4
Estudo de caso: Problema de seqüenciamento de carros

O problema de seqüenciamento de carros em linhas de produção das indústrias automobilísticas é um tipo particular de problema de escalonamento que consiste em determinar a ordem dos carros a serem produzidos na linha de montagem de cada dia de produção que melhor satisfaz os requisitos das oficinas de pintura e de montagem. Estes requisitos procuram reduzir os custos da linha de produção e atender às demandas dos clientes nos prazos estabelecidos. Uma revisão bibliográfica sobre o problema encontra-se em [29]. Neste trabalho considera-se o problema tema do concurso ROADEF'2005 [24] proposto pela Renault.

4.1
Descrição do problema

O problema de seqüenciamento de carros (PSC) é um subproblema de planejamento e escalonamento da produção de veículos. O processo de planejamento da produção se inicia com os pedidos dos clientes, que são enviados em tempo real para as fábricas. Diariamente as fábricas determinam um dia para produção de cada veículo solicitado, respeitando as capacidades da linha de produção e a data de entrega que foi prometida aos consumidores. A linha de produção pode ser considerada como um processo de manufatura linear composta por três oficinas como ilustrado na Figura 4.1. Em seguida deve-se escalonar a ordem para construção dos carros na linha de montagem, satisfazendo o máximo de requisitos das oficinas de carroceria, pintura e montagem. A seqüência de carros na linha de montagem é a mesma para todas as oficinas.

Figura 4.1: Oficinas da linha de produção de uma fábrica de carros

Para o escalonamento diário, a oficina de carroceria não possui requisitos e o conjunto de veículos pré-determinado na fase anterior não pode ser alterado. Os requisitos das oficinas de pintura e de montagem são apresentados nas próximas seções.

Nas fábricas da Renault, uma aplicação industrial gerencia o processo de planejamento e escalonamento, usando programação linear para planejamento e *simulated annealing* para o escalonamento.

4.1.1 Requisitos de pintura

A oficina de pintura busca minimizar o consumo de solvente de tinta. O solvente é utilizado para limpar pistolas de tinta toda vez que a cor de pintura é alterada entre dois veículos consecutivos na seqüência de produção. Portanto, para minimizar o número de trocas da cor de pintura procura-se agrupar em lotes veículos com mesma cor de pintura.

Seqüências de veículos de mesma cor de pintura têm uma limitação no tamanho máximo, já que pistolas de tinta precisam ser lavadas regularmente mesmo se não houver troca na cor da tinta. Esta limitação é uma restrição forte e deve ser respeitada para se obter uma solução viável do problema.

4.1.2 Requisitos de montagem

Com o intuito de suavizar a carga de trabalho na linha de montagem, veículos que requerem operações especiais de montagem têm que ser igualmente distribuídos através do total de carros processados. Esses veículos são considerados como sendo "difíceis de montar", exigindo, por exemplo, teto solar ou ar condicionado, e não podem exceder uma dada quota sobre qualquer seqüência de veículos. Estes requisitos são modelados através de uma restrição de capacidade definida por uma razão N/P.

Uma restrição de capacidade N/P significa que no máximo N carros em cada seqüência consecutiva de P devem estar associados a essa restrição. Por exemplo, se $N/P = 3/5$, então não deve haver mais do que três carros restritos em qualquer seqüência consecutiva de cinco veículos. De um modo mais geral, um carro que possui uma restrição $1/P$ significa que na solução deve haver pelo menos uma seqüência de $P - 1$ carros adjacentes que não possuam essa restrição para que não haja violações dessa restrição.

Há dois tipos de restrições de capacidade: restrições de alta prioridade e restrições de baixa prioridade. As restrições de alta prioridade são relativas às características de carros que demandam um grande adicional na carga de

trabalho na linha de montagem. As restrições de baixa prioridade resultam de características dos carros que causam pequena inconveniência para a produção. Restrições de alta prioridade devem ser satisfeitas preferencialmente, em relação às restrições de baixa prioridade.

Restrições de capacidade são consideradas restrições leves,ou seja, atender a todas as restrições de capacidade não pode ser garantido de antemão quando um dia de produção é planejado. Por isso, um dos objetivos da otimização é minimizar o número de violações das restrições de capacidade.

4.1.3 Regra do dia de produção $D-1$

Quando o dia de produção D é escalonado, os últimos veículos escalonados do dia de produção $D-1$ devem ser considerados. Os veículos do dia de produção $D-1$ já estão escalonados e suas posições não podem ser alteradas. A computação do número de violações das restrições de razão no dia de produção D deve levar em conta os últimos veículos do dia de produção $D-1$. O dia de produção $D+1$ é ignorado enquanto o dia de produção D é escalonado.

4.1.4 Definição do Problema

Deve-se escalonar uma seqüência de veículos que satisfaça melhor aos requisitos da oficina de pintura e da linha de montagem. Visto que é um problema multi-objetivo, há diversas possibilidades para a ordem de prioridade entre os requisitos. Na prática existe um algoritmo para tratar cada problema. Neste trabalho considera-se a seguinte ordem de objetivos:

1. minimizar violações das restrições de capacidade de alta prioridade;
2. minimizar violações das restrições de capacidade de baixa prioridade e
3. minimizar número de trocas de cores.

A qualidade das soluções é avaliada através de uma função de custo que atribui pesos aos objetivos de acordo com os níveis de prioridade. A idéia é que a diferença entre esses pesos seja suficientemente grande para reduzir a possibilidade de compensação de um objetivo em detrimento de outro. Dessa forma, pretende-se que a melhor solução seja aquela com o menor valor do primeiro objetivo, em seguida, com o menor valor do segundo e, por fim, com o menor valor do terceiro objetivo.

4.1.5 Formalização

Segundo Rocha [29], o problema de seqüenciamento de carros pode ser definido por uma tupla (C, O, p, q, r), onde $C = \{c_1, \ldots, c_n\}$ é o conjunto de carros cuja produção deve ser escalonada e $O = \{o_1, \ldots, o_m\}$ é o conjunto das diferentes opções que os carros podem requerer. As restrições de capacidades associadas com cada opção $o_j \in O$ são definidas pelas funções $p : O \to \mathbb{N}$ e $q : O \to \mathbb{N}$, que determinam que, para cada seqüência de carros consecutivos de tamanho $q(o_j)$, no máximo $p(o_j)$ destes carros podem requerer a opção o_j. As demandas de opções são definidas pela função $r : C \times O \to \{0, 1\}$, ou seja, $r(c_j, o_j)$ retorna 1 se a opção o_j deve ser instalada no carro c_i, e retorna 0 caso contrário. A notação a seguir será utilizada para manipular as seqüências de carros:

- uma **seqüência** de k carros é denotada por $\pi =< c_{i_1}, c_{i_2}, \ldots, c_{i_k} >$;
- o número de carros que uma seqüência π contém é chamado **tamanho** de π e é denotado por $|\pi|$;
- a **concatenação** de duas seqüências π_1 e π_2 é a seqüência formada pelos carros de π_1 seguidos pelos carros de π_2 e é denotada por $\pi_1|\pi_2$;
- o **número de carros que requerem uma opção** o_j em uma seqüência $\pi =< c_{i_1}, c_{i_2}, \ldots, c_{i_k} >$ é dado por

$$r(\pi, o_j) = \sum_{l=1}^{k} r(c_{il}, o_j);$$

- o **custo** de uma seqüência $\pi =< c_{i_1}, c_{i_2}, \ldots, c_{i_k} >$ é o número de restrições de capacidade violadas, dado por

$$custo(\pi) = \sum_{j=1}^{m} \sum_{\substack{\pi' : |\pi'| = q(o_j) \\ \pi' \subseteq \pi}} violacao(\pi', o_j),$$

onde

$$violacao(\pi', o_j) = \begin{cases} 0 \text{ , se } r(\pi', o_j) \leq p(o_j); \\ 1 \text{ , caso contrário.} \end{cases}$$

Para efeito de simplificação, o número de carros que requerem uma opção o_j no conjunto formado por todos os carros C também é dado por

$$r(C, o_j) = \sum_{i=1}^{n} r(c_i, o_j).$$

Além da notação acima, existe um conceito importante utilizado pela heurística construtiva, chamado de **taxa de utilização** das opções [31]. Este conceito determina, juntamente com o número de carros a serem produzidos e o número de configurações diferentes de opções, a dificuldade de uma instância do problema. A taxa de utilização de uma opção o_j em um conjunto ou uma seqüência de carros é a razão entre o número de carros que demandam o_j na seqüência e o número máximo de carros que podem demandar o_j nesta seqüência de modo a satisfazer a restrição de capacidade associada. Formalmente, a taxa de utilização de uma opção o_j no conjunto de carros C é definida por

$$taxaUtil(o_j) = \frac{r(C, o_j) \cdot q(o_j)}{n \cdot p(o_j)}.$$

Escrevendo de outra forma, a taxa de utilização é definida por

$$taxaUtil(o_j) = \left(\frac{r(C, o_j)}{n}\right) / \left(\frac{p(o_j)}{q(o_j)}\right).$$

Ou seja, a taxa de utilização de uma opção é dada pela razão entre o número de carros demandando essa opção e o número de carros da seqüência dividida pela razão da restrição correspondente. Se a taxa de utilização é maior que 1, a razão do numerador é maior que a razão do denominador e, conseqüentemente, a demanda é maior que a capacidade. Logo, a restrição será violada. Se, ao contrário, a taxa de utilização é muito pequena e próxima de 0, a demanda é muito baixa em relação à capacidade da linha de produção.

4.2 Estratégia algorítmica

Neste trabalho utiliza-se a estratégia para solução proposta por Ribeiro et al. [26, 27, 29]. O projeto da estratégia baseia-se nas características e peculiaridades do problema. O PSC é um problema que visa a otimização de três objetivos tratados com níveis de prioridade diferentes, através de pesos a eles associados na função objetivo. Denomina-se de subproblema o tratamento da otimização separada de cada objetivo. Como a diferença entre os pesos deve ser relativamente grande para enfatizar seus diferentes níveis de prioridade, a idéia de resolver os subproblemas do PSC separadamente por ordem decrescente de prioridade torna-se uma estratégia viável e adequada.

No entanto, quando resolvem-se os subproblemas separadamente é muito comum existirem muitas soluções de mesmo custo, o que se denomina de um empate. Para o subproblema de minimizar as trocas de cores, que é um problema mais simples, essa característica é facilmente notada, por exemplo,

através da permutação de grupos de carros de mesma cor. Para o subproblema de minimizar violações de restrições de capacidade de alta ou baixa prioridade, os empates podem ser observados no exemplo da Figura 4.2, onde se tem várias seqüências com o mesmo número de violações para uma restrição de capacidade de razão 1/4. Mesmo quando o número de restrições aumenta, os empates ainda acontecem.

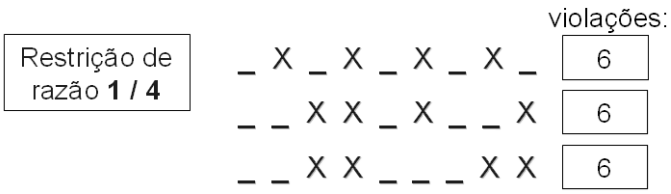

Figura 4.2: Soluções com mesmo número de violações

A estratégia utilizada para resolver o PSC foi, então, desenvolver algoritmos específicos para cada um de seus subproblemas, respeitando o nível de prioridade dos objetivos. Os algoritmos desenvolvidos são técnicas heurísticas e são aplicados de forma seqüencial, conforme citado abaixo:

`Passo 1`: *Heurística construtiva*;
`Passo 2`: *Algoritmo para otimizar o primeiro objetivo*;
`Passo 3`: *Algoritmo para otimizar o segundo objetivo*;
`Passo 4`: *Algoritmo para otimizar o terceiro objetivo.*

4.2.1 Heurística construtiva

Uma solução inicial é construída escolhendo-se iterativamente o próximo carro a ser escalonado na seqüência de acordo com um critério guloso. O critério guloso adotado consiste em escolher o carro que causa o menor número de novas violações de restrições em cada iteração. De uma maneira mais formal, dada uma seqüência parcial π, o próximo carro c_i a ser escolhido é aquele que minimiza

$$novasViolacoes(\pi, c_i) = \sum_{j=1}^{m} \left(r(c_i, o_j) \times violacao(ultCars(\pi| < c_i >, q(o_j)), o_j) \right),$$

onde

$$ultCars(\pi', k) = \begin{cases} \text{seqüência com os últimos } k \text{ carros de } \pi', \text{ se } |\pi'| \geq k \\ \pi', \text{caso contrário.} \end{cases}$$

No entanto, freqüentemente existem vários carros candidatos que minimizam o número de novas violações. Diante disto, um segundo critério guloso é adotado para resolver os empates. Este critério é um pouco mais elaborado, pois favorece uma distribuição eqüitativa das opções. A distribuição é baseada na seguinte idéia: se o número médio de carros que demandam uma opção é menor na seqüência em construção π do que no conjunto total de carros C, então estes carros são favorecidos para entrar na seqüência e vice-versa. Mais formalmente, escolhe-se o carro c_i que minimiza a função:

$$\eta_{SD}(c_i, \pi) = \sum_{j=1}^{m} \left\{ (r(c_i, o_j) = 0) \text{ XOR } \left(\frac{r(C, o_j)}{n} > \frac{r(\pi, o_j)}{|\pi|} \right) \right\}.$$

O primeiro carro a ser escalonado é escolhido aleatoriamente entre os carros com o número máximo de opções, uma vez que a fórmula acima não pode ser aplicada quando $|\pi| = 0$.

Ainda que o critério acima seja utilizado para quebrar os empates, não é raro que eles persistam. Neste caso, um terceiro critério guloso é utilizado, favorecendo carros que requerem opções com alta taxa de utilização. Isso é feito através da soma das taxas de utilização das opções requeridas por cada carro. A taxa de utilização ora considerada é dinâmica, pois é atualizada sempre que um carro é adicionado à seqüência parcial π e é definida pela seguinte função:

$$taxaUtilDin(o_j, \pi) = \frac{[r(C, o_j) - r(\pi, o_j)] \cdot q(o_j)}{[n - |\pi|] \cdot p(o_j)}$$

O último critério de desempate é chamado de soma de taxas de utilização dinâmicas e é dado por:

$$\eta_{SD}(c_i, \pi) = \sum_{j=1}^{m} r(c_i, o_j) \cdot taxaUtilDin(o_j, \pi).$$

Escolhe-se, então, o carro ainda não escalonado que maximiza η_{SD}. Se os empates persistirem após esta segunda tentativa de desempate, um carro é escolhido aleatoriamente dentre os carros candidatos.

4.2.2 Vizinhanças e busca local

A otimização dos diversos objetivos é realizada por heurísticas que utilizam estruturas de vizinhança em comum. Os movimentos aplicados pelas heurísticas de busca local são baseados nas estruturas de vizinhança de troca e deslocamento.

Na vizinhança de troca, um movimento corresponde a inverter as posições de dois carros e é definido por um par de posições (i, j). Em uma seqüência de carros $\pi =< c_{k_1}, \ldots, c_{k_n} >$, a aplicação do movimento $swap(\pi, i, j)$ resulta na seqüência $\pi' =< c_{k_1}, \ldots, c_{k_{i-1}}, c_{k_j}, c_{k_{i+1}}, \ldots, c_{k_{j-1}}, c_{k_i}, c_{k_{j+1}}, \ldots, c_{k_n} >$.

Na vizinhança de deslocamento, um movimento também é representado por um par de posições (i, j) e corresponde a remover um carro de sua posição atual i e inseri-lo em uma nova posição, j. A aplicação do movimento $shift(\pi, i, j)$ na seqüência de carros $\pi =< c_{k_1}, \ldots, c_{k_n} >$ resulta na seqüência $\pi' =< c_{k_1}, \ldots, c_{k_{i-1}}, c_{k_{i+1}}, \ldots, c_{k_j}, c_{k_i}, c_{k_{j+1}}, \ldots, c_{k_n} >$, se $i < j$.

É utilizado um procedimento de busca local diferenciado que não é do tipo primeiro aprimorante, nem maior aprimorante [27]. A vizinhança é dividida em diversas sub-vizinhanças, onde cada uma contém todos vizinhos gerados pela troca ou deslocamento de um carro. Portanto, há n sub-vizinhanças, uma para cada carro. As sub-vizinhanças são utilizadas progressivamente, do primeiro ao último carro, a cada iteração da busca. A partir de uma solução, a busca avança para a melhor solução que não é pior do que a solução corrente na sub-vizinhança correspondente, mesmo se tiver custo igual.


```
procedure LocalSearch(π);
1  repeat
2      φ ← cost(π)
3      for i = 1, ..., |π| do
4          Δ* ← 0
5          L ← ∅
6          for j = 1, ..., |π| do
7              Δ ← cost(move(π, i, j)) − cost(π)
8              if Δ < Δ* then
9                  L ← {j}
10                 Δ* ← Δ
11             else if Δ = Δ* then
12                 L ← L ∪ {j}
13             end-if
14         end-for
15         if L ≠ ∅ then
16             Escolha k ∈ L aleatoriamente
17             π ← move(π, i, j)
18         end-if
19     end-for
20 until φ = cost(π)
21 return π
end LocalSearch;
```

Figura 4.3: Pseudo-código da heurística de busca local.

A Figura 4.3 mostra o pseudo-código do procedimento de busca local. O custo da solução π é calculado por $cost(\pi)$ e $move(\pi, i, j)$ significa o movimento de inverter os carros nas posições i e j ou o movimento de deslocar o carro da posição i para a posição j, dependendo da vizinhança que estiver sendo usada. O procedimento de busca é definido pelo laço nas linhas 1–20, cuja execução é realizada até que nenhuma solução aprimorante possa ser encontrada. O custo da solução corrente ϕ é inicializado na linha 2. O laço nas linhas 3–19 trata os movimento relativos a cada carro $i = 1, \ldots, n$. A variável Δ^* contém o valor da redução no custo da solução corrente referente ao melhor movimento aceitável e é inicializada com zero na linha 4, visto que somente movimentos que não piorem a solução corrente podem ser aceitos. O conjunto L dos melhores movimentos envolvendo o carro i é inicializado na linha. Os movimentos associados ao carro i são investigados no laço 6–14. A redução Δ no custo da solução corrente, relativa ao movimento $move(\pi, i, j)$, é computada na linha 7. Se Δ é menor do que a menor redução de custo do melhor movimento identificado até então, o último é atualizado nas linhas 9 e 10. Se tem custo igual ao mehor movimento identificado até então, o novo movimento é adicionado na lista de melhores movimento na linha 12. No caso de serem encontrados vários movimentos de mesmo custo envolvendo o carro i que não piorem a solução corrente, o movimento a ser aplicado na solução corrente é selecionado aleatoriamente a partir do conjunto de melhores movimentos na linha 16. O movimento escolhido é aplicado à solução corrente na linha 17. A solução obtida pela busca local é retornada na linha 19.

4.2.3 Otimização do primeiro objetivo

O primeiro objetivo está relacionado com as restrições de capacidade de alta prioridade. O objetivo é minimizar o número de violações dessas restrições e, de preferência, eliminá-las.

A heurística construtiva obtém uma solução inicial considerando somente as restrições de capacidade. Por isso, pode-se iniciar esta fase a partir de uma solução inviável através da violação no tamanho limite para as seqüências de carros de mesma cor. O algoritmo para otimização do primeiro objetivo também permite explorar regiões com soluções inviáveis, deixando a restauração da viabilidade para um momento mais tarde.

O algoritmo desenvolvido é baseado na metaheurística busca local iterativa (ILS) [22]. A estratégica básica dessa metaheurística consiste em sucessivamente aplicar perturbações e buscas locais na solução corrente.

A função de perturbação utilizada no algoritmo proposto consiste em

remover um dado número de carros da solução corrente e, em seguida, inseri-los na solução de acordo com o critério guloso descrito anteriormente. Os carros removidos sempre são carros envolvidos em violações das restrições de capacidade de alta prioridade.

A busca local aplicada é baseada na vizinhança definida pelo movimento de troca, com um melhoramento adicional. Uma busca local rápida foi desenvolvida considerando uma propriedade básica. Só consegue-se melhorar uma solução para o primeiro objetivo quando pelo menos um dos carros trocados viola alguma restrição de razão de alta prioridade. Empiricamente é verificado que a fração de carros envolvidos nas violações é menor do que 40%, este melhoramento agiliza a exploração de soluções aprimorantes nesta vizinhança. Portanto, são apenas realizadas trocas de pares de carros que envolvam pelo menos um carro que viole alguma restrição relativa ao primeiro objetivo.

Uma fase de intensificação foi adicionada à heurística desenvolvida. Essa fase consiste da sucessiva aplicação de dois procedimentos de busca local sempre que, depois de um dado número de iterações, nenhum melhoramento for obtido na solução corrente. Ela começa por uma busca local na vizinhança de deslocamento de carros, seguido de uma busca local na vizinhança de troca. Essa fase também pode ser vista como um procedimento de diversificação, pois além de otimizar na vizinhança da solução corrente, é permitido aceitar movimentos com o mesmo custo, escolhidos aleatoriamente. Desse modo, é possível visitar muitas soluções distintas com o mesmo custo durante uma mesma aplicação da busca local, estendendo o conjunto de soluções visitadas.

São realizados reinícios na busca sempre que a solução corrente não é melhorada após um dado número de iterações. As soluções iniciais da busca são obtidas por uma perturbação mais forte da solução corrente, através da remoção de um maior número de carros ou pela aplicação da heurística construtiva. A primeira opção é preferida, visto que nenhuma informação será guardada das iterações anteriores quando se reinicia com uma solução obtida pela heurística construtiva.

O número de reinícios é um dos critérios de parada utilizados pelo algoritmo. O segundo critério de parada é baseado no tempo de computação, visto que há um limite total de tempo para a otimização de todos objetivos. O terceiro e último critério de parada é quando não há mais violação das restrições de capacidade de alta prioridade na solução corrente.

Na Figura 4.4 está o pseudocódigo do algoritmo `ILS_HPRC` para otimização do primeiro objetivo. Na linha 1, a melhor solução π^* e a solução corrente π iniciais são definidas como a solução obtida pela heurística construtiva. O laço nas linhas 2–25 é realizado até que um dos critérios de parada

```
procedure ILS_HPRC(π0);
1   π ← π0, π* ← π0
2   while critérios de parada não satisfeitos do
3       π' ← Perturbation(π)
4       π' ← LocalSearch(π)
5       if custo(π) ≥ custo(π') then
6           π ← π'
7       end-if
8       if custo(π) ≥ custo(π*) then
9           π* ← π
10      end-if
11      if o número de iterações desde o último
            melhoramento em π for igual a α then
12          π' ← Intensification(π)
13          if custo(π) ≥ custo(π') then
14              π ← π'
15          end-if
16          if custo(π) ≥ custo(π*) then
17              π* ← π
18          end-if
19      end-if
20      if o número de iterações desde o último
            melhoramento em π for igual a β then
21          if cost(π) = cost(π*) then
22              π ← Restart(π)
23          else π ← π*
24      end-if
25  end-while
26  return π*
end ILS_HPRC;
```

Figura 4.4: Pseudocódigo do algoritmo de minimização do primeiro objetivo.

seja satisfeito. Cada iteração do algoritmo ILS inicia na linha 3 com uma perturbação aplicada à solução corrente π, levando a uma solução intermediária π'. Uma busca local na vizinhança de troca de carros é aplicada a π' na linha 4. Se a nova solução π' for melhor ou tão boa quanto a π, ela substitui a solução corrente na linha 6. A melhor solução conhecida π^* é atualizada na linha 9 se π' for melhor ou tão boa quanto ela. Se nenhum melhoramento foi obtido na solução atual π depois de α iterações, a fase de intensificação é realizada na linha 12 e a solução corrente tem a possibilidade de ser atualizada na linha 14. A melhor solução conhecida π^* pode ser atualizada na linha 17 se a solução obtida na intensificação for melhor. Se nenhum melhoramento na solução corrente π for obtido depois de β iterações (com $\alpha < \beta$), um reinício é

realizado se o custo dessa solução for igual ao custo da melhor solução conhecida π^*. Caso contrário, faz-se a solução corrente π ser igual a melhor solução conhecida (linha 23).

4.2.4 Otimização do segundo objetivo

O algoritmo proposto para a otimização do segundo objetivo inicia com uma solução obtida pelo algoritmo para a otimização do primeiro objetivo. O primeiro (número de violações das restrições de razão de alta prioridade) e o segundo (número de violações das restrições de razão de baixa prioridade) objetivos são levados em consideração nesta fase. O segundo objetivo é otimizado sem piora do primeiro, que também pode ser melhorado durante este processo.

O algoritmo desenvolvido é baseado na metaheurística busca em vizinhança variável (VNS) [17]. A metaheurística VNS propõe uma troca sistemática de vizinhanças $N_1, \ldots, N_{k_{max}}$ onde são aplicadas perturbações dentro dessas vizinhanças. Basicamente, cada iteração consiste em uma fase de perturbação, seguida de um procedimento de busca local. A busca inicia pela exploração de vizinhanças menores. A região de busca é aumentada sempre que uma solução de melhora não é encontrada dentro da vizinhança atual.

A fase de perturbação faz uso de duas estruturas de vizinhança de um modo oscilatório, trocando o tipo de perturbação sempre que a vizinhança de mais alta ordem da perturbação corrente é atingida. Na primeira estrutura de vizinhança são realizados movimentos de troca aplicados em pares de carros do mesmo tipo escolhidos aleatoriamente, preservando-se assim o valor do custo do primeiro objetivo. Aqui, carros do mesmo tipo são aqueles associados ao mesmo conjunto de restrições de razão de alta prioridade. A segunda estrutura de vizinhança está relacionada com movimentos de deslocamento, que consiste em remover alguns carros envolvidos nas violações da solução corrente e inseri-los em novas posições.

Denota-se por $N_k^1, k = k_1, \ldots, k_{max}$, a seqüência de vizinhanças de deslocamento, onde k é o número de carros que são removidos e subseqüentemente inseridos em novas posições. Similarmente, denota-se por $N_q^2, q = q_1, \ldots, q_{max}$, a seqüência de vizinhança de troca, onde q é o número de trocas de carros.

A estratégia para uma busca local rápida apresentada anteriormente é utilizada. Soluções que sofrem perturbações por movimentos de deslocamento podem ter seu primeiro objetivo deteriorado. Por isso, o procedimento de busca local tenta otimizar a soma ponderada do primeiro com o segundo objetivos. No caso de soluções que sofrem perturbações por movimentos de troca, a busca local considera somente pares de carros do mesmo tipo, conseqüentemente

preservando o custo associado ao primeiro objetivo.

O procedimento de intensificação é similar ao descrito para a otimização do primeiro objetivo. Ele é aplicado sempre que o tipo de perturbação é alternado. O primeiro e o segundo objetivos podem ser melhorados, uma vez que ambos são considerados nesse procedimento.

A execução é finalizada quando qualquer um dos três critérios de parada for satisfeito. O primeiro é um número máximo de fases de intensificação desde o último melhoramento da solução corrente. O segundo critério é definido por um limite de tempo gasto na otimização do segundo objetivo, para garantir que o terceiro objetivo também seja otimizado. Por último, o terceiro critério de parada é satisfeito se não houver violações de restrições de razão de baixa prioridade.

O pseudocódigo apresentado na Figura 4.5 provê uma visão geral do algoritmo `VNS_LPRC` para a otimização do segundo objetivo. Algumas variáveis e parâmetros são descritos a seguir. A variável π^* representa a solução corrente, que é também a melhor solução conhecida, e π' a solução obtida por alguma perturbação. Representa-se por ϕ^* o custo da solução corrente e por ϕ o custo da solução obtida na última fase de intensificação. O contador i acumula o número de fases de intensificação realizadas desde o último melhoramento na solução corrente. O limite superior para este contador é o parâmetro i_{max}. Contadores k e q representam a ordem da vizinhança corrente usada na geração das perturbações de deslocamento e troca, respectivamente. Os parâmetros k_1 e q_1 definem as mais baixas ordens de vizinhança para as perturbações de deslocamento e troca, respectivamente. Analogamente, k_{max} e q_{max} definem as mais altas ordens de vizinhança para as perturbações de deslocamento e troca, respectivamente. Finalmente, P é uma variável binária cujo valor é 1 se a perturbação corrente consiste de movimentos deslocamento e 0 caso contrário.

O algoritmo possui como solução inicial a solução π_0 obtida pela heurística `ILS_HPRC`, usada para a otimização do primeiro objetivo. Configurações iniciais são realizadas nas linhas 1–2. O laço nas linhas 3–27 é realizado até um dos critérios de parada ser satisfeito. As ordens de vizinhança e o custo da solução corrente tem seus valores iniciais definidos nas linhas 4 e 5, respectivamente. Uma iteração do laço nas linhas 6–18 é composto pela aplicação da perturbação na solução corrente π^* resultando na solução π' na linha 7, seguida por uma busca local aplicada nessa última solução obtida na linha 8. A nova solução π' é aceita na linha 9 para substituir a solução corrente π^* se ela for pelo menos tão boa quanto a melhor solução conhecida. O condicional na linha 12–17 atualiza a ordem de vizinhança que será usada na próxima iteração. Se qualquer melhoramento for realizado na solução corrente, a busca

```
procedure VNS_LPRC(π_0);
1   π* ← π_0, φ* ← cost(π_0)
2   i ← 0, P ← 1
3   while critérios de parada não satisfeitos do
4       k ← k_1, q ← q_1
5       φ ← φ*
6       while (P = 1 and k ≤ k_max) or (P = 0 and q ≤ q_max) do
7           π' ← Perturbation(π*, P, N_k^1, N_q^2)
8           π' ← LocalSearch(π')
9           if cost(π') ≤ cost(π*) then
10              π* ← π'
11          end-if
12          if cost(π*) < φ* then
13              k ← k_1, q ← q_1
14              φ* ← cost(π*)
15          else
16              k ← k + 1, q ← q + 1
17          end-if
18      end-while
19      P ← 1 − P
20      π* ← Intensification(π*)
21      φ* ← cost(π*)
22      if φ* < φ then
23          i ← 0
24      else
25          i ← i + 1
26      end-if
27  end-while
28  return π*
end VNS_LPRC;
```

Figura 4.5: Pseudocódigo do algoritmo de minimização do segundo objetivo.

reiniciará a partir da mais baixa ordem de vizinhança (linha 13) e o custo ϕ^* da melhor solução conhecida é atualizado (linha 14). Caso contrário, a busca reiniciará a partir de uma ordem de vizinhança maior (linha 16). Quando a perturbação corrente alcançar sua mais alta ordem (k_{max} ou q_{max}, dependendo do seu tipo), o laço nas linhas 6–18 termina e o tipo da perturbação é trocado na linha 19. O procedimento de intensificação é aplicado à solução corrente na linha 20 e o custo da melhor solução conhecida é atualizado na linha 21. A variável i, usada para o critério de parada, é atualizado na linha 23 ou na linha 25, dependendo do resultado do procedimento de intensificação. O algoritmo pára sempre que algum dos critérios de parada for satisfeito.

4.2.5 Otimização do terceiro objetivo

O algoritmo para a otimização do terceiro objetivo também é baseado na metaheurística VNS [17]. Todos os objetivos são considerados simultaneamente, de modo que o primeiro objetivo seja preservado e o segundo e terceiro objetivos são melhorados. A solução inicial obtida pelo algoritmo `VNS_LPRC`, usado para otimizar o segundo objetivo, pode ser inviável. Por exemplo, pode haver lotes de carros de mesma cor cujo tamanho exceda o tamanho máximo permitido. Por isso, a primeira fase do algoritmo consiste em tornar viável a solução inicial, sempre que necessário.

No processo de tornar viável a solução inicial alguns cuidados devem ser tomados para não piorar os objetivos previamente otimizados. Nesse sentido, dois procedimentos separados foram desenvolvidos. O primeiro utiliza movimentos de troca para eliminar as violações. Os movimentos são aplicados somente em pares de carros que compartilhem as mesmas restrições de capacidade, preservando o custo do primeiro e segundo objetivos. Se o primeiro procedimento não for capaz de conseguir a viabilidade, o segundo é aplicado. Somente o último garante a viabilidade da solução, embora com um possível deterioramento dos objetivos previamente otimizados. O segundo procedimento, através de movimentos de deslocamento, remove carros de lotes que violem o tamanho máximo para carros de mesma cor, inserindo-os em novas posições viáveis. As novas posições são aquelas que levam às menores variações do custo da solução.

No algoritmo de otimização do terceiro objetivo, as perturbações são produzidas por movimentos de deslocamento e troca, do mesmo modo descrito para o algoritmo de otimização do segundo objetivo. Como anteriormente, o tipo de perturbação é alternado sempre que a perturbação de mais alta ordem for alcançada. Soluções produzidas por movimento de deslocamento que deteriorem o primeiro objetivo são descartadas e a busca local é aplicada na solução corrente. Isso não se aplica no caso de soluções produzidas por movimentos de troca, pois o custo associado ao primeiro objetivo é preservado através da garantia de que as trocas envolvam apenas carros do mesmo tipo (em relação às restrições de razão de alta prioridade).

Os conjuntos de vizinhança $N_s^1, s = s_1, \ldots, s_{max}$ e $N_t^2, t = t_1, \ldots, t_{max}$ correspondem, respectivamente, aos movimentos de deslocamento e troca. Os valores de s_{max} e t_{max} devem ser menores do que aqueles usados no algoritmo para otimização do segundo objetivo, senão muitas soluções perturbadas serão descartadas.

Soluções obtidas por perturbações têm o valor do primeiro objetivo

preservado, mas podem ter o valor do segundo objetivo deteriorado. Assim sendo, a busca local otimiza a soma ponderada do segundo e terceiro objetivos. A busca local completa é adotada, mas somente são considerados movimentos de trocas de carros envolvendo carros de mesmo tipo (com relação às restrições de razão de alta prioridade). A solução obtida pela busca local é descartada se ela deteriorar o segundo objetivo.

A fase de intensificação é similar àquela previamente apresentada. A única diferença é que todos os objetivos são simultaneamente considerados. Uma fase de intensificação é realizada sempre que o tipo de perturbação é alterado.

A Figura 4.6 mostra o pseudocódigo do algoritmo `VNS_PCC` proposto para a otimização do terceiro objetivo. Sua estrutura é similar à do algoritmo `VNS_LPRC` para a minimização do segundo objetivo, descrito anteriormente. A única diferença consiste na adição do procedimento de reparo para tornar uma solução viável (linha 1) e dos filtros realizados nas soluções obtidas pela perturbação de inserção de carros (linhas 8 até 10). O algoritmo encerra quando esgotar o tempo para execução (linha 4).

```
procedure VNS_PCC(π_0);
1   if π_0 é inviável then π_0 ← MakeFeasible(π_0)
2   π* ← π_0, φ* ← cost(π*)
3   P ← 1
4   while tempo limite não excedido do
5       s ← s_1, t ← t_1
6       while (P = 1 and s ≤ s_max) or (P = 0 and t ≤ t_max) do
7           π' ← Perturbation(π*, P, N_s^1, N_t^2)
8           if first_objective_cost(π') > first_objective_cost(π*) then
9               π' ← π*
10          end-if
11          π' ← LocalSearch(π')
12          if cost(π') ≤ cost(π*) then
13              π* ← π'
14          end-if
15          if cost(π*) < φ* then
16              s ← s_1, t ← t_1
17              φ* ← cost(π*)
18          else
19              s ← s + 1, t ← t + 1
20          end-if
21      end-while
22      P ← 1 − P
23      π* ← Intensification(π*)
24      φ* ← cost(π*)
25  end-while
26  return π*
end VNS_PCC;
```

Figura 4.6: Pseudocódigo do algoritmo de minimização do terceiro objetivo.