o espaço de soluções com movimentos de realocação e troca.

Röpke e Pisinge (2006) desenvolveram uma heurística baseada na *Large Neighborhood Search* – LNS (Shaw, 1998). O LNS é uma busca local baseada em duas ideias para definir e explorar estruturas de vizinhança de alta complexidade. A primeira delas consiste em fixar uma parte da solução e, assim, facilitar a busca nessa porção do espaço de soluções do problema. A segunda consiste em prosseguir com a busca por meio de programação por restrições, programação inteira, técnicas *branch-and-cut*, entre outras.

Montané e Galvão (2006) utilizaram a metaheurística Busca Tabu considerando quatro tipos de estruturas de vizinhança. Essas estruturas utilizam movimentos de realocação, troca e *crossover*. Para gerar uma solução vizinha foram desenvolvidas duas estratégias, sendo que a primeira considera o primeiro movimento viável (*First Improvement*) e a segunda, o melhor movimento viável (*Best Improvement*).

Chen (2006) tratou o problema por meio de uma técnica baseada nas metaheurísticas *Simulated Annealing* – SA (Kirkpatrick *et al.*, 1983) e Busca Tabu, enquanto Chen e Wu (2006) desenvolveram uma metodologia baseada na heurística *record-to-record travel* (Dueck, 1993), a qual é uma variação do SA.

Algoritmos construtivos, heurísticas de refinamento e técnicas baseadas na metaheurística Busca Tabu são apresentados em Bianchessi e Righini (2007). Essas técnicas utilizam movimentos de troca de nós (*node-exchange-based*) e troca de arcos (*arc-exchange-based*).

Wassan *et al.* (2007) propuseram uma versão reativa da metaheurística Busca Tabu. Para gerar uma solução inicial, foi utilizado o método da varredura (*sweep method*) e para explorar o espaço de soluções, movimentos de realocação, de troca e de inversão do sentido da rota.

Em Subramanian *et al.* (2008) foi desenvolvido um algoritmo baseado em *Iterated Local Search* (ILS), tendo como busca local o procedimento *Variable Neighborhood Descent* (VND). Para gerar uma solução inicial foi utilizada uma adaptação da heurística de inserção de Dethloff (2001), porém sem considerar a capacidade residual do veículo. O VND explora o espaço de soluções usando movimentos baseados em realocação, troca e *crossover*. O VND realiza, a cada melhora na solução corrente, uma intensificação nas rotas alteradas, por meio dos procedimentos *reverse*, e os de busca local *Or-opt*, *2-opt*, *exchange*. O procedimento *Or-opt* implementado consiste em permutar um, dois ou três clientes consecutivos em uma rota. O procedimento *reverse* consiste em inverter o sentido da rota, caso haja redução na carga do veículo nos arcos. Já os procedimentos *2-opt* e *exchange* realizam a busca local baseada em movimentos de permutação de um par de arcos e dois clientes, respectivamente. Os mecanismos de perturbação aplicados no ILS foram o *ejection chain*, o *double swap* e o *double bridge*. O *ejection chain* consiste em transferir um cliente de cada rota a outra adjacente. O *double swap* consiste em realizar duas trocas sucessivas e o *double bridge* consiste em remover quatro arcos e inserir quatro novos arcos de forma a criar uma nova rota. Uma descrição detalhada desse algoritmo, bem como uma nova formulação de programação matemática para o PRVCES pode ser encontrada em Subramanian (2008).

Zachariadis *et al.* (2009) trataram o PRVCES com uma técnica híbrida, combinando as metaheurísticas Busca Tabu e *Guided Local Search* (Voudouris e Tsang, 1996). Posteriormente, os mesmos autores propuseram em Zachariadis *et al.* (2010) um algoritmo evolucionário, que usa uma memória adaptativa para armazenar características de soluções de alta qualidade geradas durante o processo de busca. Essas características são, então, usadas para produzir novas soluções em regiões promissoras do espaço de busca, as quais são, a seguir, melhoradas por um método de Busca Tabu.

Uma boa revisão do PRVCES pode ser encontrada em Parragh *et al.* (2008a, 2008b).

Para comparar as abordagens da literatura, Dethloff (2001) propôs um conjunto com 40 problemas envolvendo 50 clientes cada. Salhi e Nagy (1999) apresentaram 28 problemas-teste com 50 a 199 clientes, sendo

que a metade desses têm restrições de limite de tempo. Por fim, Montané e Galvão (2006) adaptaram 18 problemas-teste de Solomon *et al.* (1995) e Gehring e Homberger (1999), envolvendo 100, 200 e 400 clientes.

A Tabela 1 mostra o número de melhores resultados conquistados pelos melhores algoritmos da literatura nos conjuntos de problema-teste da literatura. Por esta tabela, verifica-se que: (*i*) no conjunto de Dethloff (2001), os algoritmos de Zachariadis *et al.* (2010) e Subramanian *et al.* (2008) são os melhores, alcançando todas as melhores soluções conhecidas; (*ii*) no conjunto de Salhi e Nagy (1999), os melhores algoritmos são os de Wassan *et al.* (2007) e o de Zachariadis *et al.* (2010) e (*iii*) no conjunto de Montané e Galvão (2006), o melhor algoritmo é o de Subramanian *et al.* (2008).

Tabela 1. Número de melhores resultados nos conjuntos de problemas-teste, por algoritmo

Algoritmo	***Conjunto de Problemas-teste***		
	Dethloff (2001)	*Salhi e Nagy (1999)*	*Montané e Galvão (2006)*
Chen e Wu (2006)	–	1	–
Röpke e Pisinger (2006)	26	–	–
Wassan *et al.* (2007)	–	6	–
Zachariadis *et al.* (2010)	40	6	10
Subramanian *et al.* (2008)	40	2	13

Neste trabalho é expandido e aprimorado o trabalho de Mine *et al*. (2010), no qual se apresenta um novo algoritmo heurístico para resolver o PRVCES. O algoritmo proposto, denominado GENILS, combina as técnicas *Iterated Local Search* (Stützle *et al*., 1999), *Variable Neighborhood Descent* (Mladenović e Hansen, 1997) e adaptações das heurísticas GENIUS (Gendreau *et al.*, 1992) e Inserção Mais Barata. Ele difere do de Subramanian *et al*. (2008) basicamente por incluir a heurística GENIUS e os procedimentos 3-*opt* e 4-*opt* na exploração do espaço de busca. Conforme mostram os resultados obtidos, estas estratégias se mostraram eficientes na resolução do problema.

O restante deste trabalho está organizado como segue. Na Seção 2 descreve-se o problema abordado. A metodologia proposta é apresentada na Seção 3. Na Seção 4 são apresentados e analisados os resultados encontrados, enquanto na Seção 5 conclui-se o trabalho, apontando-se trabalhos futuros.

2. DESCRIÇÃO DO PROBLEMA

O Problema de Roteamento de Veículos com Coleta e Entrega Simultânea (PRVCES), ou *Vehicle Routing Problem with Simultaneous Pickup and Delivery* (VRPSPD), é uma variante do PRV clássico. Neste problema é considerado um depósito com uma frota ilimitada de veículos de capacidade Q e um conjunto N de clientes espalhados geograficamente. Cada cliente $i \in N$ está associado a duas quantidades d_i e p_i, que representam o que vai ser entregue e coletado em um cliente i, respectivamente. O objetivo do problema é definir as rotas necessárias para atender a todos os clientes, de forma a minimizar os custos referentes ao deslocamento dos veículos e satisfazer as seguintes restrições: (a) cada rota deve iniciar e finalizar no depósito; (b) todos os clientes devem ser visitados uma única vez e por um único veículo; (c) as demandas por coleta e entrega de cada cliente devem ser completamente atendidas; (d) a carga do veículo, em qualquer momento, não pode superar a capacidade do veículo. Em algumas variantes desse problema, considera-se também a necessidade de cada veículo não percorrer mais que um determinado limite de distância (tempo). A Figura 1 ilustra um exemplo do PRVCES.

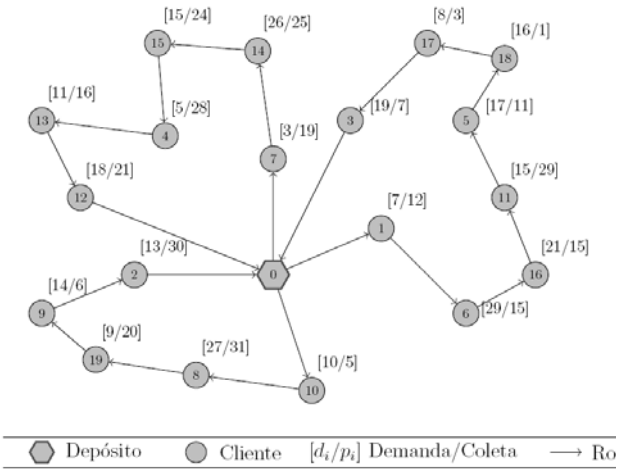


Figura 1. Exemplo de PRVCES

Na Figura 1, os clientes são representados pelos números 1 a 19 e o depósito pelo número 0 (zero). Cada par [d_i / p_i] denota a demanda e coleta em um cliente i, respectivamente. Nesta figura, há três rotas a serem executadas por veículos de capacidade $Q = 150$. Em uma delas, o veículo sai do depósito e atende aos clientes 10, 8, 19, 9 e 2, nesta ordem, retornando ao depósito no final. No primeiro cliente atendido nessa rota, é feita uma entrega de 10 unidades do produto e recolhida outras 5 unidades. A última visita do veículo ocorre no cliente 2, o qual demanda 13 unidades do produto e necessita que sejam coletadas 30 unidades.

3. METODOLOGIA

Apresenta-se, nesta Seção, o algoritmo proposto para resolver o PRVCES. Na Subseção 3.1, seu pseudocódigo é apresentado. Nas subseções seguintes, os procedimentos que o compõe são detalhados.

3.1. Algoritmo GENILS

O algoritmo proposto para resolver o PRVCES é um

algoritmo híbrido, denominado GENILS, que combina os procedimentos heurísticos *Iterated Local Search* – ILS (Stützle e Hoos, 1999), *Variable Neighborhood Descent* – VND (Hansen e Mladenović, 2001; Mladenović e Hansen, 1997) e GENIUS (Gendreau *et al*., 1992).

O ILS é uma metaheurística que explora o espaço de soluções por meio da aplicação de perturbações em ótimos locais encontrados na busca. Basicamente, ele consiste em partir de uma solução inicial, aplicar uma busca local nessa solução e, para escapar desse ótimo local e se dirigir para outras regiões do espaço de soluções, aplica perturbações nesse ótimo local.

No algoritmo GENILS desenvolvido, a busca local do ILS é feita pelo VND e a solução inicial é gerada com base em três procedimentos construtivos: Inserção Mais Barata rota a rota, Inserção Mais Barata com múltiplas rotas e uma adaptação da heurística GENIUS (Gendreau *et al*., 1992). Apesar de em testes preliminares a heurística baseada na GENIUS normalmente ter gerado soluções de melhor qualidade na maioria dos casos, houve casos em que o melhor desempenho foi da heurística da Inserção Mais Barata com múltiplas rotas. Por outro lado, a heurística da Inserção Mais Barata rota a rota, apesar de ganhar apenas em poucos casos, era necessária para fornecer o número de rotas iniciais para as demais heurísticas. Assim, optou-se por implementar todas essas heurísticas construtivas.

O pseudocódigo do algoritmo GENILS proposto é mostrado na Figura 2.

O algoritmo GENILS começa gerando três soluções iniciais s^A, s^B, e s^C, cada qual por um dos métodos descritos na Subseção 3.3. Essas soluções são, a seguir, refinadas pelo procedimento de busca local VND (descrito na Subseção 3.6) e a melhor solução obtida é usada como solução inicial *s*. Para escapar do ótimo local *s*, é feita uma perturbação (descrita na Subseção 3.7), gerando-se uma nova solução *s*'. Em seguida, essa solução perturbada é refinada pela busca local VND, obtendo-se um novo ótimo local *s*". Esta solução torna-se a nova solução corrente caso *s*" seja melhor que *s*; caso contrário, ela é descartada e nova perturbação é feita a partir da solução *s*. Esse procedimento é repetido até que o máximo de iterações sem melhora na solução corrente ($iter_{\max}$) seja atingido.

```
Algoritmo GENILS
Entrada: Conjunto de buscas locais,
         Número máximo de iterações iter_max
Saída:   solução s

início
  γ ← número aleatório [0;0,7]          {Subseção 3.3}
  s^A ← IMB – 1R()                      {Subseção 3.3}
  nrotas ← no. de rotas solução s^A     {Subseção 3.3}
  s^B ← IMB – MR(nrotas, γ)             {Subseção 3.3}
  s^C ← VRGENIUS(nrotas)                {Subseção 3.3}
  s^A ← VND(s^A)                        {Subseção 3.6}
  s^B ← VND(s^B)
  s^C ← VND(s^C)
  s ← s' = argmin{f(s^A),f(s^B),f(s^C)}
  iter ← 0
  enquanto (iter < iter_max)faça
    iter ← iter + 1
    s' ← perturbação(s)                 {Subseção 3.7}
    s" ← VND(s')
    se (f(s")<f(s)) faça
      s ← s"
      iter ← 0
    fim-se
  fim-enquanto
  retorne s
fim
```

Figura 2. GENILS

3.2. Função de avaliação

Uma solução *s* é avaliada pela função *f* apresentada pela expressão (1), que determina o custo total de deslocamento.

$$f(s) = \sum_{(i,j)\in A} c_{ij} x_{ij} \qquad (1)$$

em que, N: conjunto dos clientes, incluindo o depósito;
A: conjunto dos arcos (i, j), com $i, j \in N$;
c_{ij}: custo de deslocamento ou distância de i a j;
x_{ij}: variável binária que assume valor 1 se na solução s o arco $(i,j) \in A$ for utilizado ($x_{ij} = 1$) e valor zero ($x_{ij} = 0$), caso contrário.

3.3. Geração de uma solução inicial

Para gerar uma solução inicial são utilizadas três heurísticas baseadas em inserção. A primeira, denominada *IMB-1R*, é uma adaptação da heurística de Inserção Mais Barata, a qual constrói uma solução rota a rota.

Inicialmente, gera-se uma rota *r* contendo um cliente escolhido aleatoriamente. Em seguida, calcula-se o custo de inserção e_{ij}^k, dado pela expressão (2), de cada cliente *k* que ainda não está presente na solução, entre cada par de clientes *i* e *j* da rota *r*.

$$e_{ij}^k = (c_{ik} + c_{kj} - c_{ij}) - \gamma(c_{0k} + c_{k0}) \qquad (2)$$

Na expressão (2), a primeira parcela refere-se ao custo de inserção de um cliente *k* entre os clientes *i* e *j* e a segunda parcela corresponde a uma bonificação dada a um cliente que se situa distante do depósito. Essa bonificação é controlada por um fator $\gamma \in [0, 1]$ e favorece a inserção de um cliente, de forma a não adicioná-lo tardiamente à rota. Detalhes sobre a influência do parâmetro γ podem ser encontrados em Subramanian *et al*. (2008).

O cliente que tiver o menor custo de inserção é adicionado à rota, desde que a sua inserção deste não viole a restrição de capacidade do veículo. Caso a inserção de qualquer cliente implique na sobrecarga do veículo, então a rota corrente é finalizada, e inicia-se a construção de uma nova rota. Esse procedimento é repetido até que todos os clientes sejam adicionados à

solução.

A segunda heurística construtiva utilizada, a Inserção Mais Barata com Múltiplas Rotas, denotada por *IMB-MR*, foi proposta por Subramanian *et al*. (2008) e baseia-se na heurística de inserção de Dethloff (2001). Inicialmente, são construídas *nrotas* rotas, cada qual com um único cliente escolhido de forma aleatória, sendo *nrotas* o número de rotas geradas pela heurística *IMR-1R*. Em seguida, adiciona-se um cliente *k* entre os clientes *i* e *j* de uma rota *r*, tal que o custo de inserção e_{ij}^{k}, dado pela expressão (2), seja o menor possível. Esse procedimento é executado iterativamente até que não haja mais clientes a serem inseridos. É importante destacar que a inserção de um cliente somente é realizada se não houver a sobrecarga do veículo na rota considerada. Como essa heurística é dependente do número de rotas iniciais definidas *a priori*, pode ser que ao final do procedimento restem clientes que não podem ser adicionados a nenhuma rota, isto é, o método nem sempre gera uma solução completa. Nesse caso, continua-se a construção rota a rota por meio da heurística *IMB-1R*.

A terceira e última heurística construtiva utilizada, denominada VRGENIUS (linha 5 da Figura 2), é uma adaptação da heurística GENIUS (Gendreau *et al.*, 1992), desenvolvida originalmente para o Problema do Caixeiro Viajante. Essa heurística possui duas fases: uma construtiva (VRGENI) e outra de refinamento (VRUS).

A fase VRGENI é um método de inserção, cuja característica fundamental é que a inclusão de um cliente não é realizada necessariamente entre dois outros clientes consecutivos. No entanto, esses dois clientes tornam-se adjacentes após a inserção. Já a fase VRUS consiste em, a cada iteração, remover um cliente da solução e reinseri-lo em outra posição que melhore a solução corrente. Se isso não for possível com nenhum cliente, esta fase é finalizada.

Em ambas as fases, tanto a remoção, quanto a inserção de um cliente é realizada por procedimentos que analisam um espaço reduzido da vizinhança explorada pelas buscas locais *3-opt* e *4-opt*. A eficiência desses procedimentos encontra-se no fato de que o espaço analisado é restrito ao número de vizinhos de cada cliente, sendo, no máximo, igual a um parâmetro *h*.

O número de rotas necessárias para a aplicação das heurísticas de Inserção Mais Barata com múltiplas rotas (*IMB-MR*) e *VRGENIUS* é aquele proveniente da

```
Procedimento VRGENI
Entrada: Número de rotas nrotas,
         Número máximo de iterações iter_max
Saída:   solução s

início
  m ← nrotas
  N' ← N, representando o número máximo de clientes que fazem parte da solução
  enquanto (|N'| > 0) faça
    s ← ∅
    m' ← 1
    enquanto (m' ≤ m) faça
      Selecione aleatoriamente dois clientes v', v'' ∈ N'
      r ← rota contendo o depósito e os clientes v' e v''
      s ← s ∪ {r}
      N' ← N'\{v'}
      N' ← N'\{v''}
      m' ← m' + 1
    fim-enquanto
    N'' ← Ø
    enquanto (|N'| > 0) faça
      Selecione aleatoriamente um vértice v ∈ N'
      s' ← InsercaoGENI(v, s)
      se(existe um arco de s' tal que sua carga exceda a capacidade do veículo) faça
        N'' ← N'' ∪ {v}
      senão
        s ← s'
        N' ← N' ∪ N''
        N'' ← ∅
      fim-se
      N' ← N'\{v}
    fim-enquanto
    se (|N''| > 0) faça
      s ← ∅
      N' ← N
      m ← m + 1
    fim-se
  fim-enquanto
  retorne s
fim
```

Figura 3. Fase VRGENI do procedimento VRGENIUS

aplicação do método de Inserção Mais Barata rota a rota (*IMR-1R*).

A Figura 3 mostra o pseudocódigo da fase *VRGENI*. Nesse algoritmo, o método *InsercaoGENI*(*v*, *s*) consiste na fase GENI do algoritmo de Gendreau *et al*. (1992). Se o número de rotas não for suficiente para gerar uma solução viável, ele é incrementado em uma unidade e a fase *VRGENI* é aplicada novamente.

A Figura 4 mostra o pseudocódigo da fase *VRUS*. Nesse algoritmo, os métodos *RemocaoUS*(*v*, *r*) e *InsercaoGENI*(*v*, *s'*) consistem, respectivamente, nas fases US e GENI do algoritmo de Gendreau *et al*. (1992). Denota-se por p_z^r o *z*-ésimo vértice da rota $r \in R(s)$ a ser visitado, sendo $R(s)$ o conjunto de rotas da solução *s*. Dessa forma, p_1^r e p_n^r representam, respectivamente, as posições do primeiro e do último vértice da rota *r* a ser visitado. Além disso, denota-se por $v(p_z^r, s)$ o vértice que se encontra na posição *z* da rota *r* na solução *s*.

```
Procedimento VRUS
Entrada: Solução construída s
Saída:   Solução refinada s

início
  s* ← s
  para (r ∈ R(s)) faça
    p_z^r ← p_1^r
    enquanto (p_z^r ≤ p_n^r) faça
      v ← v(p_z^r, s)
      s' ← RemocaoUS(v,r)
      s' ← InsercaoGENI(v,s')
      se (f(s'') < f(s*)) faça
        s* ← s'
        r ← Primeira rota de R(s)
        p_z^r ← p_1^r
      senão
        p_z^r ← p_{z+1}^r
      fim-se
      s ← s''
    fim-enquanto
  fim-para
  s ← s*
  retorne s
fim
```

Figura 4. Fase VRUS do procedimento VRGENIUS

Na fase VRUS, realiza-se, a cada iteração, a remoção de um vértice v_i da rota *r* na solução *s*, que se encontra na posição p_z^r. Em seguida, esse vértice é inserido em uma posição que representa o melhor custo de inserção, considerando todas as rotas da solução *s*. Se a solução gerada for melhor que a melhor solução encontrada *s**, então o próximo vértice a ser considerado será o da primeira posição da primeira rota de *s*; caso contrário, o algoritmo avança para o vértice da próxima posição p_{z+1}^r. Esse procedimento é aplicado até que todos os vértices sejam analisados.

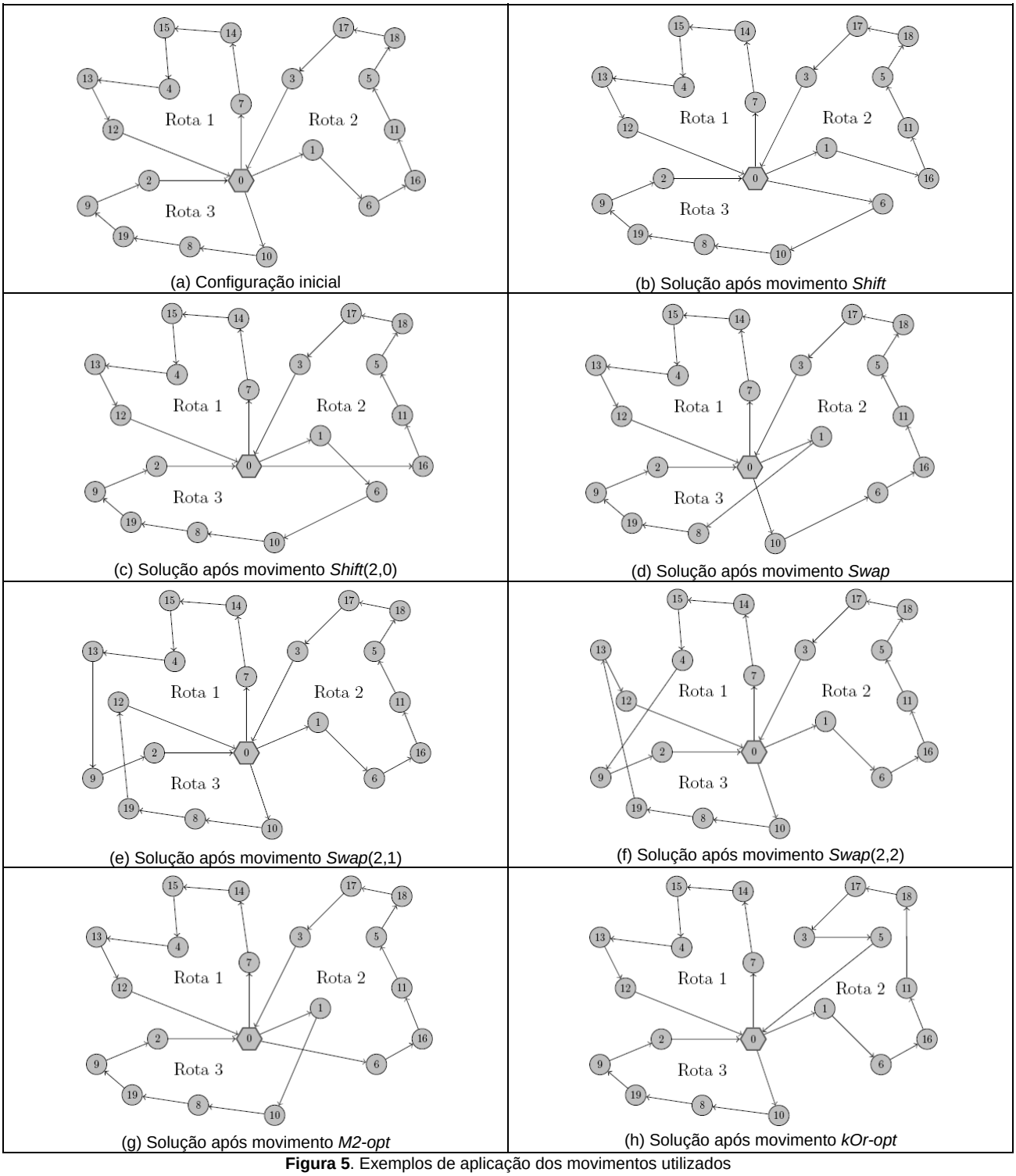
Um detalhamento maior desses procedimentos construtivos é encontrado em Mine (2009).

3.4. Estruturas de vizinhança

Para explorar o espaço de soluções do problema, foram aplicados sete tipos diferentes de movimentos, a saber: (a) *Shift*: movimento de realocação, que consiste em transferir um cliente de uma rota para outra; (b) *Shift*(2,0): movimento semelhante ao *Shift*, porém realocando dois clientes consecutivos de uma rota para outra; (c) *Swap*: consiste em trocar um cliente *i* de uma rota r_1 com um outro cliente *j* de uma rota r_2; (d) *Swap*(2,1): é análogo ao *Swap*, porém trocando dois clientes consecutivos de uma rota com um cliente de outra rota; (e) *Swap*(2,2): consiste em realizar a troca de dois clientes consecutivos de um rota com dois outros clientes consecutivos de outra rota; (f) *M2-opt*: consiste em remover dois arcos e inserir dois novos arcos de forma a recompor a rota; (g) *kOr-opt*: consiste em remover *k* clientes consecutivos de uma rota *r* e, em seguida, reinseri-los em uma outra posição nessa mesma rota. O valor de *k* é um parâmetro. *kOr-opt* é uma generalização do movimento *Or-opt*, proposto por Or (1976), que realiza a remoção de no máximo três clientes consecutivos.

Observa-se que somente são permitidos movimentos que conduzam a soluções viáveis.

A Figura 5 ilustra os resultados de aplicações dos movimentos utilizados. Na Fig. 5(a) é dada uma solução em sua configuração inicial. Em Fig. 5(b) mostra-se o resultado da aplicação do movimento *Shift*, transferindo o cliente 6 da rota 2 para a rota 3. Em Fig. 5(c) exemplifica-se o movimento *Shift*(2,0), com a realocação dos clientes 1 e 3 da rota 2 para a rota 3. Em Fig. 5(d) mostra-se a aplicação do movimento *Swap* envolvendo os clientes 1 e 10 das rotas 2 e 3, respectivamente. Em Fig. 5(e) tem-se o resultado da aplicação do movimento *Swap*(2,1) com a troca dos clientes consecutivos 9 e 2 da rota 3 com o cliente 12 da rota 1. Em Fig. 5(f) ilustra-se o resultado da aplicação do movimento *Swap*(2,2) envolvendo os clientes 13 e 12 da rota 1 e os clientes 9 e 2 da rota 3. Em Fig. 5(g) mostra-se o resultado da aplicação do movimento *M2-opt*, obtido após remoção dos arcos (1, 6) e (0, 10) e inserção dos arcos (1, 10) e (0, 6), para restabelecer a rota. Finalmente, em Fig. 5(h) ilustra-se a aplicação do movimento *kOr-opt*, com $k = 3$, com a transferência dos clientes consecutivos 18, 17 e 3 da rota 1 na posição do arco (11, 5).



(a) Configuração inicial

(b) Solução após movimento *Shift*

(c) Solução após movimento *Shift*(2,0)

(d) Solução após movimento *Swap*

(e) Solução após movimento *Swap*(2,1)

(f) Solução após movimento *Swap*(2,2)

(g) Solução após movimento *M2-opt*

(h) Solução após movimento *kOr-opt*

Figura 5. Exemplos de aplicação dos movimentos utilizados

3.5. Procedimentos *G3-opt*, *G4-opt* e *reverse*

Os procedimentos *G3-opt* e *G4-opt* são inspirados na heurística GENIUS, e representam adaptações das buscas locais 3-*opt* e 4-*opt*. A adaptação consiste em analisar a inserção de um arco (v_i; v_j) somente se os clientes v_i e v_j estiverem relativamente próximos. Para isso, define-se $N_h(v)$ como o conjunto dos h vizinhos mais próximos ao cliente v em uma rota r da solução s, sendo h um parâmetro. Além disso, considere as seguintes definições: N^r, conjunto dos clientes pertencentes à rota r; v_i: cliente $v_i \in N^r$; v_{z+1} e v_{z-1}: clientes, pertencentes à rota r, sucessor e antecessor ao cliente $v_z \in N^r$, respectivamente; v_j : cliente $v_j \in N_h(v_i)$; v_k: cliente pertencente à vizinhança $N_h(v_{i+1})$ no caminho de v_j para v_i; v_l: cliente pertencente à vizinhança $N_h(v_{j+1})$ no caminho de v_i para v_j e inv(r) a rota r no sentido inverso.

O procedimento *G3-opt* funciona da seguinte forma: a cada passo, é feita a remoção dos arcos (v_i; v_{i+1}), (v_j; v_{j+1}) e (v_k; v_{k+1}) e a inserção dos arcos (v_i; v_j), (v_{i+1}; v_k) e (v_{j+1}; v_{k+1}) na rota r, de forma a melhorar a solução s e tal que o custo seja o menor possível. Ressalta-se que ambos os sentidos da rota r são analisados. Este

procedimento é repetido até que não seja mais possível melhorar a solução *s*. A Figura 6 mostra o pseudocódigo de *G3-opt*.

```
Procedimento G3-opt
Entrada: Uma rota r de uma solução s
Saída:   Solução s melhorada

início
  r' ← ∅
  f(r') ← ∞
  enquanto (f(r) < f(r')) faça
    r' ← r
    para (r'' ∈ {r',inv(r')}) faça
      para (v_i ∈ N^r') faça
       para (v_j ∈ N_h(v_i), v_j ≠ v_i) faça
        para (v_k ∈ N_h(v_i+1), v_k ≠ v_i e v_j)
         r'''←r''/{(v_i,v_i+1),(v_j,v_j+1),(v_k,v_k+1)}
         r'''←r'''∪{(v_i,v_j),(v_i+1,v_k),
                    (v_j+1,v_k+1)}
         se (f(r''') < f(r)) faça
          r ← s'''
         fim-se
        fim-para
       fim-para
      fim-para
    fim-para
  fim-enquanto
  retorne s
fim
```

Figura 6. Procedimento *G3-opt*

O procedimento *G4-opt* é semelhante ao *G3-opt*, com a diferença de que, a cada iteração, são removidos os arcos (v_i; v_{i+1}), (v_{l-1}; v_l), (v_j ; v_{j+1}) e (v_{k-1}; v_k) e adicionados os arcos (v_i; v_j), (v_l; v_{j+1}), (v_{k-1}; v_{l-1}) e (v_{i+1}; v_k).

O procedimento *reverse* consiste em inverter o sentido de uma rota, sendo aplicado somente se ocorrer um aumento na carga residual da rota. A carga residual de uma rota é o valor da capacidade do veículo subtraído da maior carga do veículo nessa rota.

3.6. VND

O procedimento de busca local VND implementado é uma adaptação do método *Variable Neighborhood Descent* (Mladenović e Hansen, 1997). Ele consiste em explorar o espaço de soluções por meio de mudanças sistemáticas de vizinhanças. As vizinhanças são exploradas em uma certa ordem, previamente estabelecida, e sempre que uma solução de melhora é encontrada, retorna-se à primeira vizinhança. A Figura 7 ilustra seu funcionamento.

Como pode ser observado na Figura 7, o VND implementado inicialmente ordena, de forma aleatória, as sete vizinhanças baseadas nos movimentos *Shift*, *Shift*(2,0), *Swap*, *Swap*(2,1), *Swap*(2,2), *M2-opt* e *kOr-opt*, os quais são detalhados na Subseção 3.4. Observa-se que na versão original de Mladenović e Hansen (1997), em todas as chamadas desse procedimento a ordem das vizinhanças é fixa e, portanto, não muda durante a busca. No procedimento VND proposto, a cada chamada, essa ordem pode mudar. Prosseguindo,

```
Procedimento VND
Entrada: Solução s,  conjunto t vizinhanças
         distintas (Seção 3.4)
Saída:   Solução refinada s

início
  RN ← conjunto t vizinhanças,
       ordem aleatória
  i ← 1
  enquanto (i < t) faça
    Seja s_0 melhor vizinho em RN^(i)(s)
    se (f(s_0) < f(s)) então
     s ← s_0
     i ← 0
    {Intensificação das rotas modificadas s}
     s ← Busca local com Shift
     s ← Busca local com Shift(2,0)
     s ← Busca local com Swap
     s ← Busca local com M2-opt
     s ← Busca local com Swap(2,1)
     s ← Busca local com Swap(2,2)
     s ← Busca local com G3-opt
     s ← Busca local com G4-opt
     s ← Busca local com kOr-opt, k=3,4,5
     s ← reverse
    senão
     i ← i + 1
    fim-se
  fim-enquanto
  retorne s
fim
```

Figura 7. Procedimento VND

se na *t*-ésima vizinhança o melhor vizinho for de melhora em relação à solução corrente, então sobre as rotas modificadas desse vizinho é aplicada uma fase de intensificação. A intensificação consiste em realizar buscas locais nas sete vizinhanças mencionadas anteriormente, na ordem apresentada, seguidas da aplicação de buscas locais baseadas nos procedimentos *G3-opt*, *G4-opt* e *reverse*, também nessa ordem (todos descritos na Subseção 3.5). O método é interrompido se a solução corrente não for melhorada com a fase de intensificação em nenhuma das sete vizinhanças.

3.7. Perturbações

O mecanismo de perturbação do algoritmo GENIUS, linha 13 da Figura 2, consiste em escolher aleatoriamente uma das três estratégias descritas a seguir:

- Múltiplos *Shift*: Consiste em escolher um número inteiro no intervalo [1, 3], com distribuição uniforme, e aplicar, a seguir, esse número de movimentos *Shift* (descrito na Subseção 3.3) sucessivamente;
- Múltiplos *Swap*: Segue a mesma ideia da perturbação com múltiplos *Shift*, porém utilizando movimentos *Swap*;
- *Ejection chain*: Essa perturbação foi proposta por Rego e Roucairol (1996). Inicialmente, seleciona-se um subconjunto $R = \{r_1, r_2, ..., r_m\}$ com *m* rotas de forma arbitrária. Em seguida, transfere-se um cliente da rota r_1 para a rota r_2, um cliente de r_2 para r_3 e assim sucessivamente até que um cliente seja transferido da rota r_m pa-

ra a primeira rota r_1. Nesse movimento, cada cliente de uma rota é escolhido de forma aleatória.

4. RESULTADOS COMPUTACIONAIS

São apresentados, nesta Seção, os resultados computacionais obtidos pelo algoritmo heurístico híbrido proposto para resolver o PRVCES. O sistema foi desenvolvido na linguagem C++, utilizando o ambiente Microsoft Visual C++, versão 2005, e testado em um computador Intel Core 2 Duo com 1,66 GHz e 2 GB de memória RAM e sistema operacional Windows Vista Home Premium de 32 bits.

Para validar o algoritmo, foram utilizados três conjuntos de problemas-teste mencionados na Seção 1, a saber, os de Salhi e Nagy (1999), Dethloff (2001) e Montané e Galvão (2006). No conjunto de Salhi e Nagy (1999), não foram tratados os problemas que possuem restrições de limite de tempo. O número máximo de iterações do GENILS foi fixado em 10.000.

As Tabelas 2, 3 e 4 comparam o desempenho do GENILS com diferentes algoritmos da literatura. Nessas tabelas, a coluna *Problema* indica o problema-teste considerado, *Melhor* é o melhor valor encontrado pelo algoritmo do respectivo autor e *Tempo* é o tempo, em segundos, de processamento do algoritmo. Na coluna *gap*, mostra-se o desvio percentual das soluções médias do GENILS em relação aos melhores resultados existentes. O *gap* é calculado pela expressão $gap = 100 \times (Média - MelhorValor) / MelhorValor$ (coluna *gap*).

Tabela 2. Resultados obtidos pelo GENILS nos problemas-teste de Dethloff (2001)

Problema	*Röpke e Pisinger (2006)*		*Zachariadis et al. (2010)*		*Subramanian et al. (2008)*		*GENILS*		
	Melhor	*Tempo[(1)] (s)*	*Melhor*	*Tempo[(2)] (s)*	*Melhor*	*Tempo[(3)] (s)*	*Melhor*	*Tempo[(4)] (s)*	*gap (%)*
SCA3-0	636,10	232,00	**635,62**	2,50	**635,62**	0,90	**635,62**	6,77	0,00
SCA3-1	**697,80**	170,00	**697,84**	2,50	**697,84**	1,12	**697,84**	8,49	0,00
SCA3-2	**659,30**	160,00	**659,34**	2,90	**659,34**	1,19	**659,34**	8,13	0,00
SCA3-3	680,60	182,00	**680,04**	2,30	**680,04**	1,13	**680,04**	8,45	0,00
SCA3-4	**690,50**	160,00	**690,50**	2,90	**690,50**	1,32	**690,50**	8,09	0,00
SCA3-5	**659,90**	178,00	**659,90**	3,00	**659,90**	1,17	**659,90**	8,19	0,00
SCA3-6	651,10	171,00	**651,09**	3,10	**651,09**	1,23	**651,09**	8,21	0,00
SCA3-7	666,10	162,00	**659,17**	2,80	**659,17**	1,69	**659,17**	6,76	0,00
SCA3-8	**719,50**	157,00	**719,47**	3,50	**719,47**	1,08	**719,48**	8,85	0,00
SCA3-9	**681,00**	167,00	**681,00**	4,70	**681,00**	1,03	**681,00**	8,63	0,00
SCA8-0	975,10	98,00	**961,50**	2,70	**961,50**	2,52	**961,50**	5,65	0,00
SCA8-1	1052,40	95,00	**1049,65**	3,80	**1049,65**	2,98	**1049,65**	5,67	0,00
SCA8-2	1039,60	83,00	**1039,64**	3,90	**1039,64**	3,42	**1039,64**	5,92	0,00
SCA8-3	991,10	94,00	**983,34**	2,60	**983,34**	3,44	**983,34**	4,58	0,00
SCA8-4	**1065,50**	84,00	**1065,49**	2,40	**1065,49**	2,74	**1065,49**	5,98	0,00
SCA8-5	**1027,10**	96,00	**1027,08**	3,40	**1027,08**	3,44	**1027,08**	6,62	0,00
SCA8-6	972,50	93,00	**971,82**	2,70	**971,82**	2,48	**971,82**	6,57	0,00
SCA8-7	1061,00	92,00	**1051,28**	5,10	**1051,28**	5,39	**1051,28**	5,56	0,00
SCA8-8	**1071,20**	85,00	**1071,18**	3,60	**1071,18**	2,05	**1071,18**	5,57	0,00
SCA8-9	**1060,50**	86,00	**1060,50**	4,80	**1060,50**	3,10	**1060,50**	5,62	0,00
CON3-0	**616,50**	171,00	**616,52**	4,70	**616,52**	2,02	**616,52**	6,77	0,00
CON3-1	**554,50**	190,00	**554,47**	2,20	**554,47**	1,83	**554,47**	7,76	0,00
CON3-2	521,40	176,00	**518,00**	3,10	**518,00**	2,10	**518,00**	9,28	0,00
CON3-3	**591,20**	177,00	**591,19**	3,20	**591,19**	1,34	**591,19**	9,18	0,00
CON3-4	**588,80**	173,00	**588,79**	2,30	**588,79**	1,79	**588,79**	6,29	0,00
CON3-5	**563,70**	179,00	**563,70**	3,70	**563,70**	1,71	**563,70**	9,16	0,00
CON3-6	**499,10**	195,00	**499,05**	3,70	**499,05**	1,93	**499,05**	7,33	0,00
CON3-7	**576,50**	226,00	**576,48**	1,90	**576,48**	1,52	**576,48**	6,96	0,00
CON3-8	**523,10**	174,00	**523,05**	3,80	**523,05**	1,51	**523,05**	8,75	0,00
CON3-9	**578,20**	163,00	**578,25**	2,20	**578,25**	1,58	**578,25**	6,87	0,00
CON8-0	**857,20**	86,00	**857,17**	4,40	**857,17**	3,74	**857,17**	6,36	0,00
CON8-1	**740,90**	81,00	**740,85**	3,30	**740,85**	2,82	**740,85**	4,88	0,00
CON8-2	716,00	84,00	**712,89**	2,70	**712,89**	2,46	**712,89**	6,95	0,00
CON8-3	**811,10**	91,00	**811,07**	2,80	**811,07**	2,82	**811,07**	5,87	0,00
CON8-4	**772,30**	87,00	**772,25**	2,80	**772,25**	3,37	**772,25**	5,01	0,00
CON8-5	755,70	94,00	**754,88**	5,70	**754,88**	3,30	**754,88**	5,82	0,00
CON8-6	693,10	96,00	**678,92**	3,40	**678,92**	3,04	**678,92**	5,67	0,00
CON8-7	814,80	94,00	**811,96**	2,50	**811,96**	2,73	**811,96**	4,71	0,00
CON8-8	774,00	94,00	**767,53**	3,20	**767,53**	3,42	**767,53**	5,23	0,00
CON8-9	809,30	92,00	**809,00**	3,80	**809,00**	3,60	**809,00**	5,86	0,00

(1) Tempo de CPU em um computador Pentium IV 1.5 GHz.
(2) Tempo de CPU em um computador T5500 1.66 GHz.
(3) Tempo de CPU em um computador Intel Core 2 Quad 2.5 GHz.
(4) Tempo de CPU em um computador Intel Core 2 Duo 1,6 GHz.

Tabela 3. Resultados obtidos pelo GENILS nos problemas-teste de Salhi e Nagy (1999)

Problema	*Wassan et al. (2007)*		*Zachariadis et al. (2010)*		*Subramanian et al. (2008)*		*GENILS*		
	Melhor	*Tempo[(1)] (s)*	*Melhor*	*Tempo[(2)] (s)*	*Melhor*	*Tempo[(3)] (s)*	*Melhor*	*Tempo[(4)] (s)*	*gap (%)*
CMT1X	468,30	48	469,80	2,10	**466,77**	1,10	**466,77**	7,82	0,00
CMT1Y	**458,96**	69	469,80	3,80	466,77	1,08	466,77	7,61	1,68
CMT2X	**668,77**	94	684,21	5,40	684,21	6,99	684,21	17,62	2,31
CMT2Y	**663,25**	102	684,21	8,02	684,21	5,84	684,21	20,10	3,16
CMX3X	729,63	294	**721,27**	6,80	721,40	6,80	721,40	59,61	0,02
CMT3Y	745,46	285	**721,27**	11,90	721,40	7,37	**721,27**	58,72	0,00
CMT12X	**644,70**	242	662,22	11,00	662,22	8,02	662,22	22,89	2,72
CMT12Y	**659,52**	254	662,22	9,30	662,22	7,32	663,50	22,33	0,60
CMT11X	861,97	504	**838,66**	4,80	839,39	12,58	846,23	48,85	0,90
CMT11Y	**830,39**	325	837,08	21,20	841,88	14,80	836,04	287,30	0,68
CMT4X	876,50	558	**852,46**	14,40	**852,46**	50,72	**852,46**	134,26	0,00
CMT4Y	870,44	405	852,46[a]	29,60	852,46	46,06	862,28	266,76	1,17[b]
CMT5X	1044,51	483	**1030,55**	27,40	**1030,55**	53,51	1033,51	768,94	0,29
CMT5Y	1054,46	533	**1030,55**	62,80	1031,17	58,74	1036,14	398,75	0,54

(1) Tempo de CPU em um computador Sun-Fire-V440 com um processador UltraSPARC-IIIi 1062 MHz.;
(2) Tempo de CPU em um computador T5500 1.66 GHz.
(3) Tempo de CPU em um computador Intel Core 2 Quad 2,5 GHz.;
(4) Tempo de CPU em um computador Intel Core 2 Duo 1,6 GHz.
[a] Um resultado melhor de valor 852,35 foi obtido por Chen e Wu (2006).
[b] *gap* em relação ao valor encontrado por Chen e Wu (2006).

Tabela 4. Resultados obtidos pelo GENILS nos problemas-teste de Montané e Galvão (2006)

Problema	*Montané e Galvão (2006)*		*Zachariadis et al. (2010)*		*Subramanian et al. (2008)*		*GENILS*		
	Melhor	*Tempo[(1)] (s)*	*Melhor*	*Tempo[(2)] (s)*	*Melhor*	*Tempo[(3)] (s)*	*Melhor*	*Tempo[(4)] (s)*	*gap (%)*
r101	1042,62	12,20	**1009,95**	5,80	1010,90	10,51	**1009,95**	35,65	0,00
r201	671,03	12,02	**666,20**	7,90	**666,20**	6,24	**666,20**	39,62	0,00
c101	1259,79	12,07	1220,99	8,80	1220,26	12,73	**1220,18**	18,34	-0,01
c201	666,01	12,40	**662,07**	4,30	**662,07**	4,18	**662,07**	16,62	0,00
rc101	1094,15	12,30	**1059,32**	14,10	**1059,32**	9,48	**1059,32**	12,79	0,00
rc201	674,46	12,07	**672,92**	10,60	**672,92**	4,21	**672,92**	24,03	0,00
r1_2_1	3447,20	55,56	3376,30	45,10	3371,29	95,79	**3357,64**	175,81	-0,40
r2_2_1	1690,67	50,95	**1665,58**	49,40	**1665,58**	24,13	**1665,58**	103,44	0,00
c1_2_1	3792,62	52,21	3643,82	41,00	3640,20	95,17	**3636,74**	117,62	-0,10
c2_2_1	1767,58	65,79	**1726,59**	56,40	1728,14	41,94	**1726,59**	127,81	0,00
rc1_2_1	3427,19	58,39	3323,56	51,30	3327,98	76,30	**3312,92**	299,30	-0,32
rc2_2_1	1645,94	52,93	1560,34	28,10	**1560,00**	34,28	**1560,00**	77,48	0,00
r1_4_1	10027,81	330,42	9691,60	345,30	9695,77	546,39	**9627,43**	2928,31	-0,66
r2_4_1	3685,26	324,44	3606,72	273,60	**3574,86**	231,73	3582,08	768,60	0,20
c1_4_1	11676,27	287,12	11179,36	224,80	11124,30	524,35	**11098,21**	1510,44	-0,23
c2_4_1	3732,00	330,20	**3549,27**	238,20	3575,63	293,18	3596,37	569,01	1,33
rc1_4_1	9883,31	286,66	9645,27	160,70	9602,53	550,90	**9535,46**	2244,18	-0,70
rc2_4_1	3603,53	328,16	3423,62	315,70	**3416,61**	291,15	3422,11	3306,84	0,16

(1) Tempo de CPU em um computador Athlon XP 2.0 GHz.
(2) Tempo de CPU em um computador T5500 1.66 GHz.
(3) Tempo de CPU em um computador Intel Core 2 Quad 2.5 GHz.
(4) Tempo de CPU em um computador Intel Core 2 Duo 1,6 GHz.

Em relação aos 40 problemas-teste de Dethloff (2001), o GENILS foi capaz de alcançar todas as melhores soluções da literatura.

Entre os 14 problemas-teste de Salhi e Nagy (1999), o GENIUS alcançou três melhores soluções da literatura e ficou distante em até 3,16% nos demais problemas-teste. Apesar desse desempenho inferior, é importante ressaltar que neste conjunto não existe uma dominância clara de nenhum algoritmo da literatura. De fato, o algoritmo de Wassan *et al*. (2007) alcançou as melhores soluções conhecidas em 6 casos, o de Zachariadis *et al.* (2010) em 6 e o de Subramanian *et al.* (2008) em 3.

Dos 18 problemas-teste de Montané e Galvão (2006), o GENIUS é o algoritmo com melhor desempenho em termos de qualidade de solução. Com efeito, o algoritmo proposto detém as 15 melhores soluções conhecidas desse conjunto, das quais 7 são novas, e nas restantes, o *gap* foi inferior a 1,33%. Por outro lado, nesse conjunto, os algoritmos de Zachariadis *et al.* (2010) e de Subramanian *et al.* (2008) só alcançaram 8 das melhores soluções conhecidas.

Comparando-se os algoritmos com relação à capacidade de alcançar as melhores soluções nos 3 conjun-

tos consagrados de problemas-teste da literatura, ou seja, em 72 problemas-teste, observa-se que o GENIUS tem os melhores resultados em 58 casos, o de Zachariadis *et al.* (2010) em 54 e o de Subramanian *et al.* (2008) em 51.

Comparando-se os resultados dos algoritmos entre si no segundo e terceiro conjunto de problemas-teste, observa-se que: (a) No conjunto de Dethloff (2001), os algoritmos GENIUS, de Subramanian *et al.* (2008) e o de Zachariadis *et al.* (2010) ficam empatados em termos de qualidade de solução final produzida, já que todos alcançaram as melhores soluções existentes; (b) No conjunto de problemas-teste de Salhi e Nagy (1999), o GENIUS vence o algoritmo de Subramanian *et al.* (2008) em 2 casos e perde em 5, e vence o algoritmo de Zachariadis *et al.* (2010) em 2 casos e perde em 6; (c) No conjunto de problemas-teste de Montané e Galvão (2006), o algoritmo GENIUS supera o de Subramanian *et al.* (2008) em 9 casos e perde em outros 3, e vence o algoritmo de Zachariadis *et al.* (2010) em 10 casos e perde somente em 1.

Uma comparação em termos de tempos computacionais não foi feita porque os resultados dos outros algoritmos da literatura foram obtidos em máquinas distintas e os códigos-fonte deles não foram disponibilizados para comparação nesse critério.

5. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho tratou o Problema de Roteamento de Veículos com Coleta e Entrega Simultânea (PRVCES). Para resolvê-lo, foi proposto um algoritmo heurístico híbrido, denominado GENILS, que combina as técnicas heurísticas *Iterated Local Search* (Stützle *et al*., 1999), *Variable Neighborhood Descent* (Mladenović e Hansen, 1997) e adaptações das heurísticas GENIUS (Gendreau *et al.*, 1992) e Inserção Mais Barata.

As adaptações das heurísticas GENIUS e Inserção Mais Barata são usadas para gerar uma solução inicial. Para refiná-la, foi utilizada a metaheurística *Iterated Local Search* (ILS), tendo o *Variable Neighborhood Descent* (VND) como método de busca local. O VND desenvolvido difere de sua versão clássica em três aspectos: (1) a ordem de exploração das vizinhanças é definida aleatoriamente; (2) em cada vizinhança encontra-se apenas o melhor vizinho da solução corrente e não um ótimo local e (3) caso o melhor vizinho represente uma melhora, é realizada uma fase de intensificação com a aplicação de uma busca local apenas nas rotas modificadas, com ordem fixa de exploração dessas vizinhanças e usando um conjunto maior de procedimentos de busca local. Mais especificamente, a fase VND do GENIUS inicialmente ordena, de forma aleatória, as vizinhanças baseadas nos movimentos *Shift*, *Shift*(2,0), *Swap*, *Swap*(2,1), *Swap*(2,2), *M2-opt* e *kOr-opt*. Posteriormente, se na *t*-ésima vizinhança o melhor vizinho for de melhora em relação à solução corrente, então sobre as rotas modificadas desse vizinho é aplicada uma fase de intensificação. A intensificação consiste em realizar buscas locais nas sete vizinhanças apresentadas acima, na ordem dada, seguidas da aplicação de buscas locais baseadas nos procedimentos *G3-opt*, *G4-opt* e *reverse*, também nessa ordem.

De acordo com os resultados obtidos, verifica-se que o algoritmo GENIUS proposto é o que detém o maior número de melhores resultados, considerando-se três conjuntos consagrados de 72 problemas-teste da literatura. Comparando-o com cada um dos melhores algoritmos para o PRVCES, observa-se que ele ganha deles em quantidade de melhores soluções produzidas. Além disso, o GENIUS obteve soluções com variabilidade inferior a 1% em 66 dos 72 problemas-teste, o que corresponde a 92% dos casos. Um comportamento interessante do algoritmo GENILS é o fato de ele ter o melhor desempenho nos problemas-teste de maior porte, os de Montané e Galvão (2006), o que mostra o seu potencial em resolver aplicações reais, nas quais geralmente se defronta com problemas de dimensões maiores.

Como trabalhos futuros, pretende-se aprimorar os procedimentos *G3-opt* e *G4-opt* de forma a considerar a recombinação de múltiplas rotas. Além disso, pretende-se combinar o algoritmo GENILS com a metaheurística Busca Tabu, sendo esta acionada em substituição ao VND, por exemplo, após certo número de iterações do ILS. Isso se deve ao fato de que a Busca Tabu é o algoritmo base de Wassan *et al*. (2007) e Zachariadis *et al.* (2010), os quais têm a maioria dos melhores resultados dos problemas-teste de Salhi e Nagy (1999), nos quais o GENILS teve o pior desempenho.

AGRADECIMENTOS

Os autores agradecem à CAPES, CNPq, FAPEMIG e FAPERJ pelo apoio ao desenvolvimento deste trabalho. Também são gratos aos revisores anônimos pelos comentários construtivos, que guiaram a uma versão melhorada deste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

Anderberg, M. R. (2007) *Cluster analysis for applications.* Monographs and Textbooks on Probability and Mathematical Statistics. Academic Press, Inc., New York.

Bean, J. C. (1994) Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154-160.

Bianchessi, N. e G. Righini (2007) Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers & Operations Research*, 34(2):578-594.

Chen, J. F. (2006) Approaches for the vehicle routing problem with simultaneous deliveries and pickups. *Journal of the Chinese Institute of Industrial Engineers*, 23(2):141-150.

Chen, J. F. e T. H. Wu (2006) Vehicle routing problem with simultaneous deliveries and pickups. *Journal of the Operational Research Society*, 57(5):579-587.

Crispim, J. e J. Brandão (2005) Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls. *Journal of the Operational Research Society*, 56(7):1296-1302.

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

Dell'Amico, M.; G. Righini e M. Salanim (2006) A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transportation Science*, 40(2):235-247.

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

Dorigo, M.; V. Maniezzo e A. Colorni (1996) The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, v 26, p 29-41.

Dueck, G. (1993) New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104:86-92.

Gehring, H e J. Homberger (1999) A parallel hybrid evolutionary metaheristic for the vehicle routing problem with time windows. In: Miettinen K, Mäkelä M, Toivanen J, editors. *Proceedings of EUROGEN99*, v A2(S), Springer, Berlin, p 57-64.

Gendreau, M.; A. Hertz e G. Laporte (1992) New insertion and post optimization procedures or the traveling salesman problem. *Operations Research*, 40:1086-1094.

Glover, F. e M. Laguna (1992) *Tabu Search*, Kluwer Academic Publishers, Boston.

Gökçe, E. I. (2004) A revised ant colony system approach to vehicle routing problems. Masther's Thesis, Graduate School of Engineering and Natural Sciences, Sabanci University, Turkey.

Goldberg, D. E. (1989) Genetic Algorithms in Search. *Optimization and Machine Learning*. Addison-Wesley, Berkeley.

Halse, K. (1992) *Modeling and solving complex vehicle routing problems*. PhD thesis, Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, Denmark.

Hansen, P. e N. Mladenović (2001) Variable neighborhood search: Principles and applications, *European Journal of Operations Research*, 130:449-467.

Kirkpatrick, S; D. C. Gellat e M. P. Vecchi (1983) Optimization by Simulated Annealing, *Science*, 220:671–680.

Min, H. (1989) The multiple vehicle routing problems with simultaneous delivery and pick-up points. *Transportation Research A*, 23(5):377-386.

Mine, M. T. (2009) Um algoritmo heurístico híbrido para o problema de roteamento de veículos com coleta e entrega simultânea. Dissertação de mestrado, Programa de Pós-Graduação em Computação, Universidade Federal Fluminense, Niterói. Disponível em http://www.ic.uff.br/PosGraduacao/Dissertacoes/416.pdf.

Mine, M. T.; M. S. A. Silva; L. S. Ochi e M. J. F. Souza (2010) O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via Iterated Local Search e GENIUS. *Transporte em transformação XIV: trabalhos vencedores do prêmio CNT de Produção Acadêmica 2009*. Brasília: Gráfica e Editora Positiva Ltda., p. 59-78.

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

Montané, F. A. T. e R. D. Galvão (2006) A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers and Operations Research*, 33(3):595-619.

Nagy, G. e S. Salhi (2005) Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162:126-141.

Or, I. (1976) *Traveling salesman-type combinational problems and their relation to the logistics of blood banking*. Tese de doutorado, Northwestern University, Evanston, USA.

Parragh, S.; K. Doerner e R. Hartl (2008a) A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21-51.

Parragh, S.; K. Doerner e R. Hartl (2008b) A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(2):81-117.

Rego, C. e C. Roucairol (1996) *Meta-Heuristics Theory and Applications*, chapter A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem, p. 661-675. Kluwer Academic Publishers, Boston.

Röpke, S. e D. Pisinger (2006) A unified heuristic for a large class of vehicle routing problems with backhauls. Technical Report 2004/14, University of Copenhagen, Denmark.

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

Shaw, P. (1998) Using constraint programming and local search methods to solve vehicle routing problems. *In Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming* (*CP-98*), p. 417-431, London.

Solomon, M. M.; I. Joachim; J. Desrosiers; Y. Dumas e D. Villeneuve (1995) A request clustering algorithm for door-to-door handicapped transportation, *Transportation Science*, 29:63-78.

Stützle, T. e H. H. Hoos (1999) Analyzing the run-time behaviour of iterated local search for the tsp. In *Proceedings of the Third Metaheuristics International Conference*, p 449-453, Angra dos Reis, Rio de Janeiro.

Subramanian, A. (2008) Metaheurística *Iterated Local Search* aplicada ao problema de roteamento de veículos com coleta e entrega simultânea. Dissertação de mestrado, Universidade Federal da Paraíba, João Pessoa.

Subramanian, A.; L. A. F. Cabral e L. S. Ochi (2008) An efficient ILS heuristic for the vehicle routing problem with simultaneous pickup and delivery. Relatório Técnico, Universidade Federal Fluminense, disponível em http://www.ic.uff.br/~satoru/ index.php?id=2.

Topcuoglu, H. e C. Sevilmis (2002) Task scheduling with connecting objectives. *In* Yakhno, T. M. (ed.), *Lecture Notes in Computer Science*, 2457: 346-355.

Voudouris, C. e E. Tsang (1996) Partial constraint satisfaction problems and guided local search. *In Proceedings of the Second International Conference on the Practical Application of Constraint Technology* (*PACT'96*), p 337-356.

Vural, A. V. A (2003) GA based meta-heuristic for capacited vehicle routing problem with simultaneous pick-up and deliveries. Masther's Thesis, Graduate School of Engineering and Natural Sciences, Sabanci University, Turkey.

Wassan, N. A.; A. H. Wassan e G. Nagy (2007) A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries. *Journal of Combinatorial Optimization*, 15(4):368-386.

Zachariadis, E. E.; C. D. Tarantilis e C. T. Kiranoudis (2009) A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. *Expert Systems with Applications*, 36(2):1070-1081.

Zachariadis, E. E.; C. D. Tarantilis e C. T. Kiranoudis (2010) An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries. *European Journal of Operational Research*, 202:401-411.