REVISTA INTERDISCIPLINAR DE PESQUISA EM ENGENHARIA

CILAMCE 2016

XXXVII IBERIAN LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING
BRASÍLIA - DF - BRAZIL

SOLUÇÃO DO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELA DE TEMPO VIA ITERATED GREEDY SEARCH

Aguinaldo Alves Pinto

Sérgio Ricardo de Souza

aguinaldoalves.pinto@gmail.com

sergio@dppg.cefetmg.br

Centro Federal de Educação Tecnológica de Minas Gerais

Av. Amazonas 7675, Nova Gameleira, Belo Horizonte, 30510-000, MG, Brasil

Abstract. *Neste artigo é tratado o Problema de Roteamento de Veículos com Janela de Tempo. O objetivo é atender um conjunto de clientes geograficamente distribuídos com um número de veículos limitado. É considerado um único depósito, onde os veículos partem e retornam após visitar todos os clientes de sua respectiva rota. Há uma janela de tempo associada ao depósito, que indica seu período de funcionamento, além de uma janela de tempo pertencente a cada cliente, que indica o intervalo de tempo para iniciar o atendimento. Todos os veículos possuem a mesma capacidade de carga e há uma demanda correspondente a cada cliente. O algoritmo proposto combina a meta-heurística Greedy Randomized Adaptive Search Procedure (GRASP) e a Push-Forward Insertion Heuristic (PFIH) para gerar a solução inicial e é aplicado uma busca local para refinar a solução gerada através da meta-heurística Variable Neighborhood Descent (VND). Posteriormente, esta solução é refinada pela meta-heurística Iterated Greedy Search (IGS) de forma iterativa. A análise de resultados consiste em comparar as 56 instâncias propostas por Solomon (1987) com os melhores resultados da literatura.*

Keywords: *Problema de Roteamento de Veículos com Janela de Tempo, GRASP, IGS, VND*

1 INTRODUÇÃO

O Problema de Roteamento de Veículos com Janela de Tempo - PRVJT é uma variante do problema clássico proposto por Solomon (1987). Possui como características principais estabelecer o atendimento de clientes geograficamente distribuídos a partir de um único depósito. Para cada cliente é estabelecida uma restrição temporal, que determina o tempo inicial e o tempo máximo para que o veículo inicie o atendimento. O PRVJT possui restrições relativas à capacidade máxima dos veículos, à demanda associada a cada cliente e ao tempo máximo em que um veículo deve concluir uma rota e retornar ao depósito.

Vários abordagens têm produzido boas soluções ao PRVJT. Estudos foram apresentados por, dentre outros, Wang (2009), Deng et al. (2013), Pierre and Zakaria (2014) e Sripriya et al. (2015) utilizando algoritmos genéticos. Mao and Deng (2010) aborda o problema através de algoritmos evolucionários. Já Hifi and Wu (2014) aplica uma metaheurística híbrida. Já Sabar et al. (2015) utiliza uma mistura de programação matemática com método heurístico. Técnicas abordando o algoritmo *Iterated Greedy Search* (IGS) em variações do problema foram mostradas em Nucamendi et al. (2015), Yu and Lin (2015) e Yang (2015), obtendo resultados relevantes para a literatura.

A pesquisa do PRVJT vem se destacando pela necessidade das corporações em atender os clientes de forma eficiente, principalmente nas grandes cidades, na quais há restrições de horários para carga e descarga de mercadorias e de tráfego de veículos de grande porte ao longo do dia, dentre outras. Portanto, uma solução que promova a otimização dos itinerários dos veículos e uma disposição dos clientes de forma eficiente se faz cada vez mais relevante.

O PRVJT tenta atender aos clientes empregando a menor quantidade de veículos possível. Os clientes são dispostos em rotas e ordens diferentes, tendo como objetivo encontrar o melhor itinerário para o veículo, diminuindo, assim, a distância percorrida.

Este trabalho tem como objetivo, dado um conjunto de clientes, elaborar rotas de forma a reduzir o número de veículos empregados para atender os clientes, aplicando as instâncias de Solomon Solomon (1987) e a Metaheurística IGS, combinada ao GRASP como técnica de construção e VND como busca local. A Seção 2 apresenta a descrição do problema; a Seção 3 mostra os métodos utilizados para a resolução do problema; a Seção 4 apresenta os resultados computacionais; e a Seção 5 apresenta a conclusão do trabalho.

2 DESCRIÇÃO DO PROBLEMA

O problema de roteamento de veículos foi proposto por Dantzig (1959) como uma variação do problema do caixeiro viajante. Trata-se de um problema NP-Difícil (Toth and Vigo; 1998; Wang; 2009) e objeto de estudos dentro da área de otimização combinatória.

No PRVJT, tem-se um conjunto de veículos com capacidades iguais em relação à carga, partindo de um dado depósito, com uma localização geográfica definida, e uma limitação temporal, que representa a abertura e o fechamento do mesmo. Com isso, cada rota é composta de um veículo, que percorre um determinado número de clientes e retorna para o depósito, respeitando o limite imposto pela janela de tempo. Para cada cliente está associada uma janela de tempo, que determina o início do atendimento. Caso o veículo chegue antes da abertura da janela de tempo, o veículo deve aguardar a abertura da mesma. A cada cliente está relacionado um tempo de serviço, que delimita o tempo necessário para que o cliente seja atendido.

O PRVJT é considerado um problema NP-Difícil, como mostrado em Solomon (1987), e pode ser representado por um grafo completo e não direcionado $G = (V, A)$, tendo um conjunto V de N vértices, sendo o vértice 0 correspondente ao depósito, e um conjunto A de arestas, de modo que $V = \{1, 2, \ldots, N\}$ e $A = \{(i, j) \quad : \quad i, j \in V,\ i \neq j\}$. Cada vértice i apresenta um tempo de serviço ts_i, que representa o tempo gasto pelo veículo para realizar o atendimento ao cliente, e uma demanda c_i, relacionada à quantidade de mercadoria solicitada pelo cliente. As janelas de tempo (e_i, l_i) representam, respectivamente, o início e o fim do período de tempo determinado para iniciar o atendimento do cliente i. Além disso, a janela de tempo associada ao vértice 0, dada por (e_0, l_0), mostra o espaço de tempo correspondente ao funcionamento do depósito. A matriz $D = (d_{ij})$ representa a distância euclidiana entre os vértices (i, j), considerada no cálculo do custo da viajem realizada pelo veículo.

Segundo Solomon (1987), o PRVJT possui, dado um conjunto de m veículos de capacidade homogênea, o objetivo de encontrar o custo mínimo para atendimento aos clientes, utilizando o menor número de veículos. Pode-se, assim, considerar que, para avaliar quão boa é uma solução, é necessário utilizar dois critérios: o número de rotas e a distancia ou custo total para atender todos os clientes de uma determinada instância. Cada cliente deve ser atendido por um único veículo.

3 METODOLOGIA

A resolução do PRVJT via heurísticas e meta-heurísticas de busca local e populacional tem obtido resultados relevantes para a literatura. Segundo Blum et al. (2011), meta-heurísticas são técnicas que permitem gerar soluções de boa qualidade em problemas de otimização combinatória. As meta-heurísticas se diferenciam dos métodos heurísticos por proporcionarem técnicas de saída de ótimos locais e permitirem a sua utilização em diversos grupos de problemas.

Neste trabalho, são apresentados os resultados obtidos aplicando a metaheurística IGS combinada com a metaheurística GRASP para construção da solução inicial e a metaheurística VND como busca local do método.

3.1 *Push-Forward Insertion Heuristic* (PFIH)

No contexto do PRVJT, diversas propostas foram apresentadas ao longo dos anos para a construção de soluções viáveis e reduzir tanto o numero de veículos empregados quanto a distância percorrida. A PFIH foi uma dessas técnicas, no qual é estipulado um custo de inserção, que representa o quanto a solução vai ser beneficiada, caso o cliente seja inserido em determinada posição.

Esta heurística foi elaborada por Solomon (1987) e aplicada por diversos autores em variações do PRVJT que possuem janela de tempo. A heurística fundamenta-se em uma função de custo em que os clientes são dispostos em ordem crescente, ou seja, os clientes que têm custo de inserção menor são selecionados primeiro e, a partir dessa ordem definida, é realizada a tentativa de inserção desses clientes na solução. A função de custo 1 é definida por:

$$C_i = -\alpha \cdot d_{0i} + \beta \cdot l_i + \gamma \cdot [(\frac{pi}{360}) d_{0i}] \tag{1}$$

A expressão C_i é o custo de inserção de determinado cliente; d_{0i} é a distância do cliente em relação ao depósito; l_i é o limite da janela de tempo do cliente; e p_i é o ângulo polar em relação às coordenadas de cada cliente. Assim, a equação considera uma ponderação entre a distância do cliente em relação ao depósito e a janela de tempo do cliente, obtendo um valor que se coloca como boa estimativa para formar rotas com clientes mais próximos, estimulando uma redução na distância total percorrida e no número de rotas. Os parâmetros α, β e γ foram definidos empiricamente por Solomon (1987) como, respectivamente, $\alpha = 0;7$; $\beta = 0,1$; e $\gamma = 0,2$.

3.2 *Greedy Randomized Adaptive Search Procedure* (GRASP)

A metaheurística GRASP foi introduzida por Feo and Resende (1995) e consiste em duas fases: uma fase de construção e uma fase de busca local, executadas sequencialmente por um número de iterações determinado.

A fase de construção consiste em inserir os clientes na solução, observando as restrições, de forma a compor soluções de melhor qualidade. Para isso, no presente caso, é utilizada a função de custo de inserção 1 descrita na seção 3.1. Assim, é aplicada o calculo da inserção de cada cliente e é realizada uma classificação, partindo do cliente de menor custo. Uma vez feita a ordenação, o primeiro cliente é inserido na solução. Em seguida, é formada uma lista restrita de candidatos, de acordo com uma função de avaliação:

$$FA = MaiorCusto - \alpha(MaiorCusto - MenorCusto) \tag{2}$$

através de um parâmetro α, que controla o grau de aleatoriedade do método.

Após o calculo da função 2 , os clientes que obtiverem custos menores ou iguais ao valor de FA são selecionados para a lista restrita de candidatos. Com essa lista formada, é sorteado aleatoriamente um cliente e ele é inserido da seguinte forma:

(i) O cliente é inserido em todas as posições possíveis da solução;

(ii) As posições em que não violarem as restrições de janela de tempo, capacidade do veículo e tempo de viagem são calculadas o valor da função objetivo;

(iii) A posição que tiver o melhor valor da função objetivo é selecionada e o cliente é inserido na solução;

(iv) Caso nenhuma posição seja factível, é criada uma nova rota e o cliente é inserido nela.

O algoritmo 1 descreve os passos realizados na construção de uma solução. As linhas de 1 a 3 calculam os custos de inserção dos clientes baseados na função 1, ordenam de forma crescente, criam uma rota e insere o primeiro cliente na solução. As linhas de 4 à 23 realizam o processo de construção, de forma a inserir todos os clientes na solução criada. As linhas 5 à 7 calculam a função de avaliação descrita em 2 e selecionam os clientes com o valor menor ou igual ao FA. Além disso, seleciona aleatoriamente um cliente da lista restrita de candidatos. As linhas de 8 à 15 são responsáveis por percorrer todas as posições factíveis e selecionar a posição que tenha um valor da função objetivo melhor. As linhas de 16 à 21 verificam se há alguma posição factível. Caso haja, esse cliente é inserido na solução; caso contrário é criada uma nova rota e ele é inserido nela.

Na fase de busca local, é realizada uma busca utilizando a metaheurística VND, com o objetivo de explorar o espaço de busca da solução criada na primeira fase do GRASP. Com

Algorithm 1 Pseudo-Código *Construção GRASP*

```
1.  calculaCustosPFIH(c)
2.  ordenaCustosPFIH(c)
3.  CriaRota(c)
4.  Enquanto N < NumClientes
5.    Calcula FA
6.    Constrói a LCR
7.    Seleciona aleatoriamente um cliente da LRC
8.    Para i ← 2 até m
9.      Verifica posições factível
10.     Se existe posição factível
11.       calcula o valor da função objetivo
12.     Se valor função objetivo é melhor
13.       armazenar posição
14.     Fim do Se
15.   Fim do Para
16.     Se existe posição factível melhor
17.       Inserir cliente posição
18.     Senão
19.       Criar nova rota
20.       Inserir cliente na rota criada
21.     Fim do Se
22.     N++
23.   Fim do Enquanto
```

isso, é feita uma análise da vizinhança da solução, obtendo uma melhoria na função objetivo do problema. Após essa busca, é armazenada a melhor solução encontrada. Em seguida, o método realiza novamente uma construção e uma busca local até um numero máximo de iterações sem melhora.

Algorithm 2 Pseudo-Código *Greedy Randomized Adaptive Search Procedure*

```
1.  N=0 Iter
2.  Enquanto N < Iter
3.    s = ConstrucaoGRASP()
4.    s' = BuscaLocalVND(s)
5.      Se s' < s_best
6.        s_best = s'
7.        N = 0
8.      Senão
9.        N++
10.     Fim do Se
11.   Fim do Enquanto
```

O pseudo-código do método é descrito em 2. As linhas 3 e 4 são realizadas uma construção e uma busca local. As linhas entre 5 a 10 verificam se a solução gerada s' é melhor que a melhor solução encontrada s_{best}; caso positivo, a solução s' passa a ser a solução melhor e o contador que controla a quantidade de iterações é zerado. Caso o valor da função objetivo de s' seja pior, o contador é incrementado e o processo continua, até que a quantidade de iterações sem melhora seja obtido.

3.3 Variable Neighborhood Descent - VND

A meta-heurística de busca local VND neste trabalho foi implementada utilizando todas as estruturas de vizinhanças descritos na Seção 3.4. O método inicia, a partir de uma solução, uma série de movimentos que são colocados em uma determinada ordem. Quando um movimento for bem sucedido, ou seja, conseguir melhorar a função objetivo, os movimentos retornam para a primeira. Caso o movimento piore ou mantenha o valor da função objetivo, o movimento é

trocado. Este processo finaliza quando tiverem percorrido todas as estruturas de vizinhanças definidas.

Algorithm 3 Pseudo-Código *Variable Neighborhood Descent*

```
1.  Solução inicial s_0
2.  Definir e estruturas de vizinhança
3.  i ← 1
4.    Enquanto i < e
5.      Aplica movimento e em s'
6.        Se s' < s
7.          s ← s'
8.        Senão
9.          i ← i + 1
10.   fim do Enquanto
```

O algoritmo 3 recebe como entrada uma solução s_0 e também o número de estruturas de vizinhanças que será aplicada a cada iteração. São executadas as estruturas de vizinhanças seguindo a seguinte ordem: os movimentos de busca intra-rota *Exchange, Shift(3),Shift(2), Shift(1)*; depois os movimentos de busca inter-rotas *Shift(1,0), Shift(2,0), Shift(3,0), Swap(2,1)* e *Swap(2,2)* e por último a estratégia de eliminação de rota, todos eles descritos na seção 3.4. As linhas entre 5 e 9 aplicam o movimento em questão à solução corrente e avaliam: se o movimento foi bem sucedido, ou seja, teve o valor da função objetivo reduzido, o algoritmo continua a exploração pela mesma estrutura de vizinhança; caso contrário, ela é modificada. A busca finaliza quando todas as estruturas forem executadas.

3.4 Descrição dos movimentos

Para explorar o espaço de busca do VRPTW são realizados movimentos de realocação e troca, sempre com o objetivo de reduzir o número de rotas e a distância total percorrida. Neste trabalho, são desenvolvidos dez diferentes abordagens para explorar o espaço de busca. Os movimentos implementados intra-rotas são os denominados $Exchange$ e $Shift(k)$ e os movimentos inter-rotas são o do tipo $Shift(k,0)$ e $Swap(k,l)$. Além dessas estruturas, é adotado uma estratégia de modificação da solução denominada de eliminação de rota, que são descritos à seguir

- $Exchange$: efetua uma troca entre dois cliente dentro da mesma rota. São considerados apenas movimentos factíveis, portanto, a mudança só é aplicada quando todas as restrições são atendidas. Todos os clientes são candidatos a troca;
- $Shift(k)$: é feita uma troca de posição simultânea de 1,2 ou 3 clientes adjacentes dentro da mesma rota. Esta alteração na ordem de visitação dos clientes contribuem para a diminuição da distancia total percorrida. Novamente, todos os clientes são candidatos a serem trocados;
- $Shift(k,0)$: consiste em modificar retirando 1,2 ou 3 clientes consecutivos de uma rota e re-inseri-los em outra rota, respeitando as restrições do problema;
- $Swap(k,l)$: efetua uma troca entre dois cliente de uma rota por um cliente de outra rota - $Swap(2,1)$ ou da troca de dois clientes de rotas diferentes - $Swap(2,2)$. Estes movimentos procuram ajustar a ordem de visitação dos clientes, reduzindo a distancia percorrida;

- $EliminaRota$: consiste na tentativa da eliminar de uma rota. Uma rota é selecionada aleatoriamente e os elementos desta rota são transferidos para as outras rotas utilizando movimentos de reinserção.

Os movimentos de troca e reinserção de clientes são utilizados para explorar melhor o espaço de busca e reduzir a distância total percorrida. Já a estratégia de eliminação de rota visa a redução dos custos, uma vez que as rotas são o maior custo do problema, mesmo que a distância percorrida seja maior.

3.5 *Iterated Greedy Search* (IGS)

A metaheurística *Iterated Greedy Search* é uma meta-heurística proposta por Stützle (2007), inicialmente aplicada ao problema de sequenciamento de máquinas paralelas. Esta abordagem tem se mostrado eficiente em diversos problemas de otimização combinatória, como o problema de sequenciamento de máquinas paralelas com tempo de preparação Stützle (2008), escalonamento de máquinas paralelas não relacionadas Ruiz (2010); Rodriguez et al. (2013), problema do caixeiro viajante Tasgetiren et al. (2013) e Karabulut (2014). Há aplicações envolvendo problema do roteamento de veículos de carga automatizada com restrições de bateria Chebbi (2015), dentre outras. No entanto, pelo menos do conhecimento dos autores do presente artigo, não há artigos relatando a aplicação da metaheurística IGS ao PRVJT, o que torna a presente exposição relevante para a literatura.

O algoritmo IGS possui três fases distintas: na primeira fase, chamada de destruição, é retirado um conjunto de clientes, de acordo com um parâmetro definido. Na segunda, é feita uma re-inserção destes clientes na solução, respeitando-se as restrições do problema. Na ultima fase, é realizada uma busca local, visando uma melhor exploração do espaço de busca. Após essas etapas, é armazenada a melhor solução e, caso a solução obtida após as fases do método tiverem um valor da função objetivo pior que a solução de entrada do método (solução inicial), o método contabiliza uma iteração sem melhora, sendo executado até que tenha um determinado número de iterações sem melhora.

Na fase de destruição, é selecionada uma rota aleatoriamente, de forma que todas as rotas têm a mesma probabilidade de serem selecionadas. Após essa seleção, todos os clientes que compõem essa rota são retirados da solução. Com isso, passa-se a ter uma solução parcial. Já na fase de reconstrução, os clientes são dispostos em uma lista, respeitando a ordem de visitação. Para cada cliente da lista é realizada a tentativa de re-inseri-los na solução, desde que respeitem as restrições do problema. Caso algum cliente que estava na lista não seja re-inserido na solução, esses clientes irão compor uma nova rota e inseridos novamente na solução, uma vez que a factibilidade fica garantida entre esses clientes, pois a posição de cada um na rota selecionada na primeira fase é mantida.

O algoritmo 4 usa a metaheurística GRASP para a construção da solução inicial e a busca local VND para as iterações do método.

No algoritmo 4, são definidas k iterações do método sem melhora, a melhor solução S_0 obtida pela meta-heurística GRASP descrita em 2 e uma solução S' com a busca local VND descrita em 3. As linhas entre 5 e 14 mostram o método IGS propriamente dito. Na linha 6, o método de destruição é aplicado, selecionando uma rota aleatoriamente e retirando-a da solução. Na linha 7 é aplicado o procedimento de re-inserção dos clientes retirado da solução, observando as restrições. Neste procedimento, todos os clientes que não encontrem uma outra

posição factível compõem uma nova rota, que será inserida na solução. Na linha 8 é realizada uma busca local VND descrita em 3. As linhas entre 9 e 13 verificam se a solução S'' é melhor que a melhor solução $S*$; caso verdade, a solução perturbada passa a ser a solução melhor; caso contrário, é computado uma iteração sem melhora. O algoritmo finaliza quando um certo número de iterações sem melhora é executado.

4 RESULTADOS

O algoritmo apresentado neste trabalho foi testado com as 56 instâncias de Solomon (1987) com 100 clientes, executadas 30 vezes para cada instância. Os resultados foram comparados com os melhores resultados encontrados na literatura, tanto em número de rotas quanto em distância percorrida.

As instâncias de Solomon (1987) são agrupadas em classes C, R e RC. As instâncias C são de clientes agrupados de forma mais próxima geograficamente. As instâncias R são de clientes dispersos aleatoriamente e as de RC são de clientes distribuídos tanto de forma aleatória quanto de forma agrupada.

O algoritmo foi implementado em linguagem C++, utilizando como compilador o Code-Blocks, versão snv 9455, sistema operacional Linux e executado em um notebook Core i7 com 2.1 GHz e 8.00 GB de memória RAM.

As tabelas 1, 2 3 mostram os resultados obtidos nos testes e os melhores resultados da literatura. Em relação ao número de rotas formadas, os resultados do algoritmo para as instâncias da classe C ficaram 15 rotas distantes dos melhores resultados. As classes R e RC ficaram, respectivamente, 35 e 45 rotas distantes dos melhores resultados da literatura. Em relação à distância percorrida, as instâncias C ficaram 133% acima do total da literatura. Da mesma forma, as instâncias da classe R ficaram 58% e as classes RC com 57% distantes da literatura.

5 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo mostrar uma proposta de solução para o problema de roteamento de veículos com janela de tempo utilizando as meta-heurísticas GRASP e IGS, além do VND como busca local. A hibridização mostrada no trabalho teve como motivador a adaptação do algoritmo IGS para o problema de roteamento de veículos com janela de tempo,

Algorithm 4 Pseudo-Código *Iterated Greedy Search*

```
1.  Definir k iterações sem melhora
2.  Executa Meta-heurística GRASP (S_0)
3.  Realiza a busca local Meta-heurística VND (S')
4.  it ← 0
5.    Enquanto it < k
6.      Destruição (d,S')
7.      Reconstrução (S')
8.      Busca Local VND (S'')
9.        Se S'' < S*
10.         computa solução S* ← S''
11.         it ← 0
12.       Senão
13.         it ← it + 1
14.   fim do Enquanto
```

Tabela 1

Melhores Resultados			Melhores Resultados Literatura	
Instância	**Distância**	**Número de Veículos**	**Número de Veículos**	**Distância**
C101	1206,68	11	10	828,94
C102	1751,46	10	10	828,94
C103	1873,48	11	10	828,06
C104	1798,05	10	10	824,78
C105	2047,07	11	10	828,94
C106	1830	11	10	828,94
C107	1525,94	11	10	828,94
C108	2048,75	12	10	828,94
C109	1810,99	10	10	828,94
C201	1210,83	4	3	591,56
C202	1713,53	4	3	591,56
C203	1697,24	4	3	591,17
C204	1681,24	4	3	590,6
C205	1239,1	4	3	588,88
C206	1896,05	4	3	588,49
C207	1752,94	4	3	588,29
C208	1249,27	4	3	588,32
Total	28332,62	129	114	12174,29

Tabela 2

Melhores Resultados			**Melhores Resultados Literatura**	
Instância	**Distância**	**Número de Veículos**	**Número de Veículos**	**Distância**
R101	2045,89	21	19	1650,8
R102	1868,39	18	17	1486,12
R103	1669,74	14	13	1292,68
R104	1563,97	14	9	1007,31
R105	1884,62	17	14	1377,11
R106	1832,77	15	12	1252,03
R107	1628,25	13	10	1104,66
R108	1424,87	12	9	960,88
R109	1783,39	14	11	1194,73
R110	1562,3	13	10	1118,84
R111	1634,53	14	10	1096,72
R112	1613,38	13	9	982,14
R201	2096,45	5	4	1252,37
R202	1990,44	4	3	1191,7
R203	1705,23	3	3	939,5
R204	1378,74	3	2	825,52
R205	1888,27	4	3	994,42
R206	1795,45	4	3	906,14
R207	1547,22	3	2	890,61
R208	1440,54	3	2	726,82
R209	1758,5	4	3	909,16
R210	1848,16	4	3	939,37
R211	1587,04	3	2	885,71
Total	39548,14	218	173	24985,34

Tabela 3

Melhores Resultados			Melhores Resultados Literatura	
Instância	**Distância**	**Número de Veículos**	**Número de Veículos**	**Distância**
RC101	2132,17	18	14	1696,94
RC102	2017,91	16	12	1554,75
RC103	1756,7	13	11	1261,67
RC104	1618,23	13	10	1135,48
RC105	2143,93	17	13	1629,44
RC106	2012,15	16	11	1424,73
RC107	1692,37	14	11	1230,48
RC108	1575,29	13	10	1139,82
RC201	2397,74	5	4	1406,94
RC202	2136,35	4	3	1365,65
RC203	1928,77	4	3	1049,62
RC204	1715,95	4	3	798,46
RC205	2372,56	5	4	1297,65
RC206	2050,54	4	3	1146,32
RC207	2060,3	4	3	1061,14
RC208	1753,99	4	3	828,14
Total	31364,95	154	118	20027,23

que ainda não teve aplicação relatada na literatura recente. Com isso, o trabalho contribui para o estado da arte do problema.

Em trabalhos futuros, pretende-se adotar outras técnicas de hibridização em conjunto com o IGS, a fim de explorar melhor o espaço de busca e reduzir tanto a distância percorrida quanto o número de rotas geradas para aproximar os resultados com os melhores da literatura.

Agradescimentos

Agradecemos à CAPES e ao CEFET−MG pelo apoio na realização desse trabalho.

REFERÊNCIAS

Blum, C., Puchinger, J., Raidl, G. R. and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey, *Applied Soft Computing* **11**(6): 4135–4151.

Chebbi, Olfa; Chaouachi, J. (2015). Multi-objective iterated greedy variable neighborhood search algorithm for solving a full-load automated guided vehicle routing problem with battery constraints, *Electronic Notes in Discrete Mathematics* **47**.

Dantzig, G. B.; Ramser, J. H. (1959). The truck dispatching problem, *Management Science* **6**.

Deng, Y., Xiang, J. and Ou, Z. (2013). Improvement of genetic algorithm for vehicle routing problems with time windows, *Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference on*, IEEE, pp. 866–869.

Feo, T. and Resende, M. (1995). Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6**: 109–133.

Hifi, M. and Wu, L. (2014). A hybrid metaheuristic for the vehicle routing problem with time windows, *Control, Decision and Information Technologies (CoDIT), 2014 International Conference on*, IEEE, pp. 188–194.

Karabulut, Korhan; Fatih Tasgetiren, M. (2014). A variable iterated greedy algorithm for the traveling salesman problem with time windows, *Information Sciences* .

Mao, Y. and Deng, Y. (2010). Solving vehicle routing problem with time windows with hybrid evolutionary algorithm, *2010 Second WRI Global Congress on Intelligent Systems*, Vol. 1, IEEE, pp. 335–339.

Nucamendi, S., Cardona-Valdes, Y. and Angel-Bello Acosta, F. (2015). Minimizing customer's waiting time in a vehicle routing problem with unit demands, *Journal of Computer and Systems Sciences International* **54**(6): 866–881.

Pierre, D. M. and Zakaria, N. (2014). Partially optimized cyclic shift crossover for multi-objective genetic algorithms for the multi-objective vehicle routing problem with time-windows, *Computational Intelligence in Multi-Criteria Decision-Making (MCDM), 2014 IEEE Symposium on*, IEEE, pp. 106–115.

Rodriguez, F. J., Lozano, M., Blum, C. and García-Martínez, C. (2013). An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem, *Computers & Operations Research* **40**.

Ruiz, L. F.-P. R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling, *European Journal of Operational Research* **207**.

Sabar, N. R., Zhang, X. J. and Song, A. (2015). A math-hyper-heuristic approach for large-scale vehicle routing problems with time windows, *2015 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, pp. 830–837.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints, *Operations Research* **35**.

Sripriya, J., Ramalingam, A. and Rajeswari, K. (2015). A hybrid genetic algorithm for vehicle routing problem with time windows, *Innovations in Information, Embedded and Communication Systems (ICIIECS), 2015 International Conference on*, IEEE, pp. 1–4.

Stützle, R. R. T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *European Journal of Operational Research* **177**.

Stützle, R. R. T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, *European Journal of Operational Research* **187**.

Tasgetiren, M. F., Buyukdagli, O., Kızılay, D. and Karabulut, K. (2013). A populated iterated greedy algorithm with inver-over operator for traveling salesman problem, *in* B. K. Panigrahi, P. N. Suganthan, S. Das and S. S. Dash (eds), *Proceedings of the 4th International Conference on Swarm, Evolutionary, and Memetic Computing (SEMCCO 2013), Chennai, India, December 19-21, 2013*, Springer International Publishing, pp. 1–12.

Toth, P. and Vigo, D. (1998). Exact solution of the vehicle routing problem, *Fleet management and logistics*, Springer, pp. 1–31.

Wang, C.-B. C. K.-P. (2009). Solving a vehicle routing problem with time windows by a decomposition technique and a genetic algorithm, *Expert Systems with Applications* **36**.

Yang, Jun; Sun, H. (2015). Battery swap station location-routing problem with capacitated electric vehicles, *Computers & Operations Research* **55**: 217–232.

Yu, V. F. and Lin, S.-W. (2015). Iterated greedy heuristic for the time-dependent prize-collecting arc routing problem, *Computers & Industrial Engineering* **90**: 54 – 66.