LIII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2021

O problema de sequenciamento com restrições de cadência

Lara Pontes, Carlos Neves, Anand Subramanian
Universidade Federal da Paraíba – Centro de Informática
Rua dos Escoteiros, s/n – Mangabeira, João Pessoa – PB, 58055-000
laradicp@gmail.com, carlosneves@eng.ci.ufpb.br, anand@ci.ufpb.br

Maria Battarra
University of Bath – School of Management
Bath BA2 7AY, United Kingdom
mb2182@bath.ac.uk

RESUMO

Este trabalho é fruto da colaboração com uma grande empresa multinacional de automóveis, que impõe dois tipos de cadência no sequenciamento de suas operações de montagem, para que tarefas com atributos mais exigentes não sejam alocadas próximas umas das outras. O objetivo é sequenciar o número máximo de tarefas consecutivas, respeitando as restrições mencionadas. O problema de sequenciamento é aqui formalizado e provado ser fortemente NP-completo. Uma formulação de programação inteira é proposta, bem como uma formulação que encontra uma sequência viável com dado número de tarefas, se assim existir. Esta última é utilizada em algoritmos de busca binários e iterativos, aprimorados pelos limites dual combinatório e primal heurístico. Resultados computacionais revelaram melhor performance dos seguintes algoritmos: busca binária com limites dual trivial ou combinatório e primal heurístico; e busca iterativa com limite dual combinatório. As instâncias que refletem as demandas da empresa são resolvidas na otimalidade em segundos.

PALAVRAS CHAVE. Sequenciamento, Cadências, Programação Inteira, Algoritmos Exatos.

OC – Otimização Combinatória, AD&GP – PO na Administração e Gestão da Produção

ABSTRACT

This work is a result of a collaboration with a large multinational automotive company, which imposes two types of cadences on the sequence of assembly operations, so the jobs with more demanding features are not scheduled in close proximity to each other. The goal is to sequence the maximum number of consecutive jobs, without violating the aforementioned constraints. We formalize this scheduling problem, and prove that it is strongly NP-complete. An integer programming formulation is proposed, as well as a formulation that finds a feasible schedule for a given number of jobs, if any exists. The latter is embedded within binary and iterative search algorithms. Both algorithms are further enhanced by combinatorial and heuristic bounds. Extensive computational results showed that the binary search with heuristic lower bound and the iterative search with combinatorial upper bound performed best. Instances reflecting the company's needs are solved to optimality within seconds.

KEYWORDS. Scheduling. Cadences. Integer Programming. Exact Algorithms.

CO – Combinatorial Optimization, AD&GP – OR in Administration and Production Management

1. Introdução

Linhas de montagem automobilística têm sido amplamente estudadas na literatura. Essa indústria persiste com sua histórica dependência às operações manuais, como reportado por Coffey e Thornley [2006]. Dessa forma, a motivação e o bem-estar dos operários são de suma importância para a produtividade do setor, cuja carga trabalhista deve ser justa e factível.

Esta pesquisa foi inspirada a partir da colaboração com uma grande fábrica nacional de automóveis, que demanda diferentes níveis de mão de obra a depender do modelo produzido. Há, portanto, restrições de cadência associadas aos modelos, responsáveis por não sobrecarregar as estações de trabalho durante a linha de montagem. A empresa descreve conjuntos de políticas para realizar a alocação dos automóveis, as quais asseguram tempo suficiente entre as operações mais complexas. Determinado modelo de carro com certo atributo, por exemplo, precisa estar afastado, por pelo menos K posições, de qualquer modelo com o mesmo atributo. Outra possível cadência estabelece um número máximo de veículos consecutivos com dado atributo na sequência.

Apesar de se encontrar em uma área muito explorada por pesquisadores, esse problema não é comum na literatura. Han e Lin [1992] impõem restrições de separação entre vértices conectados por arestas no grafo original, similarmente às cadências de afastamento de tarefas na sequência. No entanto, tais restrições precisam ser satisfeitas por pares de vértices, enquanto as cadências são definidas para grupos de tarefas associadas a um número K. Existe, também, uma classe de problemas cuja execução das tarefas é limitada por um dado atraso [Dell'Amico, 1996; Potts e Whitehead, 2007; Zhang e van de Velde, 2010]. Sheen e Liao [2007] proporcionam uma boa revisão de literatura em problemas de *scheduling* com máquina única e restrições de atraso. Estas apresentam a mesma característica de se aplicarem somente a pares, não a conjuntos. No que concerne às cadências de consecutividade, Van den Bergh et al. [2013]; Ernst et al. [2004] aproximam-se dessas apresentando restrições que limitam o número de dias de trabalho consecutivos.

Atualmente, a empresa gera a sequência de forma manual, despendendo tempo e esforço de funcionários, para uma alocação pouco eficaz. Conforme provado no fim da Seção 2 deste artigo, o problema é enquadrado na classe de problemas fortemente NP-completos. Sendo assim, a colaboração visa a desenvolver um algoritmo que o resolva automática e eficientemente.

O restante deste trabalho é organizado como se segue. A Seção 2 define o problema formalmente, e a Seção 3 demonstra sua formulação matemática. A Seção 4 apresenta um limite superior válido e os algoritmos exatos utilizados na versão de viabilidade. A Seção 5 propõe um método heurístico de resolução do problema. A Seção 6 descreve as instâncias usadas e compara estratégias exatas, partindo inclusive de resultados heurísticos. Por fim, a Seção 7 conclui o trabalho.

2. Definição do problema

As cadências se enquadram em um dos dois seguintes tipos de política. **Política 1:** dado um subconjunto de tarefas S, não mais que $C(S)$ tarefas do conjunto S podem ser alocadas consecutivamente em qualquer posição da sequência. **Política 2:** dado um subconjunto de tarefas T, todas as tarefas em T devem ter, no mínimo, $A(T)$ tarefas não pertencentes a T afastando-as umas das outras na sequência.

Consideram-se, então, J como o conjunto de tarefas destinadas à alocação em uma estação e P como o conjunto de posições da sequência. Cada tarefa j pode ser atribuída a, no máximo, uma posição da sequência, bem como cada posição da sequência só pode hospedar, no máximo, uma tarefa. Posições vazias só são permitidas no final da sequência, isto é, proíbem-se "buracos" na linha de produção. São dados o conjunto $\mathcal{C}$ para a Política 1, em que cada $S \in \mathcal{C}$ é um subconjunto de tarefas associadas a um inteiro $C(S)$; e o conjunto $\mathcal{A}$ para a Política 2, em que cada $T \in \mathcal{A}$ é um

subconjunto de tarefas associadas a um inteiro $A(T)$. A tarefa $j \in J$ pode pertencer a nenhum, a um ou a vários conjuntos S e T.

Na versão de viabilidade do problema de sequenciamento com restrições de cadência (PSC), a ser explorada pelos algoritmos da Seção 4, o objetivo é determinar se existe alguma sequência com todas as posições atribuídas a exatamente uma tarefa. A versão de otimização, por sua vez, busca maximizar o número de tarefas alocadas.

Proposição 1. *O PSC é fortemente NP-completo.*

Demonstração. Para a redução, usa-se o *Exact Cover by 3-Sets problem* (X3C) [Karp, 1972]: dado um conjunto X com $X = \{1, \ldots, 3n\}$ e uma coleção $C = \{C_1, \ldots, C_m\}$ de subconjuntos de X com 3 elementos, o objetivo é encontrar, se possível, uma subcoleção $C' \subseteq C$ tal que cada elemento em X está contido em exatamente um subconjunto de C'. A instância correspondente do PSC é definida como se segue:

1. $P = \{1, \ldots, n\}, J = \{1, \ldots, m\}$;
2. $T_i = \left\{ j \in \{1, \ldots, m\} \;\middle|\; x_i \in C_j \right\}, A(T_i) = n + i, \forall i \in \{1, \ldots, 3n\}$;
3. $\mathcal{A} = \{T_1, \ldots, T_{3n}\}, \mathcal{C} = \emptyset$.

O tamanho da instância do X3C é

$$3m \log(3n) + \log n.$$

Tanto as tarefas quanto as posições da instância do PSC podem ser representadas por números binários. Nessa codificação, os tamanhos de P e J são $n \log n$ e $m \log m$, respectivamente. Além disso, $|\mathcal{A}| = 3n$, e cada elemento em X pode estar contido em, no máximo, m subconjuntos. Logo, o tamanho de $\mathcal{A}$ é limitado superiormente por $3mn \log m$. Os valores das cadências variam de $n + 1$ a $n + 3n$, demandando, portanto, $3n \log(4n)$. Finalmente, o tamanho da instância do PSC é limitado superiormente por

$$n \log n + m \log m + 3mn \log m + 3n \log(4n).$$

Pode-se assumir, sem perda de generalidade, que $m \geq n$. Portanto, os tamanhos das instâncias se relacionam polinomialmente, e a transformação é polinomial. Além disso, o maior parâmetro numérico da instância do PSC é $\max_{i=1,\ldots,3n} A(T_i) = 4n$, que também está relacionado ao seu tamanho polinomialmente.

Suponha, agora, que nenhuma solução para uma determinada instância do X3C exista. Considere $C' = \{C_1, \ldots, C_n\} \subseteq C$ uma subcoleção arbitrária de C, sendo $J' = \{1, \ldots, n\}$ o conjunto de tarefas associadas a C'. Por definição, pelo menos um elemento $x_i \in X$ está contido em mais de um subconjunto em C'. Suponha, ainda, que $C_1, \ldots, C_k$ são os subconjuntos em C' que contêm tal elemento. Devido a 2 e 3, sabe-se que as tarefas $1, \ldots, k$ estão contidas em T_i, e que $A(T_i) > n$. Já que as tarefas $1, \ldots, k$ precisam estar separadas por pelo menos n posições, e que $|P| = n$, é impossível construir uma sequência com as tarefas de J' de modo a preencher todas as posições de P.

Suponha, por outro lado, que a subcoleção C' é uma solução válida para a instância do X3C, e que uma solução para a versão de viabilidade do PSC foi obtida iterando-se por cada $C_j \in C'$ e atribuindo a tarefa j a uma posição diferente de P. Suponha, por contradição, que existe um par de tarefas j_1 e j_2 na sequência tal que $j_1, j_2 \in T_i$ para algum i. Isso significa que existe um elemento $x_i \in X$ contido em C_{j_1} e C_{j_2}. Logo, C' não é uma solução para a instância do X3C. Portanto, a sequência construída é viável.

Uma lógica similar pode ser aplicada na ordem reversa (PSC $\rightarrow$ X3C). $\square$

3. Formulação matemática

Diferentes tarefas podem ter os mesmos atributos, logo as mesmas cadências, o que as torna intercambiáveis na alocação. A fim de garantir uma melhor formulação, define-se o conceito de famílias: agrupamentos dessas tarefas semelhantes. $\mathcal{F}$ é uma partição de J em conjuntos $F \subseteq J$ de tarefas que pertencem à mesma família. I é o conjunto de índices de $\mathcal{F}$, com os subconjuntos $I_S \subseteq I$ e $I_T \subseteq I$ para cada $S \in \mathcal{C}$ e $T \in \mathcal{A}$. I_S contém os índices das famílias com a cadência $C(S)$, e I_T contém os índices das famílias com a cadência $A(T)$. Considerando tais definições e utilizando a variável binária x_{ip}, que assume valor 1 quando alguma tarefa da família de índice $i \in I$ está na posição $p \in P$, e 0 caso contrário, propõe-se a seguinte formulação para a versão de otimização.

$$\max \sum_{i \in I} \sum_{p \in P} x_{ip} \tag{1}$$

Sujeito a:

$$\sum_{p \in P} x_{ip} \leq |F_i| \qquad i \in I \tag{2}$$

$$\sum_{i \in I} x_{ip} \leq 1 \qquad p \in P \tag{3}$$

$$\sum_{i \in I_S} \sum_{p'=p}^{k} x_{ip'} \leq C(S) \qquad k = \min\{p + C(S), |P|\}, p \in P, S \in \mathcal{C} \tag{4}$$

$$\sum_{i \in I_T} \sum_{p'=p}^{k} x_{ip'} \leq 1 \qquad k = \min\{p + A(T), |P|\}, p \in P, T \in \mathcal{A} \tag{5}$$

$$\sum_{i \in I} x_{ip+1} \leq \sum_{i \in I} x_{ip} \qquad p \in P, p \neq |P| \tag{6}$$

$$x_{ip} \in \{0, 1\} \qquad i \in I, p \in P. \tag{7}$$

A função objetivo maximiza o número de tarefas alocadas. As restrições (2) e (3) asseguram que a demanda de cada família não é ultrapassada, e que cada posição hospeda, no máximo, uma tarefa. As restrições (4) e (5) satisfazem as políticas 1 e 2. As restrições (6) forçam posições vazias para o final da sequência. Por fim, as restrições (7) definem a natureza das variáveis x_{ip}.

Para a versão de viabilidade, não há função objetivo, tendo em vista que não ocorrerá otimização. As restrições (3) são substituídas pelas restrições (9), e as restrições (6) podem ser dispensadas, já que as restrições (9) impõem a ocupação de todas as posições, impossibilitando posições vazias em qualquer solução viável por consequência.

$$\max 0 \tag{8}$$

Sujeito a:

$$\sum_{i \in I} x_{ip} = 1 \qquad p \in P \tag{9}$$

(2), (4), (5), (7).

4. Algoritmos exatos

Nesta seção, apresentam-se dois algoritmos exatos de verificação de viabilidade, são esses as buscas binária e iterativa, bem como um limite superior válido, utilizado em ambas as estratégias.

4.1. Limite superior válido

Para todo $U_i \in \mathcal{U} = \mathcal{A} \cup \mathcal{C} = \{U_1, \ldots, U_k\}$, considere $V_i \subseteq U_i$ como o subconjunto de tarefas associadas que não estão alocadas na solução ótima. Embora não seja possível determinar os elementos de V_i *a priori*, é possível calcular um limite inferior Y para $\left|\bigcup_{i=1}^{k} V_i\right|$, e com isso, obter o limite superior válido $|J| - Y$. Para isso, utilizou-se o conceito de "sobra", definida como o número de tarefas em um conjunto (U) que não cabem em nenhuma solução viável devido à Política 1 (se $U_i \in \mathcal{C}$) ou à Política 2 (se $U_i \in \mathcal{A}$). Dado um conjunto $U_i \in \mathcal{U}$, sua sobra é obtida pela equação (10):

$$sobra(U_i) = \begin{cases} \max\left\{0, |U_i| - \left\lceil \dfrac{C(U_i) \times |P|}{C(U_i)+1} \right\rceil \right\} & \text{se } U_i \in \mathcal{C}, \\ \max\left\{0, |U_i| - \left\lceil \dfrac{|P|}{A(U_i)+1} \right\rceil \right\} & \text{se } U_i \in \mathcal{A}. \end{cases} \tag{10}$$

Baseando-se nisso, uma maneira de estimar Y é proposta a partir do conjunto de variáveis

$$Z_i = \begin{cases} sobra(U_i) & \text{se } i = 1, \\ \max\{0, sobra(U_i) - \sum_{j=1}^{i-1} \min\{Z_j, |U_i \cap U_j|\}\} & \text{caso contrário.} \end{cases}$$

Proposição 2. $Y = \sum_{i=1}^{k} Z_i \leq \left|\bigcup_{i=1}^{k} V_i\right|$.

Demonstração. Utiliza-se indução no número de conjuntos em $\mathcal{U}$. Para o caso base, $\mathcal{U} = \{U_1\}$, e o valor retornado pela fórmula é $sobra(U_1) = |V_1|$. Para a hipótese de indução, considere que existem conjuntos $W_1, \ldots, W_k$, disjuntos dois a dois, tais que $\forall i = 1, \ldots, k$, $W_i \subseteq V_i$, $Z_i = |W_i|$. Se a propriedade persistir ao adicionar um conjunto U_{k+1} a $\mathcal{U}$, a soma das variáveis Z_i será

$$\sum_{i=1}^{k+1} Z_i = \left|\bigcup_{i=1}^{k+1} W_i\right| \leq \left|\bigcup_{i=1}^{k+1} V_i\right|.$$

Se $Z_{k+1} = 0$, segue que $W_{k+1} = \emptyset \subseteq V_{k+1}$. Por outro lado, se $Z_{k+1} > 0$, então

$$Z_{k+1} = sobra(U_{k+1}) - \sum_{i=1}^{k} \min\{|U_i \cap U_{k+1}|, Z_i\} \leq |V_{k+1}| - \sum_{i=1}^{k} \min\{|U_i \cap U_{k+1}|, Z_i\}.$$

Posto isso, considere o conjunto

$$\Omega = V_{k+1} \setminus \left(\bigcup_{i=1}^{k} W_i\right),$$

cuja cardinalidade é

$$|\Omega| = |V_{k+1}| - \sum_{i=1}^{k} |V_{k+1} \cap W_i|.$$

Já que $Z_i = |W_i| \geq |V_{k+1} \cap W_i|$ e $|U_i \cap U_{k+1}| \geq |V_{k+1} \cap W_i|$ para todo $i = 1, \ldots, k$, tem-se que $Z_{k+1} \leq |\Omega|$. Portanto, há pelo menos um conjunto $W_{k+1} \subseteq \Omega$ tal que $\forall i = 1, \ldots, k$, $W_{k+1} \cap W_i = \emptyset$, $|W_{k+1}| = Z_{k+1}$. □

4.2. Busca binária

A busca binária estabelece dois limites, um máximo e um mínimo. O máximo pode ser trivial, igual ao número de tarefas da instância, ou o apresentado na Subseção 4.1. O mínimo, por sua vez, pode ser trivial, uma tarefa, ou a solução primal retornada pela heurística descrita na Seção 5. A cada iteração, é calculada a média entre os dois limites (arredondada para cima), e se verifica a viabilidade de alocar tal quantidade de tarefas na sequência. Se sim, esse valor se torna o novo limite mínimo. Caso contrário, o limite máximo assume o valor subtraído de um. O processo continua até que os dois limites sejam iguais ou até alcançar o tempo limite.

4.3. Busca iterativa

O segundo algoritmo exato utiliza uma abordagem iterativa. Dado um limite mínimo ou um limite máximo, ambos semelhantes aos utilizados pela busca binária, é feita, iterativamente, a verificação de viabilidade de mais uma ou de menos uma tarefa alocada na sequência. Para o incremento partindo de um limite inferior, o critério de parada é a comprovação de inviabilidade com o limite inferior de posições acrescido de um. Em contraste, no decremento, que parte do limite superior, o critério é o encontro de uma solução viável com o limite superior de posições vigente, atualizado a partir da comprovação de inviabilidade da alocação com o limite superior precedente. Pode haver interrupção nas buscas antes de concluídas caso o tempo limite seja alcançado.

5. Método heurístico

O algoritmo proposto é baseado na meta-heurística *Iterated Local Search* (ILS) [Ramalhinho Lourenço et al., 2019]. Primeiramente, a solução inicial s é obtida por meio de um processo construtivo aleatório, resumido à escolha aleatória, dentre as opções viáveis, da próxima tarefa a ser adicionada à sequência. A alocação continua até que, após atualizada, a lista de candidatos viáveis esteja vazia. Em seguida, inicia-se a busca local iterada com uma perturbação, que alterna entre a remoção de n tarefas da solução e a troca da posição de duas tarefas na sequência (*swap*) até n vezes. Em ambos os casos, $n = \min\{tamanho(s)/2, |J|/11\}$. O denominador de $|J|/11$ foi obtido a partir da avaliação do desempenho da heurística nas instâncias reais da empresa. Por fim, é realizada a inserção gulosa de tarefas na solução. Começando por uma posição aleatória da sequência, insere-se o maior número de tarefas em cada posição da solução. O número máximo de iterações sem melhora na busca local é dez, e o algoritmo é *multi-start*, reiniciando $|J|/2$ vezes.

6. Resultados computacionais

Todas as instâncias foram resolvidas usando apenas um núcleo de um processador Intel Xeon E5-2650 com 128 GB de RAM e sistema operacional Ubuntu 16.04. Estipulou-se tempo limite de 600 segundos por instância. As formulações propostas foram implementadas na linguagem C++ e o resolvedor de programação linear inteira adotado foi o CPLEX versão 12.9.

Nesta seção, são apresentados o processo de obtenção das instâncias e os resultados computacionais das combinações dos métodos exatos com limites superiores e inferiores variados.

6.1. Instâncias

As instâncias, divididas em quatro grupos, estão hospedadas em `https://github.com/laradicp/scheduling-with-cadences/tree/main/instances`. O grupo A equivale às 193 instâncias originais coletadas na empresa. Esse grupo foi modificado por meio da adição de uma nova cadência, gerando os demais grupos de acordo com a quantidade máxima permitida de tarefas por família. Os grupos B, C e D permitem até 8, 4 e 2 tarefas por família, respectivamente, desde que ainda haja combinações distintas de cadência para gerar novas famílias. Por forçar mais famílias em cada instância, o grupo D se mostra o mais desafiador.

6.2. Versões

Cada versão dos métodos propostos foi gerada de acordo com as possíveis combinações de limites máximos e mínimos. Para os limites máximos, foram considerados o descrito na Seção 4.1 e o trivial, isto é, $|P| = |J|$. Os limites mínimos compreenderam a solução primal retornada pela heurística e $|P| = 1$ como trivial. A busca binária assumiu esses limites dois a dois e sempre retornou soluções primais, enquanto a iterativa recebeu ou um limite máximo, retornando uma solução dual, ou um limite mínimo, retornando uma solução primal.

6.3. Resultados computacionais

Para fins de sumarização, as versões receberam códigos relativos às possíveis combinações de busca, limite máximo e limite mínimo. A busca binária recebe o código "b", e a iterativa, "i". O *upper bound* gerado pelo algoritmo da Seção 4.1 recebe o rótulo "au", enquanto o trivial recebe "tu". Por fim, o *lower bound* da heurística equivale a "hl", e o trivial, a "tl". O rótulo "puro" se refere ao modelo de otimização puro, desvinculado de qualquer busca descrita.

Nas tabelas, cada coluna representa o seguinte: #primal é a quantidade de instâncias que retornou uma solução primal; #ótimo é a quantidade de instâncias que achou o valor ótimo, mesmo que não comprovado; output indica se o método retorna um dual ou um primal; gap é o *gap* calculado em relação ao melhor dual se o *output* for primal, e em relação ao melhor primal se o *output* for dual; T é o tempo médio, em segundos, de todas as instâncias; e T_p é o tempo médio, em segundos, das execuções em que o ótimo foi provado.

Tabela 1: Conjunto de instâncias A

	#primal	#ótimo	output	gap (%)	T (s)	T_p (s)
bhlau	193	193	primal	0	1.78	1.78
bhltu	193	193	primal	0	1.79	1.79
btlau	193	193	primal	0	0.57	0.57
btltu	193	193	primal	0	0.59	0.59
ihl	193	193	primal	0	1.96	1.96
itl	193	193	primal	0	4.45	4.45
iau	193	193	dual	0	0.26	0.26
itu	193	193	dual	0	0.43	0.43
puro	193	191	dual	0.01	10.60	4.43

Tabela 2: Conjunto de instâncias B

	#primal	#ótimo	output	gap (%)	T (s)	T_p (s)
bhlau	193	193	primal	0	9.71	9.71
bhltu	193	193	primal	0	10.12	10.12
btlau	193	193	primal	0	24.60	24.60
btltu	193	192	primal	< 0.01	24.96	21.96
ihl	193	192	primal	< 0.01	16.11	13.07
itl	193	185	primal	0.23	134.75	114.62
iau	193	193	dual	0	7.18	7.18
itu	193	193	dual	0	9.70	9.70
puro	193	175	dual	0.16	69.36	14.76

Tabela 3: Conjunto de instâncias C

	#primal	#ótimo	output	gap (%)	T (s)	T_p (s)
bhlau	193	185	primal	0.03	88.50	66.36
bhltu	193	184	primal	0.06	90.57	65.64
btlau	193	173	primal	0.09	191.06	143.73
btltu	193	173	primal	0.13	188.18	140.52
ihl	193	171	primal	0.15	128.75	68.09
itl	193	102	primal	5.30	356.62	139.28
iau	192	192	dual	< 0.01	60.02	57.21
itu	192	192	dual	< 0.01	73.44	70.69
puro	192	153	dual	0.30	193.31	86.91

Tabela 4: Conjunto de instâncias D

	#primal	#ótimo	output	gap (%)	T (s)	T_p (s)
bhlau	193	137	primal	0.31	310.82	192.50
bhltu	193	136	primal	0.36	307.44	184.70
btlau	193	107	primal	0.63	388.84	218.80
btltu	193	114	primal	0.87	379.22	225.96
ihl	193	103	primal	0.58	398.27	221.75
itl	193	47	primal	6.61	514.15	246.59
iau	158	159	dual	0.14	240.37	163.41
itu	156	159	dual	0.14	280.76	212.44
puro	138	30	dual	1.38	559.26	335.59

Analisando as tabelas, pode-se observar que, nos conjuntos A e B, ambas as buscas alcançaram o valor ótimo de todas as instâncias em pelo menos uma de suas versões. Já nos conjuntos C e D, a busca binária mais eficiente conta com limite máximo trivial ou combinatório, equivalentes em desempenho, e limite mínimo heurístico. A busca iterativa com limites máximos, por sua vez, performou melhor no quesito otimalidade, apesar da lacuna em soluções primais. O modelo puro teve a pior performance em todos os grupos.

A heurística se mostrou de suma importância para a obtenção de soluções primais de qualidade nas instâncias difíceis, e a presença de um processo adicional não impactou negativamente o tempo em relação às demais versões.

Vale salientar que as linhas de produção registradas pela empresa, cujas demandas geraram o conjunto A, violaram as políticas 1 ou 2 uma média de 16,08 vezes por instância, além do tempo despendido para gerar a solução de forma manual. Tais violações sobrecarregam o processo de fabricação e, por conseguinte, os trabalhadores indevidamente, visto que esta colaboração foi capaz de encontrar o ótimo de todas as instâncias, em poucos segundos e de forma automatizada.

7. Considerações finais

Este trabalho introduziu o problema de sequenciamento com restrições de cadência, que foi provado ser fortemente NP-completo em sua versão de decisão. Para resolvê-lo, foi proposto um modelo matemático, bem como algoritmos exatos e uma abordagem heurística. Foi desenvolvido também um procedimento para computar um limite dual válido a ser utilizado, juntamente ao limite primal fornecido pela heurística, nos algoritmos exatos baseados em busca binária e busca iterativa.

Como se observa na seção anterior, os limites máximos trivial e combinatório fornecem resultados semelhantes na busca binária com limite mínimo heurístico, sendo essas as versões mais eficientes de busca binária. Em contrapartida, a busca iterativa partindo de limite máximo foi responsável por encontrar o maior número de soluções ótimas em todos os grupos. A desvantagem desse método, no entanto, é a sua confiabilidade reduzida em retornar soluções primais. Quanto à escolha do limite máximo, o provido pelo algoritmo forneceu mais soluções primais nas instâncias do grupo D e causou uma diferença relevante no tempo em comparação ao trivial. Em síntese, o problema foi resolvido com completa eficácia para as instâncias reais da empresa, e as estratégias destacadas tiveram desempenho satisfatório em todos os grupos.

Referências

Coffey, D. e Thornley, C. (2006). Automation, motivation and lean production reconsidered. *Assembly Automation*, 26: 98–103.

Dell'Amico, M. (1996). Shop problems with two machines and time lags. *Operations Research*, 44:777–787.

Ernst, A., Jiang, H., Krishnamoorthy, M., e Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27.

Han, C.-C. e Lin, K.-J. (1992). Scheduling real-time computations with separation constraints. *Information Processing Letters*, 42(2):61–66.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, p. 85–103. Springer.

Potts, C. e Whitehead, J. (2007). Heuristics for a coupled-operation scheduling problem. *Journal of the Operational Research Society*, 58:1375–1388.

Ramalhinho Lourenço, H., Martin, O., e Stützle, T. (2019). Iterated Local Search: Framework and Applications. In Gendreau, M. e Potvin, J. Y., editors, *Handbook of Metaheuristics*, p. 129–168. Springer International Publishing.

Sheen, G.-J. e Liao, L. (2007). A branch and bound algorithm for the one-machine scheduling problem with minimum and maximum time lags. *European Journal of Operational Research*, 181(1):102–116.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., e De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385.

Zhang, X. e van de Velde, S. (2010). Polynomial-time approximation schemes for scheduling problems with time lags. *Journal of Scheduling*, 13:553–559.