Heurísticas para Agendamento de Produção de uma Máquina Linear e Roteamento de Entrega com Frota Heterogênea de Veículos

Gabriel de Paula Felix and José E. C. Arroyo

Department of Computer Science, Universidade Federal de Viçosa,
Viçosa - MG, 36570-900, Brazil
gabriel.felix@ufv.br, jarroyo@dpi.ufv.br

Abstract. Este artigo aborda um problema de programação multiobjetivo do sequenciamento de produção de uma máquina linear integrado ao roteamento de veículos para entrega. Nesta abordagem as tarefas possuem penalidades de atraso, datas de vencimento e são caracterizadas por diferentes volumes, com uma frota heterogênea e limitada de veículos (HFVRP). Os critérios a serem minimizados são referentes ao atraso total ponderado e também aos custos de transporte e entrega, incluindo rotas escolhidas e veículos utilizados.
O problema é classificado como NP-Difícil no senso comum, portanto heurísticas eficientes são necessárias para obter soluções próximas da ótima em tempo computacional razoável. Neste trabalho são propostas duas heurísticas baseadas em Iterated Local Search e um Algoritmo Genético (AG). Instâncias para o problema são geradas com base em trabalhos anteriores e os resultados obtidos são comparados aos do solver de otimização CPLEX. Os resultados mostram que as heurísticas aplicadas possuem performance em tempo e qualidade superior às soluções do software a medida em que o número de clientes aumenta.
Keywords: Single machine scheduling, vehicle routing, heterogeneous fleet, time windows, linear programming, meta-heuristics.

1 Introdução

O processo de agendamento de produção e roteamento de veículos é uma tarefa que está presente em manufaturas, e possui grande importância econômica por conta dos prazos impostos, custos de entrega e manuseio do tempo gasto. Separadamente, o problema de agendamento em uma máquina linear com minimização do atraso total é provado como NP-Difícil (Du and Leung, 1990). Estudos são realizados na área de pesquisa operacional para integrar os dois problemas. (Mohammad Tamannaei, Morteza Rasti-Barzoki), estudaram o problema minimizando o atraso total e os custos de transporte utilizando tarefas com tamanhos iguais e custos iguais para todos veículos. Foram propostos três algoritmos genéticos (A.G.) e um método exato Branch-and-Bound (B&B).

Este trabalho ao buscar uma representação realista da integração entre o processo de agendamento de produção de uma máquina linear e roteamento de

veículos, considera tarefas de diferentes volumes e uma frota heterogênea e limitada de veículos, i.e., possuem capacidades e custos variados. (Christian Ullrich, 2013) desenvolveu um Algoritmo Genético (A.G.) integrando agendamento de produção de máquinas paralelas e roteamento de uma frota heterogênea, em um ambiente com janelas de tempo de entrega e veículos com diferentes capacidades.

As aplicações para este problema possuem grande abrangência de setores industriais (como o setor quimico, metalúrgico e têxtil), e a crescente pressão nos mercados globais força as empresas à manusearem seus recursos de forma adequada para garantir um bom funcionamento. Há pouco tempo, Hassanzadeh and RastiBarzoki (2017) estudaram a otimização na cadeia de produção considerando o problema de roteamento de veículos com penalidades de atraso e consumo total de recursos.

Nas últimas décadas, o problema do estudo de roteamento de veículos tornou-se uma questão de foco na literatura acadêmica (Braekers, Ramaekers e Van Nieuwenhuyse, 2016). Na abordagem deste artigo, uma vez que os veículos por possuem diferentes custos e capacidades, apresentam uma concepção desafiadora de utilização dos recursos, tornando cruciais as decisões de quais veículos utilizar. M. Gendreau, G. Laporte and C. Musaragayi (1999), propuseram um algoritmo Tabu Search para o problema de Roteamento de Veículos com Frota Heterogênea (HVRP), e foram obtidas soluções de boa qualidade para as instâncias abordadas. Condotta, Knust, Meier, and Shakhlevich (2013) estudaram o problema de minimizar o atraso máximo de uma máquina linear e considerando datas de entrega para os trabalhos, com veículos com mesma capacidade para entrar as tarefas.

Pelo conhecimento dos autores deste artigo, o problema integrado de sequenciamento em uma máquina simples e roteamento com veículos de capacidades e custos variáveis, tarefas de diferentes tamanhos, e com o objetivo de minimizar o atraso ponderado total e custos de entrega, ainda não foi abordado na literatura. São componentes deste trabalho:

- Duas abordagens heurísticas de Iterated Local Search (ILS) e um Algoritmo Genético (G.A.).
- Um modelo de programação linear para solução dos problemas integrados.

2 Descrição do Problema

Nesta seção é apresentado um modelo matemático de programação linear para solução do problema. As informações sobre parâmetros, índices, variáveis de decisão que serão levados em consideração na resolução deste, são descritos abaixo:

Indices:

i, j	Index of jobs.
k	Index of vehicles.
K	Number of vehicles.
N	Number of jobs.

Instance Parameters:

Q_k	Capacity of each vehicle k.
F_k	Cost of each vehicle k.
P_i	Processing time associated to job i.
t_{ij}	Travel time between customer of job i and customer of job j.
w_i	Penalty applied to delivery tardiness of job i.
s_i	Size of job i.
d_i	Due date associated to job i.

Decision Variables:

C_i	Completion time of job i
X_{ij}^k	Binary decision variable which indicates with value '1' that a vehicle k went from customer of job i to customer of job j, and '0' otherwise.
Y_k	Binary variable which represents with value '1' if a vehicle k is used, and '0' otherwise.
S_k	Time of a vehicle k starts to be used.
A_{ij}	Binary variable which represents if a job i is processed before job j, value '1' if it's true, and '0' otherwise.
D_i	Delivery time of job i.
T_i	Tardiness of job i.

Seja uma máquina linear que lidará com N tarefas, isto é, cada tarefa i de tamanho s_i deverá ser processada em sequência a partir do início do processo ($t = 0$). Um cliente ordena somente uma tarefa. Caso uma tarefa i seja processada antes de uma tarefa j na máquina, a variável A_ij possuirá valor '1', ou '0' caso contrário. O tempo de completude de processamento para tarefa i é representado por C_i. Para a entrega das tarefas, são disponibilizados K veículos, realizando circuitos partindo da origem 'O', sendo a distância entre dois clientes i e j representada por t_ij. Na rota percorrida para entrega, caso uma tarefa i seja entregue imediatamente anterior à uma tarefa j e seja entregue por um veículo k, esta será representada pela variável X_{ij}^k com valor '1', ou '0' caso contrário. A distância de trajeto entre dois clientes i e j é representada por t_{ij}. Este conjunto de variáveis combinadas descreve a primeira parte da função objetivo, que minimiza o a distância de entrega percorrida pelos veículos:

$$\min \sum_{i,j=0}^{N} \sum_{k=0}^{K} X_{ij}^k * t_{ij} \tag{1}$$

Cada veículo k possui capacidade Q_k e custo de utilização F_k, além de possuir tempo de ínicio S_k. Caso um veículo k seja utilizado, será representado em Y_k com o valor '1', ou '0' caso contrário. Dessa forma, a segunda parte da função

objetivo é representada pela minimização dos custos dos veículos:

$$\min \sum_{k=0}^{K} F_k * Y_k \tag{2}$$

Cada tarefa i possui tempo de processamento P_i e data de vencimento d_i vinculada, que pode ser penalizada por uma constante w_i multiplicada pelo seu atraso T_i. O tempo de chegada no destinatário é representado por D_i. A variável de atraso T é dada por $D_i - d_i$, i.e., a diferença entre o tempo de chegada e a data de vencimento. Dessa forma, $T_i = \max(0,\ D_i - d_i)$ e a terceira parte da função objetivo é a minimização do atraso total ponderado das entregas:

$$\min \sum_{i=1}^{N} w_i * T_i \tag{3}$$

Os problemas de minimização descritos acima tratam partes específicas do processo de produção ou entrega de uma manufatura. Finalmente, ao somar os três problemas de minimização, é gerado o problema tratado neste artigo, que representa a minimização dos custos de entrega e seu atraso total ponderado:

$$\min \sum_{i,j=0}^{N} \sum_{k=0}^{K} X_{ij}^{k} * t_{ij} + \sum_{k=0}^{K} F_k * Y_k + \sum_{i=1}^{N} w_i * T_i \tag{4}$$

Quando não tratado o Problema de Minimização de Atraso, resta o Problema de Roteamento com Frota Heterogênea de Veículos (HFVRP), uma versão de VRP, que separadamente é provado como strong NP-Hard (Chen,2010). Por consequência, o problema descrito é também de complexidade strong NP-Hard.

3 Representação de Solução

A representação de uma solução é disposta linearmente, de modo que tarefas e veículos sigam acompanhados de seus respectivos códigos de identificação. O conjunto de tarefas N do problema será dividido em K subconjuntos, sendo cada subconjunto destinado à um veículo específico. O tempo de ínicio de funcionamento de um veículo k, denotado por S_k, é dado pela soma dos tempos de processamento P destas tarefas somado ao tempo acumulado das tarefas processadas anteriormente (a partir de $t = 0$). Em cada subconjunto há definição de uma rota, que indica o trajeto percorrido por um veículo. Seja R uma solução viável, $k \in K$ um veículo e R_k sua rota, a representação deste trajeto é dada por:

$$R_k = O \rightarrow J_1 \rightarrow J_2 \rightarrow ... \rightarrow J_{|N_k|} \rightarrow O \tag{5}$$

Neste contexto, Ji são as tarefas entregues no trajeto e $\forall i = 1, ...|N_k| \in N$ e $|N_k|$ a quantidade de tarefas dispostas neste veículo. Todo veículo parte da

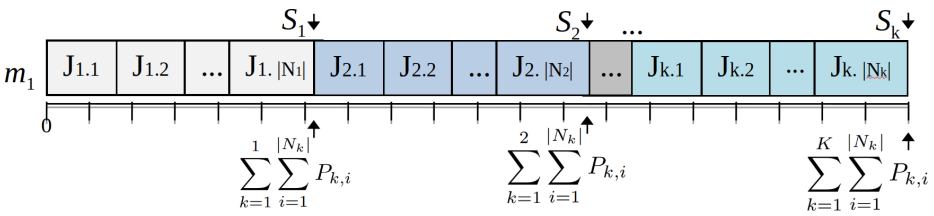

Fig. 1. Representação de solução com tarefas organizadas em lotes.

origem, e deve retornar para ela, visitando cada cliente somente uma vez. Em Figura 1 é apresentada a representação visual de uma solução.

Um subconjunto de tarefas pode não possuir tarefas e ser designado à um veículo w, indicando que este não está em uso e portanto $|N_w| = 0$. Um caso hipotético de rede de transporte é descrito abaixo, no qual a origem e seis clientes (C_1 à C_6) são dispostos cartesianamente, sendo a distância entre eles representada pelas distâncias euclidianas dos pontos (Tabela 1).

Table 1. Euclidian distance (ED) between two points in the network.

ED(x,y)	Origin	C_1	C_2	C_3	C_4	C_5	C_6
Origin	0	27	260	253	254	112	90
C_1	27	0	265	270	237	114	105
C_2	260	265	0	474	212	371	175
C_3	253	270	474	0	507	178	307
C_4	254	237	212	507	0	346	231
C_5	112	114	371	178	346	0	198
C_6	90	105	175	307	231	198	0

A Frota de Veículos é descrita na Tabela 2, onde três veículos heterogêneos(V_1, V_2 e V_3) são disponibilizados para transportar as tarefas. Nela, os preços F dos veículos são diretamente proporcionais às suas capacidades de transporte.

Table 2. Vehicles information: costs and capacities.

Vehicle	Capacity (Q)	Price (F)
V_1	204	1224
V_2	186	1116
V_3	160	960

Por último, as informações sobre as tarefas para a o problema de agendamento de produção são apontadas na Tabela 3, com estas informações a instância se completa.

Table 3. Jobs' detailed parameters.

Job	Customer	Processing Time (P)	Due date (d)	Penalty weight (w)	Job size (s)
J_1	C_1	25	266	1.8	20
J_2	C_2	49	347	2.3	31
J_3	C_3	36	303	4.3	25
J_4	C_4	96	431	4	86
J_5	C_5	52	177	2	33
J_6	C_6	43	315	4.6	42

A Figura 2 descreve visualmente a solução ótima para a instância. Neste caso hipotético, todos os três veículos foram utilizados durante a entrega, possuindo tempos de início S_1,S_2 e S_3 iguais a 180, 301 e 88, respectivamente. Dentre as seis tarefas, apenas J_6 foi entregue sem atrasos.

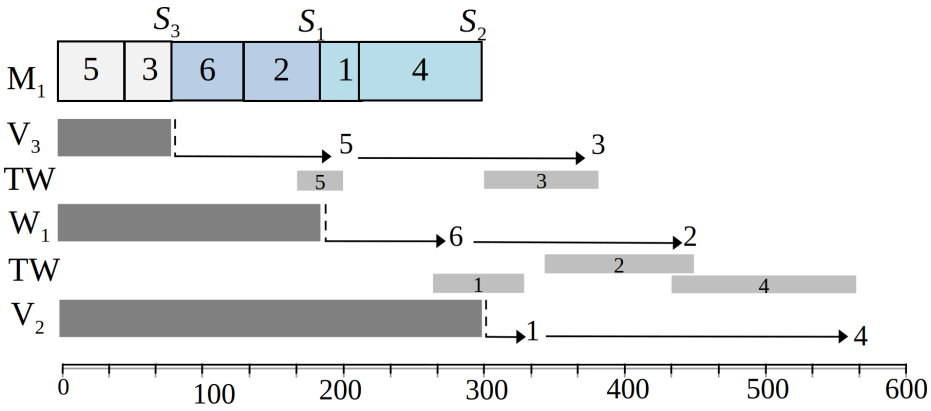

Fig. 2. Gráfico de Gantt com a descrição visual da solução.

As informações sobre a configuração obtida na resposta e sua distribuição temporal são descritas na Tabela 4.

O resultado ótimo da função objetivo possui valor absoluto 6127.5, sendo este composto por três partes integrantes do problema. A primeira, é dada pelos custos de trajeto dos três veículos, que é calculado em $DC = (112+178+253)+(90+175+260)+(27+237+254) = 1586$.

A segunda parte é definida pelos custos de utilização de veículo, que neste contexto será total, uma vez que todos veículos foram utilizados. Desta forma, $VC = 1224 + 1116 + 960 = 3300$.

Os atrasos T serão multiplicados por suas respectivas penalidades w. Isto posto, o custo total de atrasos ponderados é $TC = (62 * 1.8) + (98 * 2.3) + (75 * 4.3) + (134 * 4) + (23 * 2) + (0 * 4.6) = 1241.5$.

Table 4. Optimal Solution Settings.

Job	Customer	Completion time.(C)	Dest. Time (T)	Tardiness (T)
J_1	C_1	276	328	62
J_2	C_2	180	445	98
J_3	C_3	88	378	75
J_4	C_4	301	565	134
J_5	C_5	52	200	23
J_6	C_6	137	270	0

Portanto, ao combinar as três parcelas da função objetivo, é obtido o valor final $DC + VC + TC = 6127.5$ na solução ótima da instância.

4 Abordagem de Solução

São tratadas no artigo três abordagens para solucionar o Problema de Sequenciamento de Produção e Entrega, sendo duas versões da metaheurística Iterated Local Search (ILS) e um Algoritmo Genético (G.A.). Ullrich (2013) mostrou que a aplicação do Algoritmo Genético na variação do problema com mais de uma máquina possibilita bons resultados.

O método de busca local aplicado junto aos algoritmos é o procedimento VND (Mladenovic and Hansen 1997), com ordenação aleatória de vizinhanças (RVND). São descritas duas versões de RVND neste trabalho. Em uma das versões, uma técnica de aprimoramento interno de rotas é aplicada.

A estrutura de ambas Iterated Local Search são similares, diferenciando-se somente na chamada dos métodos de busca local RVND. O Algoritmo Genético (G.A.) desenvolvido utiliza técnicas de Torneio Binário e 2-point Crossover.

4.1 Soluções Iniciais

A qualidade de um ótimo local obtido por método de busca local depende da solução inicial trabalhada (El-Ghazali Talbi, 2009). As regras para a criação de soluções iniciais utilizadas aqui originam-se do problema de Permutation Flow Shop, sendo elas Apparent Tardiness Cost, Weighted Modified Due Date e Weighted Earliest Due Date (L. P. Molina-Sánchez, Eliana María, 2015).

O conjunto de regras têm por objetivo designar uma ordem favorável para o sequenciamento de tarefas. Por se tratar de uma Frota Heterogênea de Veículos, a ordem gerada pelas regras pode ou não ser atendida devido à restrições de capacidade dos veículos. Caso nenhuma das três consigam ser atendidas, uma solução inicial aleatória é utilizada.

4.2 Operadores de Vizinhança

O processo de busca local de uma solução visa aprimorar a atual ao considerar soluções similares do problema que seguem operações de vizinhança definidas.

Neste trabalho, são consideradas apenas soluções viáveis em todo o processo. Matematicamente, uma solução S é descartada se:

$$Q_k < \sum_{i=1}^{|N_k|} s_i; \forall k \in K; \tag{6}$$

Para constituir o conjunto de vizinhanças de uma solução, estas foram subdivididas em dois subconjuntos: vizinhanças Intra-Rota e Inter-Rota.

4.3 Vizinhanças Intra-Rota

Neste subconjunto, as operações aplicadas para diversificação da solução são restritas a alterações internas de um veículo, influenciando somente na ordem de entrega das tarefas carregadas por este. São definidas as operações:

- **Swap-Adj-Job**(S_i) — Troca de posição duas tarefas adjacentes na rota S_i.
- **Insert-Job**(S_i) — Insere uma tarefa nas demais posições da rota S_i.
- **2-opt**(S_i) — Realiza a operação de 2-opt em todas combinações de arcos distintos da rota S_i, i.e., selecionar dois arcos distintos, removê-los e reconectar os clientes de forma que a rota se diferencie e permaneça interligada.

4.4 Vizinhanças Inter-Rota

As operações de vizinhança do tipo Inter-Rota consideram duas ou mais rotas durante sua aplicação, realizando alterações em mais de uma rota ao mesmo tempo. Durante o processo, as operações aplicadas podem gerar soluções inviáveis (por exemplo, ao inserir alguma tarefa em um veículo cheio), porém estas não serão consideradas aqui, para que os algoritmos lidem somente com soluções viáveis.

Seja S uma solução viável para o problema e S_i e S_w as rotas completas realizadas pelos veículos i,$w \in K$. São definidas as operações:

- **Swap-Job**(S_i, S_w) — Trocam de posição duas tarefas $a \in S_i$ e $b \in S_w$ entre as duas rotas, de forma que a receberá a posição antiga de b na nova rota e vice-versa.
- **Insert-Job**(S_i, S_w) — Insere uma tarefa $a \in S_i$ nas demais posições da rota S_w.
- **Swap-Adj-Vehicle**(S_i, S_w) — Trocam de posição na solução dois veículos adjacentes i e w carregando consigo todas suas tarefas designadas, i.e., caso o veículo i parta agora após o veículo w, o primeiro deverá esperar todas as tarefas do segundo serem processadas para que as suas tomem início.
- **Insert-Vehicle**(S_i) — Insere um veículo i juntamente com suas tarefas designadas nas demais posições da solução.

4.5 Abordagens de RVND

O processo de busca local é realizado por VND (Mladenovic and Hansen 1997), utilizando os sete operadores de vizinhança implementados e ordenação aleatória de vizinhanças (RVND). São empregadas duas versões de RVND, com técnicas diferentes de chamada aos operadores.

A primeira versão, descrita em Algorithm 1, define-se em caminhar pelo conjunto de vizinhanças NeighborhoodSet à procura de melhorias na solução atual, e em seguida reiniciar este procedimento quando alguma melhora é alcançada. Do contrário, caso nenhuma melhoria seja obtida na execução, a busca local é encerrada.

Algorithm 1: RVND-Custom(S)

```
   Input  : Solution S
   Output : Better or equal S* solution
 1 S* ← S;;
 2 InterRoute ←Random Order(Inter-Route Neighborhoods);
 3 for i ← 1 to InterRoute do
 4 |   currInter ← InterRoute[i]; // Get inter-route neighborhood
 5 |   S' ← currInter(S); // Apply neighborhood local search
 6 |   if f(S') ≤ f(S) then
 7 |   |   S ← S';
 8 |   |   IntraRoute ←Intra-Route Neighborhoods;
 9 |   |   for j ← 1 to IntraRoute do
10 |   |   |   currIntra ← IntraRoute[j];
11 |   |   |   S ← currIntra(S);
12 |   |   |   if f(S) ≤ f(S') then
13 |   |   |   |   S' ← S;
14 |   |   |   |   j ← 1; // Restarting Intra-Route local search
15 |   |   |   end
16 |   |   end
17 |   |   i ← 1;
18 |   |   InterRoute ←Random Order(Inter-Route Neighborhoods);
19 |   end
20 |   if f(S') ≤ f(S*) then
21 |   |   S* ← S';
22 |   end
23 end
   Output : S*
```

A segunda versão, RVND-Custom, é baseada na técnica de aprimoramento interno de rota publicada por (P.H.V. Penna, L. S. Ochi, 2013), onde as vizinhanças externas à uma rota Inter-Route Neighborhoods são analisadas, e em seguida, ao obter melhora na solução, as vizinhanças internas Intra-Route Neighborhoods são acionadas para aprimorar ainda mais a solução obtida. A aplicação é descrita em Algorithm 2.

Algorithm 2: RVND-Custom(S)

Input : Solution S
Output : Better or equal $S*$ solution

$S* \leftarrow S$;
$InterRoute \leftarrow$Random Order($Inter\text{-}Route\ Neighborhoods$);
for $i \leftarrow 1$ to $InterRoute$ **do**
  $currInter \leftarrow InterRoute[i]$; `// Get inter-route neighborhood`
  $S' \leftarrow$ currInter(S); `// Apply neighborhood local search`
  if f(S') $\leq$ f(S) **then**
    $S \leftarrow S'$;
    $IntraRoute \leftarrow Intra\text{-}Route\ Neighborhoods$;
    for $j \leftarrow 1$ to $IntraRoute$ **do**
      $currIntra \leftarrow IntraRoute[j]$;
      $S \leftarrow currIntra(S)$;
      if f(S) $\leq$ f(S') **then**
        $S' \leftarrow S$;
        $j \leftarrow 1$; `// Restarting Intra-Route local search`
      end
    end
    $i \leftarrow 1$;
    $InterRoute \leftarrow$Random Order($Inter\text{-}Route\ Neighborhoods$);
  end
  if f(S') $\leq$ f($S*$) **then**
    $S* \leftarrow S'$;
  end
end

Output : $S*$

Neste algoritmo, a solução atual S analisa as vizinhanças externas às rotas (InterRoute) em uma ordem aleatória, comparando seus vizinhos, e caso algum destes possua importância superior à sua, este vizinho torna-se a solução atual S e em seguida verifica as vizinhanças internas às rotas da solução (IntraRoute) sequencialmente. A melhor solução encontrada durante o processo é retornada.

4.6 Iterated Local Search

Ao se gerar variados ótimos locais, utilizar a Iterated Local Search torna-se possível obter aprimoramento da qualidade da solução (Talbi, 2009). Neste artigo em específico, as aplicações de Iterated Local Search utilizam duas versões de busca local RVND.

Os métodos de perturbação do algoritmos descritos em seguida utilizam os operadores de vizinhança Inter-Rota Swap-Job(S_i, S_w) e Insert-Job(S_i). É realizada uma quantidade α de perturbações, sendo α estatisticamente definido na Seção 'Calibração de Parâmetros'.

4.7 ILS RVND

O esquema geral da heurística ILS proposta é apresentado em Algoritmo 3. Existem três parâmetros: α (número de perturbações), $MaxIter$ (quantidade de reinícios de solução) e $MaxIterILS$ (máximo de iterações consecutivas para o algoritmo). Inicialmente, a variável de melhor solução global é inicializada (Passo 1).

A cada iteração externa, uma nova solução inicial S é gerada (Passo 3) utilizando as regras descritas na Seção 3.1, e em seguida é feito uma busca local em S (Passo 4). Após sua possível melhora, o processo de Perturbação é inicializado (Passos 5-12), perturbando a solução atual S' com intensidade α (Passo 10), aplicando busca local (Passo 11) e atualizando a melhor solução encontrada durante este processo (Passos 6-9).

Caso alguma melhora em S' em relação à solução inicial S seja obtida, o Processo de Perturbação é reiniciado (Passo 8). Por último, a solução S' obtida é comparada a melhor solução global $S*$ e então se inicia a próxima iteração do algoritmo. O ILS-RVND retorna a melhor solução encontrada $S*$ no seu fim.

4.8 ILS-RVND com Aprimoramento de Rota Interna

P.H.V. Penna, L. S. Ochi, 2013, desenvolveram o primeiro modelo de Iterated Local Search com RVND para o Problema de Roteamento de Frota Heterogênea de Veículos (HFVRP) em um ambiente com até cem clientes, e produziram resultados competitivos com aos da literatura. Sua abordagem específica em lidar com a entrega de mercadorias serviu de inspiração para esta combinação ILS-RVND.

O diferencial desta versão de Iterated Local Search está na busca local RVND-Custom (Figura 3) descrita anteriormente, que utiliza a técnica de aprimoramento interno de rotas.

Algorithm 3: ILS-RVND $(\alpha, MaxIter, MaxIterILS)$

```
   Input  : Parameters of max. iterations numbers
   Output : Best solution S*
 1 S* ← Initialize(S*);
 2 for i ← 1 to MaxIter do
 3 |   S ← GenerateInitialSolution();
 4 |   S' ← RVND(s);
 5 |   for j ← 1 to MaxIterILS do
 6 |   |   if f(S) ≤ f(S') then
 7 |   |   |   S' ← S;
 8 |   |   |   j ← 1; // Restart Perturbing Process
 9 |   |   end
10 |   |   S' ←Perturb(S', α);
11 |   |   S' ←RVND(S'); // Local Search on Perturbed S'
12 |   end
13 |   if f(S') ≤ f(S*) then
14 |   |   S* ← S';
15 |   end
16 end
   Output : S*
```

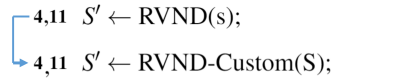

Fig. 3. Mudança entre acionamentos de RVND.

Nesta técnica, assim que uma solução atual é aperfeiçoada utilizando operadores externos às rotas, como realizar intercâmbio de tarefas entre veículos, os operadores internos às rotas são chamados.

4.9 Genetic Algorithm

Algoritmos Genéticos foram desenvolvidos por J. Holland em meados dos anos 70 (University of Michigan, USA) para entender o processo adaptativo de sistemas naturais (Talbi, 2009). Nesta abordagem de Algoritmo Genético são utilizadas técnicas de Torneio Binário para seleção de soluções e 2-point Crossovers para diversificação da prole. O esquema geral do algoritmo GA é descrito em Algorithm 4.

Existem dois parâmetros: PopSize (quantidade de de indivíduos na população) e MaxIter (número de iterações em que serão geradas novas populações. O algoritmo é iniciado no Passo 1 ao gerar uma população aleatória P, seguido de uma busca local RVND (Passo 2). A melhor solução da população P será armazenada em $P*$ (Passo 3) para ser utilizada posteriormente.

A cada iteração uma nova solução é inserida em P até que o limite de $PopSize - 1$ indivíduos (Passos 6-22) seja atingido. Para gerar cada prole, são

escolhidas duas soluções 'pai' por Torneios Binários (Passos 7-8), e em seguida são geradas duas mutações destas soluções (Passos 9-10).

Algorithm 4: Genetic Algorithm ($PopSize, MaxIter$)

```
   Input  : Population size and max. iteration number
   Output : Best solution P* in MaxIter populations
 1 P ← Generate random population(PopSize);
 2 P ←  RVND(P); // Apply local search to new population P
 3 P* ← Best(P); // Save best solution of P
 4 for i ← 1 to MaxIter do
 5 |   P' ← ∅;
 6 |   for j ← 1 to (PopSize − 1) do
 7 |   |   Parent1 ← Binary Tournament(random(P), random(P));
 8 |   |   Parent2 ← Binary Tournament(random(P), random(P));
 9 |   |   Offspring1 ← Crossover(first, second);
10 |   |   Offspring2 ← Crossover(first, second);
11 |   |   if  Feasible(Offspring1) and Feasible(Offspring2)  then
12 |   |   |   P' ← P' + BinaryTournament(Offspring1, Offspring2);
13 |   |   else
14 |   |   |   if  Feasible(Offspring1) or Feasible(Offspring2) then
15 |   |   |   |   P' ← P' + Feasible from(Offspring1, Offspring2);
16 |   |   |   else
17 |   |   |   |   if !Feasible(Offspring1) and !Feasible(Offspring2)  then
18 |   |   |   |   |   P' ← P' + Mutation(random(Parent1, Parent2));
19 |   |   |   |   end
20 |   |   |   end
21 |   |   end
22 |   end
23 |   P' ← P' + P*; // Include best solution to P'
24 |   P' ←  Fast Local Search(P');
25 |   if  f(P') ≤ f(Best(P')) then
26 |   |   P' ← P*;
27 |   end
28 |   P ←  P';
29 end
   Output : P*
```

O processo de mutação Crossover pode produzir soluções inviáveis, dessa forma a escolha de uma nova solução para a população P' precisa ser refinada (e então viável). No primeiro caso, em que as duas novas mutações são viáveis, estas participam de um Torneio Binário (Passo 12) e a escolhida será adicionada à população. No segundo caso, onde apenas uma das soluções é viável, a única prole viável será inserida na população P' (Passo 15). No último caso, se as duas mutações geradas são inviáveis (Passo 17), uma escolha aleatória entre as soluções 'pai' é feita e sofrerá uma nova mutação Crossover (Passo 18), sendo

adicionada em seguida. Após a obtenção de $PopSize - 1$ indivíduos em P', é inserida também a melhor solução $P*$ encontrada até o momento (Passo 23). Com a população completa, é realizada uma busca local rápida (Passo 24) em todos os indivíduos. Este método é descrito na Seção 3.4.1. Por fim, a melhor solução $S*$ é atualizada (Passos 25-26) e a população P' torna-se a população atual P, para a futura criação de novos indivíduos (Passo 25).

5 Computational Experiments

All the heuristic algorithms were coded in C++ and executed on a PC machine IntelR Core TM i7-4790K CPU @ 4.00GHz x 8 with 32GB RAM, running Ubuntu 14.04 LTS 64 bits. The quality of the solutions were measured by the relative percentage deviation (RPD) from the best known solution (reference solution). The RPD measure represents the percentage of how much experimental results differs from a reference value, and is calculated following the equation:

$$RPD(\%) = 100 \times \frac{TT_{\text{experimental}} - TT_{\text{reference}}}{TT_{\text{reference}}} \tag{7}$$

where $TT_{\text{reference}}$ is the value of the reference solution, and $TT_{\text{experimental}}$ is the obtained solution by a given heuristic.

6 Conclusions

Acknowledgments. The authors thanks the financial support of FAPEMIG, CAPES and CNPq, Brazilian research agencies.