Capítulo 18

Heurísticas para o Problema de Sequenciamento de Carros em Linhas de Montagem

Daniel Brasil*, Thiago Ferreira de Noronha, Caroline Rocha

Resumo: Este capítulo trata do Problema do Sequenciamento de Carros, que consiste em determinar a ordem em que carros devem ser produzidos, de forma a minimizar o número de violações às restrições de capacidade da linha de montagem. As principais heurísticas e estruturas de dados existentes na literatura para o problema são descritas e avaliadas experimentalmente, utilizando instâncias de teste com dados obtidos de fábricas reais. O resultado é um estudo comparativo entre os algoritmos e estruturas de dados, identificando os pontos fortes e fracos de cada um.

Palavras-chave: Problema do sequenciamento de carros, Heurísticas, Meta-heurísticas, Buscal local, Estruturas de dados.

Abstract: *This chapter deals with the Car Sequencing Problem, which consists in defining the order in which cars must be produced so as to minimize the number of violations to the capacity constraints of the assembly line. The main heuristics and data structures existing in the literature for the problem are described and experimentally evaluated by using test instances with data obtained from real factories. The result is a comparative study of the algorithms and data structures, identifying the strengths and weaknesses of each one.*

Keywords: *Car sequencing problem, Heuristics, Metaheuristics, Local search, Data structures.*

1. Introdução

O problema de sequenciamento de carros (PSC) em linhas de produção de indústrias automobilísticas foi descrito pela primeira vez por Parello et al. (1986) e pode ser visto como uma espécie muito particular de problema de escalonamento.

O número de opcionais disponíveis aos compradores tem propiciado o crescimento considerável da diversidade dos carros manufaturados. Devido aos custos da linha de produção, os diferentes veículos são todos montados usando a mesma unidade de montagem. Já que qualquer modificação do fluxo físico dentro da fábrica implica em custo muito elevado, muitos esforços têm sido concentrados na otimização do gerenciamento de recursos de produção. Desta forma, os clientes podem ser supridos com os carros desejados nas datas acordadas a custos mais baixos.

A linha de produção pode ser considerada como um processo de manufatura linear composto por três oficinas (*shops*): oficina de armação de carroceria, oficina de pintura e oficina de montagem. Para programar a linha de produção, o trabalho é dividido em duas etapas:

(1) determinar o dia de produção para cada carro encomendado, de acordo com as capacidades da linha de produção e com os prazos de entrega prometidos aos clientes;

(2) sequenciar os carros a serem produzidos na linha de montagem em cada dia de produção, respeitando os requisitos das oficinas de armação de carroceria, pintura e montagem.

O PSC considera que o conjunto de carros a ser produzido em um determinado dia de produção já é conhecido, restringindo-se, então, ao desenvolvimento da etapa (2) do trabalho de produção nas fábricas. Vale salientar que o conjunto de carros de cada dia de produção, que é determinado na etapa (1), não pode ser mudado na etapa de sequenciamento.

*Autor para contato: daniel@dcc.ufmg.br

Lopes et al. (Eds.), Meta-Heurísticas em Pesquisa Operacional (2013) DOI: 10.7436/2013.mhpo.18 ISBN 978-85-64619-10-4

Somente os requisitos da linha de montagem são considerados na versão clássica do PSC. Desta forma, o problema consiste em encontrar a sequência que melhor satisfaz tais requisitos. O problema de decisão associado ao PSC, que consiste em decidir se é possível encontrar uma sequência satisfazendo todos os requisitos de montagem, foi demonstrado ser NP-Completo por Gent (1998) e por Kis (2004).

Quanto às estratégias de solução, o PSC tem sido tratado frequentemente usando técnicas de programação por restrições (Dincbas et al., 1988; van Hentenryck et al., 1992), que exploram o espaço de busca de uma maneira sistemática. A fim de reduzir o espaço de busca, esta estratégia é combinada com técnicas de filtragem para restringir o domínio das variáveis (Règin & Puget, 1997). Mesmo assim, em algumas instâncias, estas abordagens não conseguem reduzir o domínio suficientemente a ponto de tornar a busca completa tratável. Diante disto, o problema tem atraído a atenção para o desenvolvimento de estratégias que buscam soluções aproximadas (Drexl et al., 2006; Gottlieb et al., 2003; Smith, 1996; Solnon, 2000; Warwick & Tsang, 1995).

O presente trabalho tem como objetivo revisitar as principais heurísticas para resolver o problema de sequenciamento de carros, discutindo os detalhes envolvidos na implementação dos algoritmos, sobretudo no que diz respeito ao uso de estruturas de dados eficientes.

1.1 Requisitos de montagem

A linha de montagem pode ser vista como uma sequência de estações ou unidades de produção projetadas para trabalhar com um carro por vez. Os carros em produção são colocados na linha de montagem, que se move através das unidades de produção para a instalação de opcionais. Alguns opcionais, tais como condicionador de ar, teto solar e equipamentos de som, requerem operações especiais de montagem. No entanto, as unidades de produção tem capacidades limitadas e precisam de tempo para configurar os opcionais a serem instalados à medida que a sequência de carros se move na linha de montagem. Portanto, os carros não podem ser sequenciados arbitrariamente.

A fim de suavizar a carga de trabalho, os carros que requerem operações especiais de montagem precisam ser distribuídos equitativamente ao longo da sequência de carros a serem produzidos, pois estes carros são considerados difíceis de montar. De fato, o total de ocorrência destes carros não deve exceder uma certa quota em qualquer subsequência de carros na linha de montagem, de forma a respeitar as capacidades das unidades de produção.

O requisito de capacidade da unidade para a instalação de um opcional o_i é modelado por uma **restrição de capacidade de razão** N_i/P_i, o que significa que no máximo N_i carros com o opcional i devem ser produzidos em cada subsequência consecutiva de P_i carros na linha de montagem. Por exemplo, se um determinado opcional está associado a uma restrição de capacidade de razão 1/3, não deve existir mais que um carro com este opcional em qualquer sequência consecutiva formada por 3 carros. De forma geral, se existe uma restrição de capacidade de razão $1/P$ para um opcional, dois carros com este opcional devem ser separados por no mínimo $P-1$ carros consecutivos que não possuam este opcional. A Figura 1 ilustra esta situação, onde um carro é representado pelo símbolo "X", se requer o referido opcional, e pelo símbolo "_", caso contrário.

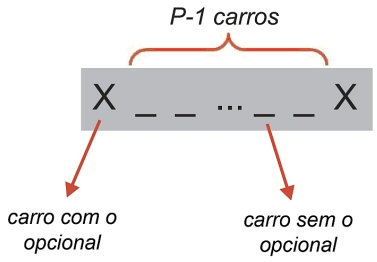

Figura 1. Sequência de carros satisfazendo a restrição de capacidade $1/P$.

As restrições de capacidade mais difíceis de satisfazer são aquelas representadas por razões menores, ou seja, uma restrição da capacidade de razão 1/5 é mais restritiva que uma restrição de capacidade de razão 2/3. Deve-se notar também que uma razão 1/3 é diferente de 2/6, conforme mostra a Figura 2. A sequencia (a) viola a restrição de capacidade de razão 1/3, mas não viola a restrição da capacidade de razão 2/6, como se pode ver na sequência (b).

Uma vez determinado o conjunto de carros para um dia de produção, não se pode garantir de antemão que exista um sequenciamento que respeite todas as restrições de capacidade. Desta forma, tem-se como objetivo minimizar o número de violações destas restrições.

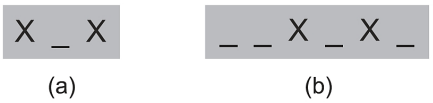

Figura 2. Diferença entre restrições de capacidade 1/3 (a) e 2/6 (b).

1.1.1 Cálculo do número de violações de restrições de capacidade

Sabe-se que a solução ideal para uma restrição de capacidade N_i/P_i associada ao opcional o_i consiste em um dia de produção onde existem no máximo N_i carros com este opcional em qualquer sequência consecutiva de P_i carros. Entretanto, quando não é possível gerar um sequenciamento sem nenhuma violação, os carros associados à mesma restrição devem ser distribuídos da melhor maneira possível. A qualidade da distribuição é dada pelo número de violações nas sequências consecutivas ao longo da linha de produção.

Seja C o número de carros associados a uma restrição de capacidade de razão N/P em uma sequência de tamanho P. O número de violações desta restrição ao longo desta sequência é dado por $N - C$, se $C < N$; 0, caso contrário. A Figura 3 mostra o número de violações de uma restrição de capacidade de razão 1/5 para quatro subsequências de carros.

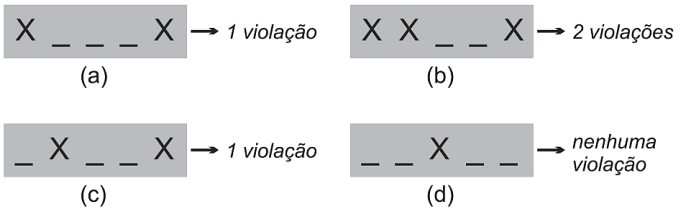

Figura 3. Cálculo do número de violações de uma restrição de capacidade de razão 1/5.

O número total de violações de restrições de capacidade em um dia de produção é igual à soma do número de violações de cada restrição em todas as sequências consecutivas de carros daquele dia. Vale salientar que, para se calcular o número de violações em um dia de produção D, leva-se em consideração os últimos carros do dia anterior, chamado de $D - 1$. Ou seja, para uma restrição de capacidade N/P, a primeira sequência considerada para cálculo das violações contém os $P - 1$ últimos carros sequenciados do dia de produção $D - 1$ e o primeiro carro do dia D.

A Figura 4 mostra um exemplo do número total de violações de uma restrição de capacidade 1/5 ao longo de um dia de produção D em que 6 carros devem ser sequenciados. Para o cálculo do número total de violações da restrição de capacidade 1/5, considera-se um total de 10 subsequências de carros, cujos números de violações podem ser visto na figura, obtendo-se um total de 8 violações desta restrição ao longo do dia de produção.

O objetivo de se calcular o número de violações de restrições de capacidade desta maneira é que violações nas primeiras e últimas sequências contribuam de forma semelhante àquelas ocorridas em sequências intermediárias, uma vez que as violações causadas pelos carros intermediários aparecem em várias sequências consecutivas.

2. Heurísticas Construtivas

Esta seção descreve as principais heurísticas construtivas para PSC encontradas na literatura. Em todos os trabalhos descritos, uma sequência de carros é construída de forma gulosa, escolhendo-se iterativamente o carro que causa o menor número de violações adicionais à sequência em construção. Como muitos carros podem atender a este critério, Gottlieb et al. (2003) propõem uma série de critérios gulosos para decidir empates. Ribeiro et al. (2008) propõem um heurística gulosa que considera critérios de desempate de forma hierárquica, enquanto a heurística construtiva de Solnon (2008) utiliza uma função para atribuir probabilidades de escolha aos carros candidatos. Para facilitar a descrição dos algoritmos, a mesma formalização para PSC proposta em Gottlieb et al. (2003) foi utilizada a seguir.

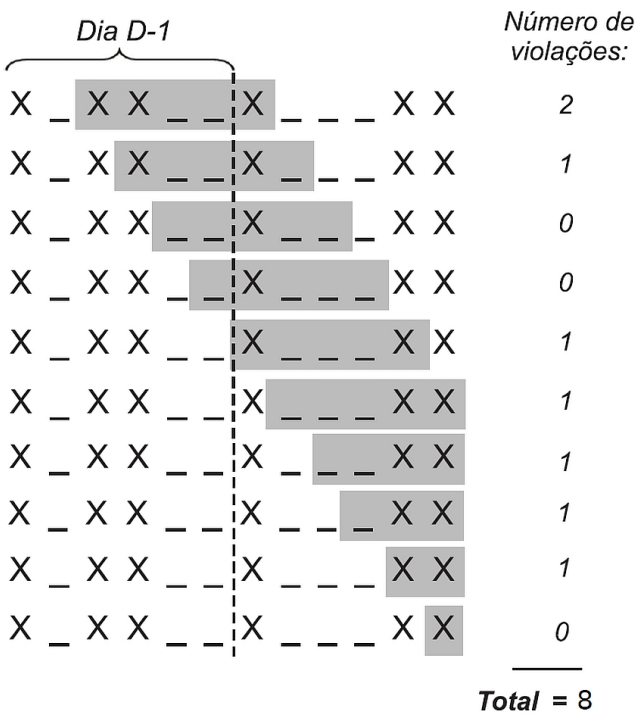

Figura 4. Número de violações de uma restrição de capacidade 1/5 em todas as sequências consecutivas de carros do dia de produção.

2.1 Formalização do problema

O problema de sequenciamento de carros pode ser definido como uma tupla (C, O, p, q, r), onde $C = \{c_1, \ldots, c_n\}$ é o conjunto de carros a serem sequenciados e $O = \{o_1, \ldots, o_m\}$ é o conjunto dos possíveis opcionais requeridos pelos carros. As restrições de capacidade associadas a cada opcional $o_j \in O$ são representadas por uma razão p_j/q_j, que denota que no máximo p_j carros podem demandar o opcional o_j em cada sequência de carros consecutivos de tamanho q_j. As demandas de opcionais são definidas pela função $r : C \times O \to \{0, 1\}$, ou seja, $r(c_i, o_j)$ retorna 1 se o opcional o_j deve ser instalado no carro c_i, e retorna 0 caso contrário.

Uma sequência de carros é denotada por $\pi = \langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$, onde $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$. O número de carros em uma sequência π, denotado por $|\pi|$, é chamado de "tamanho de π". A concatenação de duas sequências π_1 e π_2, denotada por $\pi_1|\pi_2$, é a sequência formada pelos carros de π_1 seguidos pelos carros de π_2. O número de carros que requerem um opcional o_j em uma sequência $\pi = \langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ é dado por

$$r(\pi, o_j) = \sum_{\ell=1}^{|\pi|} r(c_{i_\ell}, o_j). \tag{1}$$

O custo de uma sequência $\pi = \langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ corresponde ao número de violções de restrições de capacidade, o qual é dado por:

$$custo(\pi) = \sum_{j=1}^{m} \sum_{\substack{\pi' \subseteq \pi \\ |\pi'| = q_j}} violação(\pi', o_j), \tag{2}$$

onde

$$violação(\pi', o_j) = \begin{cases} 0 \text{ , se } r(\pi', o_j) \leq p_j; \\ r(\pi', o_j) - p_j \text{ , caso contrário.} \end{cases} \tag{3}$$

Para efeito de simplificação, o número de carros que requerem um opcional o_j em um subconjunto $W \subseteq C$ é denotado por:

$$r(W, o_j) = \sum_{c_i \in W} r(c_i, o_j). \tag{4}$$

Dada uma tupla (C, O, p, q, r), o PSC consiste em encontrar uma sequência π^* tal que $custo(\pi^*)$ é mínimo.

Além da notação acima, existe um conceito importante utilizado pelas heurísticas construtivas que é o da taxa de utilização de um opcional o_j em um conjunto ou uma sequência de carros. Este conceito, apresentado

em Smith (1996), fornece um indicativo do grau de dificuldade de um dado opcional e do problema como um todo. A taxa de utilização de um opcional o_j no conjunto de carros C é dado por:

$$taxaUtil(o_j) = \left(\frac{r(C, o_j)}{n}\right) \Big/ \left(\frac{p_j}{q_j}\right); \tag{5}$$

ou seja, é dada pela razão entre o número de carros demandando o opcional o_j e o número de carros a serem sequenciados, dividida pela razão da restrição de capacidade envolvida. Isto significa que, se a taxa de utilização é maior que 1, a demanda é maior do que a capacidade e, logo, a restrição será violada. Se, por outro lado, a taxa de utilização é muito pequena e próxima de 0, a demanda é muito baixa em relação à capacidade da linha de produção.

2.2 Critérios gulosos

As heurísticas construtivas constroem uma solução inicial escolhendo iterativamente o próximo carro a ser adicionado à sequência de acordo com um critério muito intuitivo: em cada iteração, escolhe-se o carro que causa o menor número de novas violações de restrições quando inserido no final da sequência parcial. De uma maneira mais formal, dada uma sequência parcialmente construída π, o próximo carro c_i a ser adicionado é aquele que minimiza:

$$\delta(\pi, c_i) = \sum_{j=1}^{m} r(c_i, o_j) \cdot violação(últimos(\pi|\langle c_i\rangle, q_j), o_j), \tag{6}$$

onde $últimos(\pi', k)$ retorna a sequência formada pelos últimos k carros de π', se $|\pi'| \geq k$ ou retorna π', caso contrário. Entretanto, geralmente existem vários carros candidatos que minimizam o número de novas violações, o que faz necessária a utilização de critérios gulosos adicionais que são usados para resolver os empates. Tais critérios podem ser:

1. Escolha aleatória: um carro é escolhido aleatoriamente dentre o conjunto de carros candidatos;
2. Maior taxa de utilização estática (Smith, 1996): seleciona-se o carro que requer o opcional com maior taxa de utilização. Se empates persistirem, seleciona-se o carro que requer o opcional com a segunda maior taxa de utilização, e assim por diante;
3. Maior taxa de utilização dinâmica (Gottlieb et al., 2003): é similar ao critério 2, mas a taxa de utilização é atualizada sempre que um novo carro é adicionado à sequência. A taxa de utilização dinâmica de um opcional o_j em relação à sequência parcial π é dada pela seguinte função:

$$taxaUtilDin(o_j, \pi) = \frac{[r(C, o_j) - r(\pi, o_j)] \cdot q_j}{[|C| - |\pi|] \cdot p_j}; \tag{7}$$

4. Soma estática de taxas de utilização (Gottlieb et al., 2003): seleciona-se o carro $c_i \in C \setminus \pi$ que maximiza:

$$\eta_{SS}(c_i, \pi) = \sum_{j=1}^{m} r(c_i, o_j) \cdot taxaUtil(o_j); \tag{8}$$

5. Soma dinâmica de taxas de utilização (Gottlieb et al., 2003): seleciona-se o carro $c_i \in C \setminus \pi$ que maximiza:

$$\eta_{SD}(c_i, \pi) = \sum_{j=1}^{m} r(c_i, o_j) \cdot taxaUtilDin(o_j, \pi); \tag{9}$$

6. Distribuição equitativa (Gottlieb et al., 2003): se o número médio de carros que demandam um opcional é menor na sequência em construção π do que no conjunto total de carros C, então estes carros são favorecidos para entrar na sequência. Formalmente, escolhe-se o carro $c_i \in C \setminus \pi$ que maximiza a função:

$$\eta_{DE}(c_i, \pi) = \sum_{j=1}^{m} \left\{ (r(c_i, o_j) = 0) \text{ XOR } \left(\frac{r(C, o_j)}{n} > \frac{r(\pi, o_j)}{|\pi|}\right)\right\}. \tag{10}$$

   O resultado do operador XOR será verdadeiro em dois casos: quando já existirem muitos carros sequenciados com o opcional o_j e o carro c_i não possuir este opcional, ou quando existirem poucos carros sequenciados com o opcional o_j e o carro c_i possuir o opcional. Portanto, o carro cujos opcionais atenderem mais vezes a esta condição será escolhido para entrar na solução. O primeiro carro a ser sequenciado é escolhido aleatoriamente entre os carros com o número máximo de opcionais, uma vez que a fórmula acima não pode ser aplicada quando $|\pi| = 0$.

Em Gottlieb et al. (2003), diferentes heurísticas construtivas são propostas usando cada um dos seis critérios de desempate. Em Ribeiro et al. (2008), é desenvolvida uma heurística que usa os critérios 6 e 5, nesta ordem, para decidir empates. Se, ainda assim, houver mais de um carro candidato, um dentre eles é escolhido aleatoriamente. Em Solnon (2008), os carros que minimizam o número de violações adicionais à sequência em construção π formam uma lista de candidatos, denotada por *cand*. Então, o próximo carro a ser inserido é escolhido aleatoriamente a partir de *cand* baseado em uma função de probabilidade dada por:

$$p(c_i, cand, \pi) = \frac{[\eta_{SD}(c_i, \pi)]^\beta}{\sum_{c_k \in cand} [\eta_{SD}(c_k, \pi)]^\beta}, \tag{11}$$

para todo $c_i \in cand$, onde β é um parâmetro que permite ajustar a intensidade do critério guloso. Quanto maior o valor de β, mais gulosa será a estratégia.

3. Estruturas de Vizinhança

As principais heurísticas de busca local encontradas na literatura para o problema de sequenciamento de carros exploram a vizinhança de uma solução $\pi = \langle c_{k_1}, \ldots, c_{k_n} \rangle$ através da aplicação dos seguintes tipos de movimento:

- `troca`(π, i, j): troca os carros que estão nas posições i e j. Por exemplo, se $i < j$, este movimento resulta na sequência $\pi' = \langle c_{k_1}, \ldots, c_{k_{i-1}}, c_{k_j}, c_{k_{i+1}}, \ldots, c_{k_{j-1}}, c_{k_i}, c_{k_{j+1}}, \ldots, c_{k_n} \rangle$.
- `insere`(π, i, j): remove um carro de sua posição atual i e o insere em uma nova posição j. Por exemplo, se $i < j$, a aplicação deste movimento resulta na sequência $\pi' = \langle c_{k_1}, \ldots, c_{k_{i-1}}, c_{k_{i+1}}, \ldots, c_{k_{j-1}}, c_{k_j}, c_{k_i}, c_{k_{j+1}}, \ldots, c_{k_n} \rangle$.
- `inverte`(π, i, j): inverte a subsequência de carros da posição i até a posição j, com $i < j$. Por exemplo, se $i = 1$ e $j = 4$, a aplicação deste movimento resulta na sequência $\pi' = \langle c_{k_4}, c_{k_3}, c_{k_2}, c_{k_1}, c_{k_5}, c_{k_6}, \ldots, c_{k_n} \rangle$.
- `embaralha`(π, i, j): embaralha de forma aleatória a subsequência de carros começando na posição i e terminando na posição j, com $i < j$. Por exemplo, se $i = 1$ e $j = 4$, a aplicação deste movimento pode resultar na sequência $\pi' = \langle c_{k_3}, c_{k_1}, c_{k_4}, c_{k_2}, c_{k_5}, c_{k_6}, \ldots, c_{k_n} \rangle$.

4. Meta-Heurísticas

Diversas heurísticas têm sido propostas para o PSC, dentre as quais pode-se citar algoritmos baseados nas meta-heurísticas Colônia de Formigas (Solnon, 2000, 2008; Gagné et al., 2006), Algoritmos Genéticos (Warwick & Tsang, 1995; Cheng et al., 1999), Busca em Vizinhança Variável (Ribeiro et al., 2008; Gavranovic, 2008), Busca Tabu (Gavranovic, 2008), Busca Local Iterativa (Ribeiro et al., 2008) e abordagens híbridas (Estellon et al., 2008). Foram selecionadas três destas heurísticas, descritas a seguir.

4.1 Algoritmo de colônia de formigas

O algoritmo de colônia de formigas proposto por Solnon (2008) para o PSC é baseado em uma combinação de duas estruturas de feromônio utilizadas para guiar a busca no espaço de soluções: a primeira tem o objetivo de identificar sequências promissoras de carros; a segunda, de identificar carros mais difíceis de serem sequenciados.

Usando a primeira estrutura, trilhas de feromônio $\tau_1(c_i, c_j)$ são depositadas em pares de carros $(c_i, c_j) \in C \times C$, representando a experiência passada da colônia de formigas com relação a sequenciar o carro c_j logo após o carro c_i. Para esta estrutura de feromônio, as trilhas são limitadas pelo intervalo $[\tau_{min_1}, \tau_{max_1}]$, começando no limite superior do intervalo. Em cada ciclo, após todas as formigas terem construído uma sequência completa, as trilhas de feromônio sofrem evaporação e somente as formigas mais aptas depositam feromônio, respeitando os limites do intervalo $[\tau_{min_1}, \tau_{max_1}]$, conforme descrito pelo pseudocódigo da Figura 5. O parâmetro ρ_1 na linha 2 representa a taxa de evaporação do feromônio.

Usando a segunda estrutura de feromônio, as formigas depositam uma quantidade de feromônio $\tau_2(cc)$ em classes de carros $cc \in classes(C)$, onde uma classe de carros é o subconjunto de carros que requerem os mesmos opcionais e $classes(C)$ é o conjunto de todas as classes existentes em C. A função $classeDe(c_i) = \{o_j \in O | r(c_i, o_j) = 1\}$ diz qual a classe do carro c_i. A quantidade de feromônio $\tau_2(cc)$ representa a experiência passada com relação à dificuldade de sequenciar carros desta classe sem violar restrições de capacidade. Um limite inferior τ_{min_2} é imposto para $\tau_2(cc)$ e usado como valor inicial no algoritmo. Ao contrário da atualização da primeira estrutura de feromônio, toda formiga deposita feromônio enquanto constrói uma sequência. Ou seja, em cada iteração do processo construtivo, o feromônio é depositado antes de escolher o próximo carro

```
1  para ∀(c_i, c_j) ∈ C × C faça
2      τ_1(c_i, c_j) ← max{(1 − ρ_1) · τ_1(c_i, c_j), τ_min_1}
3  fim-para

4  se custo(π) é o menor do ciclo atual então
5      para ∀ par de carros consecutivos ⟨c_i, c_j⟩ ⊆ π faça
6          τ_1(c_i, c_j) ← min{τ_1(c_i, c_j) + 1/custo(π), τ_max_1}
7      fim-para
8  fim-se
```

Figura 5. Pseudocódigo da evaporação (linhas 1-3) e do depósito (linhas 4-8) da primeira estrutura de feromônio.

a ser inserido, de acordo com o pseudocódigo da Figura 6, onde $\delta(\pi, c_j)$ é a variação no custo da sequência π com a adição do carro c_j, como mostra a Equação 6. No entanto, o processo de evaporação é similar ao anterior, ou seja, acontece somente após a construção de cada sequência.

```
1  se δ(π, c_j) > 0, ∀c_j ∈ cand então
2      para ∀ classe de carros cc ∈ {classeDe(c_i)|c_i ∈ C \ π} faça
3          τ_2(cc) ← τ_2(cc) + δ(π, c_i)
4      fim-para
5  fim-se
```

Figura 6. Pseudocódigo do depósito da segunda estrutura de feromônio.

As formigas constroem sequências de carros conforme o algoritmo descrito na Seção 2. No entanto, a função de probabilidade, usada na escolha do próximo carro c_i adicionado à sequência em construção π, depende de dois fatores de feromônio, como expresso abaixo:

$$p(c_i, cand, \pi) = \frac{[\tau_1(c_j, c_i)]^{\alpha_1} \cdot [\tau_2(classeDe(c_i))]^{\alpha_2}}{\sum_{c_k \in cand} [\tau_1(c_j, c_k)]^{\alpha_1} \cdot [\tau_2(classeDe(c_k))]^{\alpha_2}}, \tag{12}$$

onde c_j é o último carro da sequência π e $classeDe(c_j)$ representa a classe de carros à qual c_j pertence. Os parâmetros α_1 e α_2 são usados para atribuir pesos relativos aos dois fatores de feromônio.

4.2 Algoritmo de busca local rápida

O algoritmo de busca local rápida, do inglês *Very Fast Local Search* (VFLS), foi proposto por Estellon et al. (2008) para o desafio ROADEF 2005, uma competição organizada bienalmente pela Sociedade Francesa de Pesquisa Operacional (ROADEF, 2005). O problema de sequenciamento de carros abordado na competição considera restrições adicionais relacionadas à cor dos carros a fim de minimizar o uso de solventes, além de considerar diferentes níveis de prioridade entre as restrições de capacidade. A heurística VFLS descrita a seguir foi adaptada em Solnon (2008) para resolver a versão clássica do problema.

A heurística VFLS é um método de busca baseado na exploração rápida de pequenas vizinhanças, cujo pseudocódigo pode ser visto na Figura 7. Inicialmente, uma sequência de carros é construída usando uma heurística construtiva baseada no critério guloso η_{SD} descrito na Seção 2.2. Em cada iteração, o algoritmo seleciona o primeiro vizinho que não deteriora o custo da solução atual. Para tanto, escolhe-se uma dentre cinco estruturas de vizinhança (linha 3) e uma solução vizinha de acordo com a estrutura selecionada (linhas 4 e 5). As quatro estruturas de vizinhança descritas na Seção 3 são consideradas no algoritmo. A estrutura de vizinhança a ser explorada em cada iteração é selecionada aleatoriamente de acordo com probabilidades definidas previamente. O algoritmo termina quando a condição de parada, que pode ser, por exemplo, tempo limite máximo ou numero máximo de iteração, for satisfeita.

O modo mais simples de se escolher as posições i e j para se obter uma solução vizinha é sorteá-los aleatoriamente. Esta estratégia é muito usada no algoritmo VFLS, sobretudo nas primeiras iterações. Entretanto, estratégias mais agressivas são adotadas à medida que a busca por melhores soluções torna-se mais difícil. A estratégia de escolha das posições envolvidas no movimento depende da estrutura de vizinhança na qual o movimento é aplicado, conforme mostrado na Tabela 1. Uma vez selecionada a estrutura de vizinhança, valores de probabilidades são utilizados para definir a chance de se escolher cada uma das estratégias para a obtenção de um vizinho da solução atual.

```
procedimento VFLS
1  π ← sequência incial
2  enquanto condição de parada não satisfeita faça
3      Escolha uma estrutura de vizinhança
4      Escolha as posições (i, j) envolvidas no movimento
5      π′ ← movimento(π, i, j)
6      se custo(π′) ≤ custo(π) então
7          π ← π′
8      fim-se
9  fim-enquanto
10 retorne π
```

Figura 7. Algoritmo VFLS para o PSC (Estellon et al., 2008).

Estratégia	Descrição	Vizinhanças
Genérica	Seleciona i e j aleatoriamente	`troca`, `insere`, `inverte`, `embaralha`
Similar	Seleciona i e j tais que ambos compartilhem algum opcional	`troca`
Consecutiva	Seleciona i aleatoriamente e faz $j = i+1$	`troca`
Violação	Seleciona i em uma posição onde ocorre uma violação e j aleatoriamente	`troca`
Denominador	Seleciona i e um opcional k aleatoriamente e faz $j = i + q(k)$	`insere`

Tabela 1. Estratégias de seleção das posições envolvidas nos movimentos.

4.3 Algoritmo de busca local iterativa

Ribeiro et al. (2008) desenvolveram um sofisticado algoritmo para o PSC baseado na meta-heurística de busca local iterativa, do inglês *Iterated Local Search* (ILS), também para o desafio ROADEF (ROADEF, 2005). A heurística ILS prosposta é baseada na exploração rápida de vizinhanças graças ao uso de estrutura de dados eficientes, utilizando os quatro procedimentos fundamentais descritos a seguir:

- `Pertubação`(π): realiza um pequeno número de movimentos do tipo `insere` com carros envolvidos em violações na solução π;
- `BuscaLocal`(π): realiza busca local exaustiva na solução π usando movimentos do tipo `troca`;
- `Intensificação`(π): consiste na aplicação sucessiva de dois procedimentos de busca local: o primeiro realiza movimentos do tipo `insere` e o segundo com movimentos do tipo `troca`, nesta ordem;
- `Diversificação`(π): consiste em reiniciar a busca a partir de uma nova solução, que pode ser obtida através da aplicação de uma grande perturbação na solução π (removendo e reinserindo um grande número de carros) ou da aplicação da heurística construtiva.

O pseudocódigo da Figura 8 apresenta uma descrição geral da heurística ILS. A melhor solução conhecida π^* é inicializada com a solução inicial vinda da heurística construtiva, conforme descrita na Seção 2. Cada iteração da heurística ILS começa com uma perturbação na solução atual π (linha 3), seguida de busca local na solução vinda da perturbação (linha 4). A nova solução π' é aceita se seu custo for menor ou igual ao da solução atual π (linha 5). O procedimento de intensificação é executado na linha 7 se nenhuma melhoria for realizada na solução atual π após α iterações. O procedimento de diversificação é realizado na linha 11 se a solução atual π não tiver sido aprimorada nas últimas β iterações (com $\alpha < \beta$) e se o custo de π é igual ao da melhor solução conhecida. Caso contrário, recomeça-se a busca a partir da melhor solução conhecida (linha 12). A melhor solução π^* é atualizada, se necessário, na linha 14. Por fim, π^* é retornada na linha 16.

```
procedimento ILS(π)
1  π* ← π
2  enquanto condição de parada não satisfeita faça
3      π' ← Perturbação(π)
4      π' ← BuscaLocal(π')
5      se custo(π) ≤ custo(π') então s ← π'
6      se o número de iterações desde a última melhoria
       em π for igual a α então
7          π ← Intensificação(π)
8      fim-se
9      se o número de iterações desde a última melhoria
       em π for igual a β então:
10         se custo(π) = custo(π*) então
11             π ← Reinício(s)
12         senão π ← π*
13     fim-se
14     se custo(π) < custo(π*) então π* ← π
15 fim-enquanto
16 retorne π*
fim ILS
```

Figura 8. Pseudocódigo da heurística ILS para PSC (Ribeiro et al., 2008).

5. Detalhes de Implementação

As estruturas de dados para representar as informações e a solução atual do problema são um aspecto importante na implementação das heurísticas e têm um papel crucial em sua complexidade computacional e, consequentemente, no seu desempenho. As estruturas de dados devem ser eficientes do ponto de vista das operações que deverão ser realizadas sobre elas, principalmente no caso dos métodos de busca local, uma vez que, normalmente, esta é a componente das heurísticas que demanda mais tempo computacional. Esta seção concentra-se nas estruturas de dados utilizadas na implementação da busca local `troca`, já que esta é a mais utilizada dentre todas as buscas locais para o PSC. As estruturas de dados utilizadas em Estellon et al. (2008) são apresentadas na Seção 5.1, enquanto as estruturas de dados propostas em Ribeiro et al. (2008) são apresentadas na Seção 5.2.

5.1 Estruturas de dados de Estellon et al. (2008)

Uma solução para o problema, ou seja, uma sequencia π de carros, é representada por um vetor, aqui denotado por V, onde cada posição guarda um número que representa o carro que deve ser produzido naquela posição ao longo da linha de montagem. Além deste vetor, para cada uma das restrições de capacidade $o_j \in O$, um outro vetor, denotado por X_j, armazena em cada posição o valor 1 ("X") se o carro que está naquela posição requer o opcional o_j; ou 0 ("_"), caso contrário. Desta forma, o cálculo do custo de um vizinho obtido pela troca dos carros nas posições i e k da solução pode ser calculado para cada restrição de capacidade separadamente. O custo total do vizinho é dado pela soma do custo das violações para todos os opcionais.

Trocar dois carros que não estão associados a uma determinada restrição certamente não altera o número de violações desta restrição. O mesmo acontece quando se deseja trocar dois carros que estão associados a uma mesma restrição. Portanto, é necessário analisar apenas trocas de dois carros em que um dos carros está associado a uma dada restrição e o outro não. Para se obter a variação no número de violações a uma restrição p_j/q_j geradas pela retirada ou inserção de um carro, varrem-se todas as q_j sequências às quais o carro pertence. Para cada sequência que violar a restrição de capacidade, adiciona-se 1, no caso da inserção, ou subtrai-se 1, no caso da remoção.

Sejam n o número de carros na solução, m o número de restrições de capacidade e Q o denominador máximo dentre todas as restrições, a complexidade de pior caso da avaliação de um movimento `troca`(π, i, k) é $O(Q \cdot m)$, uma vez que a variação no custo do vizinho tem que ser feito para cada uma das m restrições de capacidade.

A aplicação do movimento na solução, ou seja, a atualização das estruturas de dados é feita em tempo $O(1)$ para os vetores V e X_j, já que consiste apenas em trocar os valores armazenados nas posições i e

k envolvidas na troca. Desta forma, a complexidade de pior caso da aplicação de um movimento `troca` é $O(1) + m \cdot O(1) = O(m)$.

5.2 Estruturas de dados de Ribeiro et al. (2008)

As estruturas de dados utilizadas em Estellon et al. (2008) são as mais intuitivas para representar uma solução para o PSC, mas não necessariamente as mais eficientes para a implementação dos algoritmos de busca local. Além dos vetores V e X_j, $\forall o_j \in O$, Ribeiro et al. (2008) propuseram a utilização de três vetores adicionais para cada $o_j \in O$. Estes vetores auxiliam na avaliação da variação de custo causada pela aplicação de um movimento de `troca` na solução.

O primeiro vetor associado a uma restrição de capacidade de razão p_j/q_j, denominado A_j, $\forall o_j \in O$, armazena o número de carros que requerem o opcional o_j em cada sequência consecutiva da solução corrente. O valor armazenado na posição i deste vetor representa o número de carros associados à restrição na sequência de q_j carros consecutivos que começa na posição i.

O exemplo da Figura 9 ajuda a entender melhor os valores que são armazenados em A_j. Neste exemplo, a restrição de capacidade considerada tem razão 1/4. As sequências consideradas ao longo da solução são ilustradas na figura e o número de carros associados à restrição em cada sequência bem como a posição correspondente no vetor de sequências podem ser vistos ao lado. Por exemplo, na primeira posição do vetor, tem-se o valor 3, que corresponde ao número de carros associados à restrição na sequência formada pelos quatro primeiros carros da solução. A segunda posição do vetor de sequências ter armazenado o valor 2, correspondendo ao número de carros associados à restrição na sequência que começa no carro sequenciado na segunda posição, e assim sucessivamente, até a última sequência da solução.

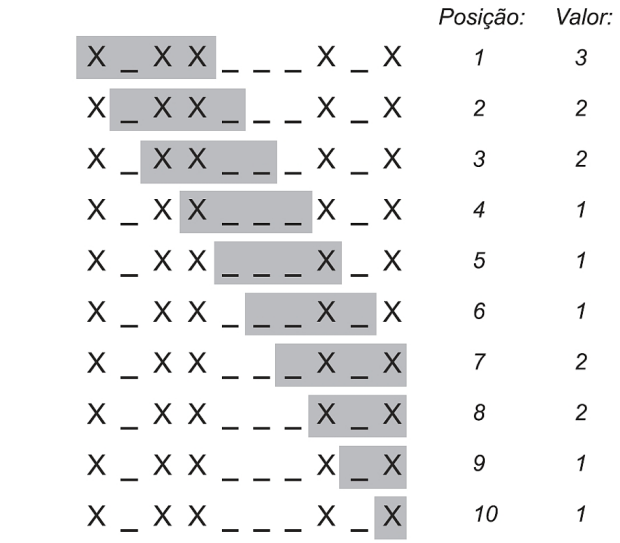

Figura 9. Vetor que guarda o número de carros associados a uma restrição em cada sequência.

O segundo vetor utilizado, denominado B_j, $\forall o_j \in O$, armazena em cada posição o número de sequências em que o número de carros associados a uma certa restrição na sequência é maior que o numerador da razão desta restrição, ou seja, o número de sequências que estão violando a restrição desde o início da solução até aquela posição. O valor armazenado na posição i do vetor B_j associado a uma restrição de razão p_j/q_j corresponde ao número de sequências que violam esta restrição, considerando-se a sequência que começa na posição 1 até a sequência da posição i.

A Figura 10 ilustra o vetor B_j referente à mesma restrição de capacidade de razão 1/4 para a mesma solução apresentada na Figura 9. Com o auxílio do vetor A_j, que guarda o número de carros associados à restrição em cada sequência, pode-se ver que a primeira posição de B_j tem o valor 1, o que corresponde a uma sequência violando a restrição até esta posição do vetor A_j, ou seja, uma sequência que possui valor em A_j maior que $p_j = 1$. Na segunda posição, o valor armazenado é 2, correspondendo às duas primeiras sequências e, na terceira posição, tem-se três sequências violando a restrição até esta posição. Na quarta posição, o valor continua igual a três, pois a sequência desta posição não viola a restrição. Desta forma, todas as posições de B_j são calculadas.

O terceiro vetor, denominado C_j, $\forall o_j \in O$, é construído de forma análoga ao vetor B_j. A diferença é que são contabilizadas as sequências cujo valor em A_j é maior ou igual ao numerador da razão da restrição

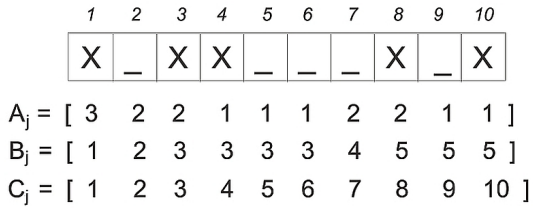

Figura 10. Vetores auxiliares correspondentes a uma solução.

considerada. A ideia é contabilizar as sequências em que a entrada de mais um carro associado à restrição aumenta o número de violações. Pode-se observar que isto inclui sequências que não violam a restrição, porém cujo valor no vetor A_j é igual ao numerador da razão, valor máximo permitido para respeitar a restrição. Se um carro associado a esta restrição é adicionado a estas sequências, elas passam então a violar a restrição. A Figura 10 apresenta o vetor C_j referente à mesma restrição de razão $1/4$ para a mesma solução apresentada na Figura 9.

O cálculo do custo de um vizinho obtido pela troca dos carros nas posições i e k da solução é apresentado a seguir. Assim como no caso de Estellon et al. (2008), este cálculo pode ser feito para cada um dos opcionais separadamente. O custo total do vizinho é dado pela soma das violações para todos os opcionais. A aplicação do movimento de trocar dois carros que não estão associados a uma determinada restrição, certamente, não causa variação no custo da solução. O mesmo acontece quando se deseja trocar dois carros que estão associados a uma mesma restrição. Portanto, é analisada apenas a troca de dois carros onde um dos carros está associado a uma dada restrição e o outro não. O número de violações de uma restrição de capacidade de razão p_j/q_j que é decrescido no custo da solução com a retirada do carro associado a esta restrição da posição i é obtido a partir do vetor B_j correspondente e é denotado por Δ_1. Como um carro pode diminuir ou aumentar somente uma violação em uma sequência, então é suficiente calcular quantas sequências, que possuem este carro, estão violando a restrição considerada. Desta forma, o valor de Δ_1 é dado por:

$$\Delta_1 = \begin{cases} B_j[i] - B_j[i - q_j], \text{ se } i - q_j > 0; \\ B_j[i], \text{ caso contrário.} \end{cases} \tag{13}$$

O valor contido em $B_j[i]$ corresponde ao número de sequências violando a restrição desde a primeira sequência até a sequência que começa na posição i, que é a última sequência que contém o carro na posição i da solução. A posição $i - q_j$ refere-se à última sequência à qual o carro da posição i não pertence, uma vez que um carro está contido em exatamente q_j sequências. Portanto, a subtração destes dois valores resulta no número de sequências que contêm o carro da posição i e que violam a restrição. Se o valor de $i - q_j$ é menor que zero, significa que o carro aparece no início da solução e, portanto, $B_j[i]$ é o valor desejado. Então, Δ_1 representa o número de violações da restrição que é reduzido se a troca dos carros considerados for realizada.

De maneira análoga, o número de violações adicionais de uma restrição decorrente da entrada de um carro associado à restrição na posição k, que não continha um carro associado a ela, é calculado a partir do vetor C_j e denotado por Δ_2:

$$\Delta_2 = \begin{cases} C_j[k] - C_j[k - q_j], \text{ se } k - q_j > 0; \\ C_j[k], \text{ caso contrário.} \end{cases} \tag{14}$$

Conforme foi dito anteriormente, este vetor guarda o número de sequências em que a entrada de um carro associado à restrição correspondente aumenta o número de violações. Finalmente, a variação no custo causada pela troca destes dois carros é dada por $\Delta = \Delta_2 - \Delta_1$.

No exemplo da Figura 10, suponha que se deseja avaliar a variação no custo causada pela inversão dos carros das posições 3 e 7. Vale lembrar que a restrição de capacidade considerada tem razão $1/4$. O valor que será acrescido devido à entrada de um carro com a restrição na posição 7 é dado por $\Delta_2 = C_j[7] - C_j[3] = 4$. Por outro lado, o valor que será reduzido decorrente da entrada de um carro sem a restrição na posição 3 é dado por $\Delta_1 = B_j[3] = 3$. Assim, a aplicação deste movimento na solução resultará no acréscimo de uma violação desta restrição.

Uma atenção especial deve ser dada quando a distância entre as posições i e k dos carros que se deseja trocar é menor do que o denominador da razão da restrição que se está considerando, ou seja, quando $|i - k| < q_j$. Quando isto acontece, os dois carros pertencem a uma ou mais sequências em comum e estas sequências não devem ser consideradas na avaliação do movimento, uma vez que ambos os carros continuarão nestas sequências e, por conseguinte, o número de violações nelas não sofrerá alteração se o movimento for aplicado. Diante disto, as violações decrescidas decorrentes da troca do carro da posição i associado a uma restrição

de capacidade de razão p_j/q_j com o carro da posição k, que não está associado a esta restrição, se $i < k$ e $k - i < q_j$, é dado por:

$$\Delta_1 = \begin{cases} B_j[k - q_j] - B_j[i - q_j], \text{ se } i - q_j > 0 \text{ e } k - q_j > 0; \\ B_j[k - q_j], \text{ se } i - q_j \leq 0 \text{ e } k - q_j > 0; \\ 0, \text{ caso contrário.} \end{cases} \quad (15)$$

Por sua vez, o número de violações que será aumentado em decorrência da aplicação do movimento, quando $i < k$ e $k - i < q_j$, é dado por:

$$\Delta_2 = C_j[k] - C_j[i]. \quad (16)$$

Observa-se que a última sequência considerada no cálculo de Δ_1 começa na posição $k - q_j$, que é a última sequência que contém o carro em i antes do intervalo de sequências em comum. O valor de Δ_1 pode ser $B_j[k - q_j]$ ou 0, quando um ou ambos os carros estão na primeira sequência do dia de produção, respectivamente. Analogamente, pode-se observar que o cálculo de Δ_2 começa com a sequência da posição $k - i + 1$, pois esta é a primeira sequência após o intervalo de sequências em comum entre os dois carros. Quando $i > k$, a forma como os vetores são acessados é invertida: B_j é acessado de forma análoga a C_j, e vice-versa.

Na Figura 11, continua-se com o exemplo das Figuras 9 e 10. Deseja-se trocar os carros da posição $i = 4$, que está associado à restrição, e da posição $k = 6$, que não está associado a ela. Tem-se, portanto, as sequências das posições 3 e 4 como sequências em comum entre os carros. Como o carro da posição 4 está na primeira sequência, mas o outro carro não está, pela Equação 15, tem-se que $\Delta_1 = B_j[6 - 4] = 2$, enquanto que, pela Equação 16, $\Delta_2 = C_j[6] - C_j[4] = 2$. Portanto, este movimento não causa alteração no número de violações desta restrição ($\Delta = \Delta_2 - \Delta_1 = 0$).

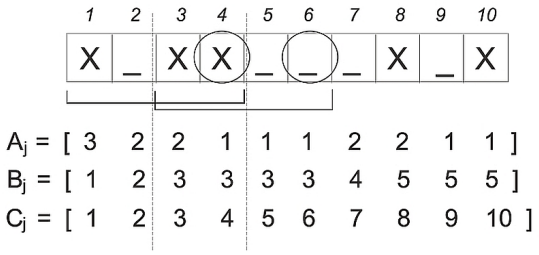

Figura 11. Vetores auxiliares correspondentes a uma solução.

Sejam n o número de carros na solução, m o número de restrições de capacidade e Q o denominador máximo dentre todas as restrições. A complexidade da avaliação de um movimento demanda tempo $O(m)$, uma vez que ela é feita através das fórmulas dadas anteriormente em tempo $O(1)$ para cada uma das m restrições de capacidade.

A aplicação de movimento `troca` na solução, ou seja, a atualização das estruturas de dados tem a seguinte complexidade assintótica no pior caso. O custo computacional de atualizar o vetor V é $O(1)$ e o custo computacional de atualizar todos os vetores X_j é $O(m)$, tal como calculado na Subseção anterior. Para atualizar o vetor A_j de uma restrição de razão p_j/q_j, deve-se atualizar exatamente q_j posições do vetor para cada um dos carros trocados, correspondendo a cada uma das sequências a que eles pertencem. Assim, a atualização dos vetores A_j de todas as restrições resulta em uma complexidade $O(Q \cdot m)$, pois $q_j \leq Q$ para todo $o_j \in O$. A atualização dos vetores B_j e C_j de uma restrição de razão p_j/q_j, decorrente da troca dos carros das posições i e k, implica, de forma geral, na alteração de todas as posições dos vetores a partir da posição $i - q_j$, se $i < k$, ou $k - q_j$, caso contrário. Esta atualização é necessária pois os valores armazenados nestes vetores correspondem a valores que são acumulados desde o início da solução e a alteração de qualquer posição deste vetor influenciará em todas as posições subsequentes. Desta forma, a atualização dos vetores B_j e C_j de todas as restrições demanda tempo computacional $O(n \cdot m)$. Isto implica que toda a atualização necessária após a aplicação de um movimento `troca` é $O(1)+O(m)+O(Q \cdot m)+O(n \cdot m)+O(n \cdot m) = O(n \cdot m)$, pois $Q \leq n$.

6. Experimentos Computacionais

Na seção anterior, observou-se que o custo computacional de se consultar o custo de um vizinho obtido pela troca de dois carros utilizando as estruturas de dados de Estellon et al. (2008) é $O(Q \cdot m)$, enquanto o custo

de efetuar a troca de dois carros em uma solução e atualizar as respectivas estruturas de dados é $O(m)$, onde n é o número de carros na solução, m é o número de restrições de capacidade e Q é o denominador máximo dentre todas as restrições de capacidade. Também observou-se que o custo computacional de se consultar o custo de um vizinho obtido pela troca de dois carros utilizando as estruturas de dados de Ribeiro et al. (2008) é $O(m)$, enquanto o custo de efetuar a troca de dois carros em uma solução e atualizar as respectivas estruturas de dados é $O(n \cdot m)$. Desta forma, pode-se dizer que as estruturas de dados de Ribeiro et al. (2008) são mais eficientes do que as de Estellon et al. (2008) quanto ao cálculo do custo de um vizinho. Por outro lado, pode-se dizer que as estruturas de dados de Estellon et al. (2008) são mais eficientes do que as de Ribeiro et al. (2008) quanto à atualização das estruturas de dados, uma vez que uma troca é efetivamente realizada.

Nesta seção, observa-se o impacto destas estruturas de dados em dois contextos diferentes. Na Seção 6.1, as duas estruturas de dados são comparadas num contexto de uma heurística de busca local, onde o número de consultas de custo de vizinhos é geralmente bem maior que a quantidade de trocas de carro que são efetivamente realizadas. Em seguida, na Seção 6.2 as duas estruturas de dados são comparadas num contexto de uma heurística VFLS, onde o número de consultas de custo de vizinhos é próximo do número de trocas que são efetivamente realizadas.

Os experimentos foram executados em um processador *Intel Core 2 Duo* CPU P8600 a 2.40GHz, com 3072 KBytes de memória *cache* e 3 GBytes de memória RAM. O sistema operacional utilizado foi o Ubuntu versão 11.10. Os algoritmos foram codificados em C++ e compilados com a versão 4.6.1 do GNU/GCC.

As instâncias utilizadas nos experimentos foram geradas a partir do conjunto de instâncias de teste fornecido pela montadora Renault para o *ROADEF Challenge* 2005 (ROADEF, 2005) e tratam-se de instâncias semelhantes às de fábricas reais. As instâncias do ROADEF contêm dois tipos de restrição, descritas como restrições de alta prioridade (HPRC – *High Priority Ratio Constraint*) e baixa prioridade (LPRC – *Low Priority Ratio Constraint*). A Tabela 2 mostra as especificações de cada instância. A primeira coluna apresenta o identificador da instância e a segunda mostra todas as restrições de razão daquela instância. Por uma questão de espaço, as restrições que possuem o mesmo numerador são agrupadas. A terceira coluna contém o número de carros da instância.

Instâncias	**Restrições de razão**	**Carros**
HPRC_022	1/(3); 5/(6)	526
HPRC_023	1/(2,2,3,6,8,10); 3/(4); 4/(5); 6/(7)	1110
HPRC_024	1/(2,2,3,8,10,50)	1270
HPRC_025	1/(4,8); 2/(3,3)	1161
HPRC_028_1	1/(2,3,6,7,8,10,10,12,20)	365
HPRC_028_2	1/(6)	65
HPRC_029	1/(6,7); 3/(5); 2/(5)	730
HPRC_035_1	1/(5,6)	1284
HPRC_035_3	1/(2,15); 2/(3)	269
HPRC_039	1/(3,5)	1000
HPRC_048_1	1/(2,2,3,4,4); 2/(3)	591
HPRC_048_2	1/(2,3,4,4,6,8,12)	546
HPRC_064_1	1/(2,4,5,7,8,14,30); 3/(4); 2/(3); 4/(5); 16/(24)	825
HPRC_064_2	1/(2,2,5,40)	412
LPRC_022	1/(3,3,4,4,6,10); 10/(15)	526
LPRC_023	1/(2,2,2,2,2,3,4); 2/(3)	1110
LPRC_024	1/(5,7,25,40,40,40,50)	1270
LPRC_025	1/(2,2,3,3,3,6,7,18,20,97); 3/(5); 2/(3)	1161
LPRC_028_1	1/(2,5,8,8,10,12,13,14); 2/(3,7); 4/(5)	365
LPRC_028_2	1/(2,2,3,4,5,5,28); 2/(3)	65
LPRC_029	1/(2,3); 36/(43)	730
LPRC_035_1	1/(2,3)	1284
LPRC_035_3	1/(2,25)	269
LPRC_039	1/(11,12,13,13,36,90,120); 40/(60,60)	1000
LPRC_048_1	1/(3,4,4,4,5,6,6,7,8,8,10,12,12,50,310,310,310); 9/(10); 2/(3)	591
LPRC_048_2	1/(3,3,4,5,5,5,6,8,8,8,10,11,16,20,50,50)	546
LPRC_064_1	1/(3,6,22)	825
LPRC_064_2	1/(21)	412

Tabela 2. Especificação do conjunto de instâncias de teste utilizadas nos experimentos computacionais.

6.1 Buscas locais

Nesta seção observa-se o impacto das estruturas de dados de Estellon et al. (2008) e de Ribeiro et al. (2008) no desempenho de um algoritmo de busca local sob a vizinhança `troca`. Foram implementadas duas variações da busca local. A busca local Primeiro Aprimorante (PA) realiza efetivamente a troca de dois carros assim que for encontrada uma troca que diminui o custo da solução. Já a busca local Melhor Aprimorante (MA) testa todas as trocas de carros possíveis e só então realiza a melhor troca, ou seja, aquela dentre todos os vizinhos que mais reduz o número de violações na solução.

As buscas locais PA e MA foram implementadas com as duas estruturas de dados descritas na seção anterior. A geração de soluções iniciais foi feita utilizando uma permutação aleatória dos carros da instância. Foi medido o tempo computacional de cada implementação até que um ótimo local fosse atingido. Vale salientar que, independente da estrutura de dados utilizadas, PA e MA realizam, respectivamente, as mesmas operações ao longo da sua execução. Entretanto, o tempo computacional para realizar cada operação varia com a estrutura de dados utilizada.

Os resultados deste experimento são apresentados na Tabela 3. A primeira coluna mostra o identificador da instância. As quatro colunas seguintes apresentam a média do tempo de cinco execuções da busca local Melhor Aprimorante implementada com as estruturas de dados de Estellon et al. (2008) e de Ribeiro et al. (2008), respectivamente. A razão entre o número de consultas de vizinhos e o número de trocas de carros efetivamente efetuados, chamado aqui de *Fator de Atualização* (FA), da busca local Melhor Aprimorante é exibido na coluna 4. Os mesmos dados são mostrados nas cinco últimas colunas para a busca local Primeiro Aprimorante. Em negrito, estão destacados os tempos de execução obtidos pela estrutura de dados mais rápida. Pode-se observar que a implementação da busca local Melhor Aprimorante com as estruturas de dados de Ribeiro et al. (2008) gerou média de tempo menor do que a implementação com as estruturas de dados de Estellon et al. (2008) em 21 das 28 instâncias testadas. Isto se deve ao fato de que, nesta busca local, o número de consultas de custo de vizinhos é muito maior que o número de atualizações, o que beneficia as estruturas de dados de Ribeiro et al. (2008). No caso da busca local Primeiro Aprimorante, a implementação com estruturas de dados de Ribeiro et al. (2008) gerou média de tempo menor do que a implementação com as estruturas de dados de Estellon et al. (2008) em apenas 15 das 28 instâncias testadas. Isto se deve ao fato de que, nesta busca local, o número de consultas de custo de vizinhos é bem menor do que o mesmo número na busca local Melhor Aprimorante, já que uma troca é efetivamente realizada toda vez que um vizinho aprimorante é encontrado. Neste caso, apesar da complexidade assintótica de pior caso para se calcular um vizinho com as estruturas de dados de Ribeiro et al. (2008) ser menor do que com as estruturas de dados de Estellon et al. (2008), o maior custo computacional de atualizar as estruturas de dados da primeira resulta em um custo computacional semelhante ao da segunda.

6.2 VFLS

Neste seção, observa-se o impacto das estruturas de dados de Estellon et al. (2008) e de Ribeiro et al. (2008) no desempenho de uma heurística VFLS baseada na busca local `troca`. A cada iteração, um vizinho que consiste da troca de dois carros da solução é gerado aleatoriamente. Se o custo do vizinho for melhor ou igual que o custo da solução corrente, esta última é atualizada. Vale salientar que, ao contrário dos algoritmos de busca local, são realizadas trocas de carros que resultam em soluções de mesmo custo que a solução corrente. Para cada instância, um valor alvo para o custo da solução da heurística foi fixada e a heurística foi programada para executar até que uma solução com custo igual ou melhor que o custo alvo fosse encontrada.

Duas implementações desta heurística foram testadas, usando as estruturas de dados de Estellon et al. (2008) e as de Ribeiro et al. (2008). Vale salientar, que independente da estrutura de dados utilizada, a heurística VFLS resultante realiza as mesmas decisões ao longo da sua execução. Entretanto, o tempo computacional de realizar cada operação varia com a estrutura de dados utilizada.

Os resultados deste experimento podem ser observados na Figura 12, onde são traçados os gráficos da distribuição empírica de probabilidade das duas implementações atingirem o alvo em função do tempo para as instâncias HPRC_23, HPRC_24, HPRC_29 e HPRC_48_2. Os gráficos para as outras instâncias apresentam o mesmo comportamento. Estes gráficos foram traçados utilizando-se a metodologia descrita em Aiex et al. (2002, 2007), conhecida como *Time-to-Target value plots*. Primeiramente, os tempos de 100 execuções de cada heurística são ordenados. Em seguida, associa-se ao i-ésimo menor tempo τ_i uma probabilidade $\phi_i = (i - \frac{1}{2})/200$. Por fim, traçam-se os pontos $b_i = (\tau_i, \phi_i)$, para $i = 1, \ldots, 200$. Pode-se observar que a probabilidade da implementação de VFLS com as estruturas de dados de Estellon et al. (2008) atingir o alvo é sempre maior que a da implementação de VFLS com as estruturas de dados de Ribeiro et al. (2008) em todas as instâncias apresentadas na Figura 12. O grande número de vizinhos de `troca` com o mesmo custo, e, consequentemente, o grande número de atualizações das estruturas de dados, explica o fato da implementação de VFLS com as estruturas de dados de Estellon et al. (2008) ser bem mais eficiente do que aquela com as estruturas de dados

Instâncias	Melhor Aprimorante					Primeiro Aprimorante				
	Estellon et al.	Desvio padrão	Ribeiro et al.	Desvio padrão	FA	Estellon et al.	Desvio padrão	Ribeiro et al.	Desvio padrão	FA
HPRC_022	**3,21**	0,46	3,28	0,37	0,07%	0,91	0,31	**0,73**	0,33	1,26%
HPRC_023	490,82	48,76	**459,89**	34,93	0,02%	11,70	1,12	**11,44**	1,33	2,84%
HPRC_024	641,19	25,16	**596,75**	21,35	0,01%	13,36	1,33	**13,08**	1,58	2,52%
HPRC_025	225,20	14,62	**219,25**	5,25	0,01%	5,96	0,69	**5,70**	0,77	1,78%
HPRC_028_1	20,29	1,63	**17,02**	1,76	0,15%	1,70	0,25	**1,65**	0,23	8,84%
HPRC_028_2	**0,58**	0,08	0,60	0,08	3,66%	**0,04**	0,03	0,05	0,03	12,58%
HPRC_029	36,60	1,84	**33,27**	1,95	0,04%	2,18	0,33	**2,16**	0,32	2,28%
HPRC_035_1	**0,48**	0,33	0,61	0,34	1,16%	**0,47**	0,11	0,51	0,11	16,35%
HPRC_035_3	**2,72**	0,26	3,13	1,47	0,27%	**0,63**	0,21	0,64	0,32	7,78%
HPRC_039	34,84	3,08	**32,65**	1,56	0,02%	2,45	0,27	**2,39**	0,26	0,38%
HPRC_048_1	60,33	3,90	**56,13**	5,23	0,06%	**2,40**	0,37	2,46	0,49	4,88%
HPRC_048_2	35,97	3,62	**35,18**	3,50	0,07%	**2,57**	0,19	2,63	0,24	4,23%
HPRC_064_1	245,93	17,02	**220,46**	10,75	0,03%	7,14	0,75	**6,86**	0,77	2,87%
HPRC_064_2	8,14	0,67	**7,72**	1,39	0,12%	1,21	0,11	**0,74**	0,11	2,76%
LPRC_022	17,42	2,19	**17,09**	2,37	0,07%	2,09	0,30	**1,93**	0,39	2,43%
LPRC_023	123,02	7,40	**120,90**	5,87	0,02%	**9,04**	1,23	9,51	1,97	0,65%
LPRC_024	512,91	15,16	**454,27**	18,53	0,01%	18,27	1,90	**16,79**	1,78	1,82%
LPRC_025	1,209,53	25,59	**1,110,16**	22,97	0,01%	**18,05**	1,85	19,11	2,84	4,31%
LPRC_028_1	34,89	2,20	**31,97**	1,39	0,15%	**1,87**	0,26	1,88	0,39	15,48%
LPRC_028_2	**0,52**	0,30	0,58	0,30	4,24%	**0,45**	0,31	0,48	0,30	29,71%
LPRC_029	**2,84**	0,16	2,96	0,42	0,03%	1,63	0,27	**1,41**	0,44	0,79%
LPRC_035_1	0,72	0,38	**0,48**	0,27	1,17%	**0,59**	0,26	0,62	0,26	19,00%
LPRC_035_3	**1,74**	0,26	1,82	0,28	0,27%	**0,48**	0,31	0,62	0,33	6,61%
LPRC_039	244,16	7,86	**176,09**	4,53	0,02%	19,20	3,00	**14,29**	2,11	2,17%
LPRC_048_1	124,24	12,61	**118,37**	12,60	0,06%	12,67	0,72	**11,66**	1,97	4,90%
LPRC_048_2	68,10	3,85	**66,42**	4,33	0,07%	**4,76**	0,44	4,76	0,42	4,31%
LPRC_064_1	51,89	1,56	**47,98**	1,34	0,03%	3,95	1,61	**3,36**	0,81	1,91%
LPRC_064_2	0,77	0,29	**0,87**	0,30	0,10%	**0,70**	0,31	0,78	0,24	1,04%

Tabela 3. Comparação do tempo de execução das busca locais Primeiro Aprimorante e Melhor Aprimorante implementadas com as estruturas de dados de Estellon et al. (2008) e de Ribeiro et al. (2008).

de Ribeiro et al. (2008), pois a primeira pode ser atualizada em $O(m)$ após a troca de dois carros, enquanto a atualização da segunda custa $O(n \cdot m)$.

7. Considerações Finais

Este capítulo apresentou o Problema do Sequenciamento de Carros, que consiste em determinar a ordem em que carros devem ser produzidos, de forma a minimizar o número de violações às restrições de capacidade da linha de montagem. Ademais, foram descritas as principais heurísticas e estruturas de dados existentes na literatura para o problema. Realizou-se uma série de experimentos computacionais baseados em dados reais que permitiu elaborar um estudo comparativo entre os algoritmos revisitados e suas estruturas de dados, identificando os pontos fortes e fracos de cada um.

Além da revisão da literatura sobre o PSC, percorrendo as heurísticas mais bem-sucedidas para o problema até então, este capítulo teve como objetivo principal mostrar que a escolha das estruturas de dados é um aspecto importante na implementação de heurísticas pois têm um papel crucial em seu desempenho. As estruturas de dados devem ser eficientes do ponto de vista das operações que deverão ser realizadas sobre elas, sobretudo as operações executadas com maior frequência.

Referências

Aiex, R.M.; Resende, M.G.C. & Ribeiro, C.C., Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8(3):343–373, 2002.

Aiex, R.M.; Resende, M.G.C. & Ribeiro, C.C., TTTPLOTS: A Perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, 2007.

Cheng, J.; Lu, Y.; Puskorius, G.; Bergeon, S. & Xiao, J., Vehicle sequencing based on evolutionary computation. In: *Proceedings of the Congress on Evolutionary Computation*. Piscataway, USA: IEEE Press, v. 3, p. 1207–1215, 1999.

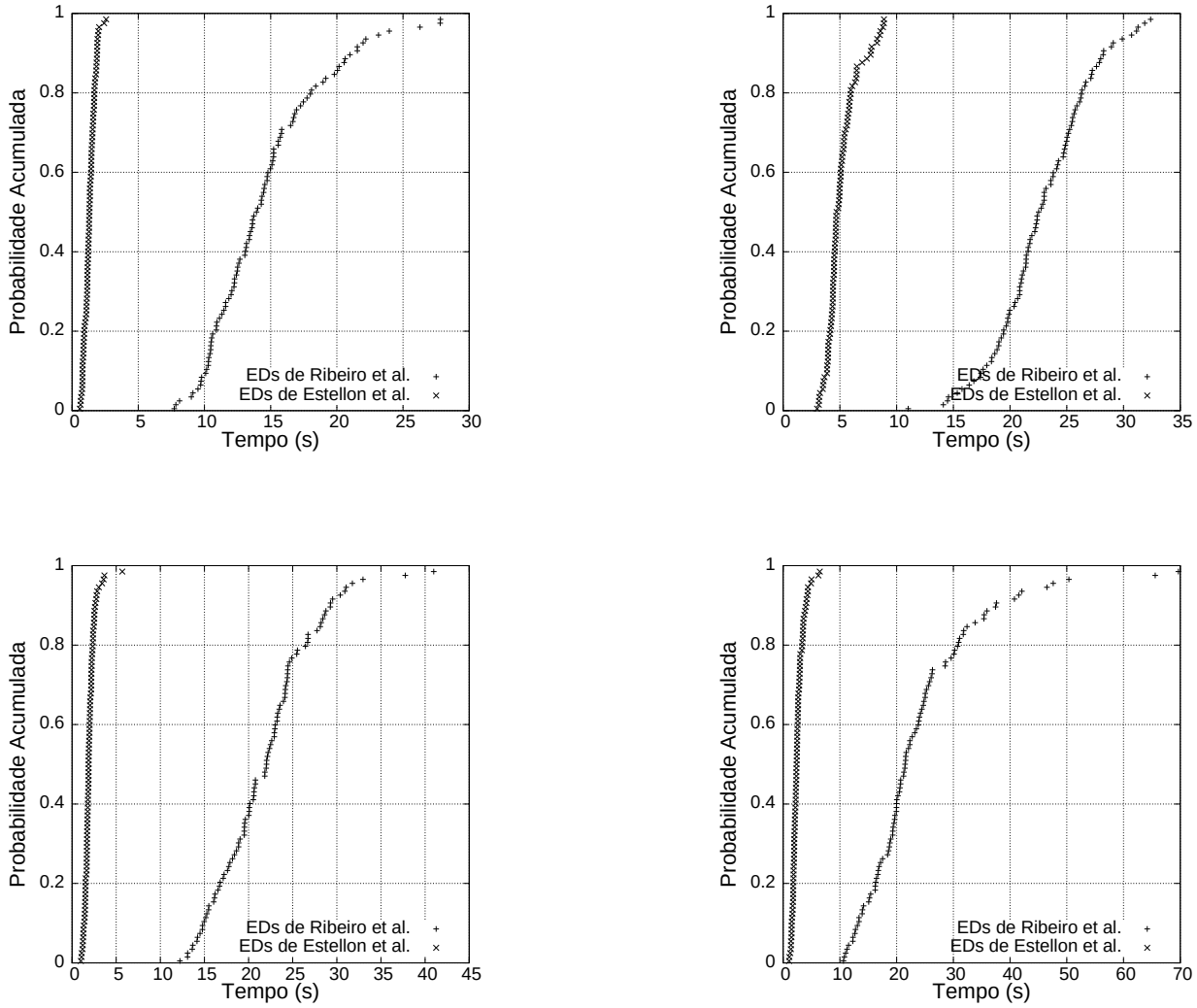

Figura 12. Gráficos da distribuição empírica de probabilidade das duas implementações atingirem o alvo em função do tempo para as instâncias HPRC_23 (a), HPRC_24 (b), HPRC_29 (c) e HPRC_48_2 (d).

Dincbas, M.; Simonis, M. & van Hentenryck, P., Solving the car sequencing problem in constraint logic programming. In: Kodratoff, Y. (Ed.), *Proceedings of the 8th European Conference on Artificial Intelligence*. London, UK: Pitman, p. 290–295, 1988.

Drexl, A.; Kimms, A. & Matthießen, L., Algorithms for the car sequencing and the level scheduling problem. *Journal of Scheduling*, 9(2):153–176, 2006.

Estellon, B.; Gardi, F. & Nouioua, K., Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research*, 191(3):928–944, 2008.

Gagné, C.; Gravel, M. & Price, W.L., Solving real car sequencing problems with ant colony optimization. *European Journal of Operational Research*, 174(3):1427–1448, 2006.

Gavranovic, H., Local search and suffix tree for car sequencing problem with colors. *European Journal of Operational Research*, 191(3):972–980, 2008.

Gent, I.P., *Two Results on Car Sequencing Problems*. Technical Report APES-02-1998, Department of Computer Science, University of Strathclyde, Glasgow, UK, 1998.

Gottlieb, J.; Puchta, M. & Solnon, C., A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In: Cagnoni, S.; Johnson, C.G.; Marchiori, E.; Corne, D.W. & Gottlieb, J. (Eds.), *Applications of Evolutionary Computing*. Heidelberg, Germany: Springer-Verlag, v. 2611 de *Lecture Notes in Computer Science*, p. 246–257, 2003.

Kis, T., On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335, 2004.

Parello, B.D.; Kabat, W.C. & Wos, L., Job-shop scheduling using automated reasoning: a case study of the car sequencing problem. *Journal of the Automated Reasoning*, 2(1):1–42, 1986.

Règin, J.C. & Puget, J.F., A filtering algorithm for global sequencing constraints. In: Smolka, G. (Ed.), *Principles and Pratice of Constraint Programming*. Heidelberg, Germany: Springer-Verlag, v. 1330 de *Lecture Notes in Computer Science*, p. 32–46, 1997.

Ribeiro, C.C.; Aloise, D.; Noronha, T.F.; Rocha, C. & Urrutia, S., An efficient implementation of a VNS/ILS heuristic for a real-life car sequencing problem. *European Journal of Operational Research*, 191(3):596–611, 2008.

ROADEF, , Roadef challenge 2005. http://challenge.roadef.org/2005/en/, 2005. Visitado em: 07/11/2012.

Smith, B.M., Succeed-first or fail-first: a case study in variable and value ordering. Report 96.26, University of Leeds, Division of Artificial Intelligence, Leeds, UK, 1996.

Solnon, C., Solving permutation constraint satisfaction problems with artificial ants. In: Horn, W. (Ed.), *Proceedings of the 14th European Conference on Artificial Intelligence*. Amsterdam, Netherlands: IOS Press, p. 118–122, 2000.

Solnon, C., Combining two pheromone structures for solving the car sequencing problem with ant colony optimization. *European Journal of Operational Research*, 191(3):1043–1055, 2008.

van Hentenryck, P.; Simonis, H. & Dincbas, M., Constraint satisfation using constraint logic programming. *Artificial Intelligence*, 58:113–159, 1992.

Warwick, T. & Tsang, E., Tacking car sequencing problems using a genetic algorithm. *Evolutionary Computation*, 3(3):267–298, 1995.

Notas Biográficas

Daniel Brasil é graduado em Ciência da Computação (Universidade Federal de Minas Gerais, 2011). Atualmente cursa o Mestrado do Programa de Pós-Graduação do Departamento de Ciência da Computação da UFMG.

Thiago Ferreira de Noronha é graduado em Ciência da Computação (Universidade Federal do Rio Grande do Norte, 2001), mestre e doutor em Informática (Pontifícia Universidade Católica do Rio de Janeiro, 2004 e 2008, respectivamente). Trabalhou na Google Brasil como engenheiro de software de 2008 a 2009. Atualmente é professor pesquisador do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais.

Caroline Rocha é graduada em Ciência da Computação (Universidade Federal do Rio Grande do Norte, 2002), mestre em Computação (Universidade Federal Fluminense, 2005) e doutora em Informática (Université de Montréal, Canadá). Atualmente é professora adjunta da Escola de Ciências e Tecnologia da UFRN.