SMNE UFPR
Simpósio de Métodos Numéricos em Engenharia
25 a 27 de outubro, 2017

Aplicação de Lazy Constraints para o Problema do Balanceamento de Linha de Modelo Misto

Thiago Cantos Lopes
Leandro Magatão

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial - CPGEI
Universidade Tecnológica Federal do Paraná - UTFPR
Av. Sete de Setembro, 3165 - Rebouças CEP 80230-901
Curitiba, Brasil

***Resumo*—Neste artigo, uma aplicação de *lazy constraints* é apresentada para avaliar a redução do esforço computacional associado à resolução de um problema combinatorial de otimização. O problema abordado foi o de balanceamento de linha, com minimização do tempo de ciclo como objetivo. As *lazy constraints* são restrições que podem ser adicionadas em um modelo base durante a execução do processo de busca executado em um *solver* de estado da arte (ex., CPLEX, Gurobi). A implementação das *lazy constraints* foi realizada por meio do uso de *Callbacks*. A cada nova solução incumbente obtida, inferências são feitas com o objetivo de adicionar restrições que reduzam o espaço de busca. Os testes computacionais realizados indicaram que para instâncias pequenas as restrições adicionadas acarretaram, na média, maiores tempos computacionais. Para instâncias maiores, contudo, uma tendência diferente foi observada: as *lazy constraints* contribuíram para redução do tempo total de processamento.**

***Palavras-chave*—Lazy constraints, Callbacks, Balanceamento de Linhas**

I. Introdução

O problema de balanceamento de linha foi primeiramente estudado em 1955 [1]. Consiste em alocar tarefas à estações de maneira a otimizar uma medida de performance (número de estações ou taxa de produção) respeitando restrições tecnológicas características da linha. Há um amplo campo de estudos deste problema, incluindo métodos para resolver suas formas mais simples [2] e uma ampla gama de variantes do mesmo [3]–[5].

Quando múltiplos produtos são produzidos na mesma linha, a mesma é chamada linha de modelo misto. Neste caso, análises mais sofisticadas são necessárias tendo em vista que o tempo de processamento de cada tarefa muda de acordo com o modelo de produto (podendo inclusive ser zero quando a tarefa não existe para aquele modelo em particular). Trabalhos anteriores [6], [7] mostram a conexão deste problema com aspectos de sequenciamento de modelo misto [8]. Nestas variantes, o número de variáveis do problema é significativamente aumentado, devido a variáveis (contínuas) de *scheduling*, necessárias para análises de temporização associadas ao problema. Por exemplo, [6] ilustra um modelo para determinação do regime permanente, dado uma sequência cíclica de produtos.

Em contextos de maior incerteza, analisar apenas uma sequência de produtos pode ser inadequado. Por outro lado, aumentar o número de sequências consideradas aumentará também o conjunto de variáveis contínuas de *scheduling* associadas às mesmas. Isso pode tornar as relaxações lineares dos nós de uma busca *Branch-and-Bound* [9] demasiado longas. Nesse sentido, restrições que reduzam o número de nós considerados em um processo de otimização tem o potencial teórico de ser muito úteis ao mesmo. O presente trabalho busca avaliar uma aplicação de *lazy constraints* [10] durante a execução do problema tendo por objetivo a redução do tempo computacional e do número de nós explorados durante o processo de otimização.

Os autores agradecem o apoio financeiro da Fundação Araucária (Acordos 141/2015, 06/2016 e 041/2017 FA-UTFPR-RENAULT), e ao CNPq (406507/2016-3).

De modo simplificado *lazy constraints* são restrições informadas pelo usuário durante a execução, elas podem ser restrições originais do problema que foram omitidas inicialmente, ou ainda restrições informadas pelo usuário segundo os critérios do mesmo.

O presente trabalho é estruturado da seguinte forma. A Seção II apresenta as definições empregadas ao longo do trabalho. A Seção III apresenta as restrições definidas. A Seção IV apresenta os experimentos e discute os materiais apresentados. A Seção V apresenta os resultados obtidos nos experimentos e os discute. A Seção VI apresenta direções para trabalhos futuros e a Seção VII as conclusões do presente artigo.

II. Definições

Seja T o conjunto de tarefas a serem alocadas a um conjunto de estações S. Cada variável binária de alocação $x_{t,s}$ indica uma possível alocação. Em um problema de balanceamento, relações de precedência são comuns: Seja P o conjunto de relações entre tarefas (t_1, t_2) tal que a tarefa t_2 deva ser alocada a uma estação não anterior a t_1. Cada tarefa t possui um tempo de processamento $d_{t,m}$ para cada modelo m (conjunto M) de produto. Seja $d_t = \sum_m d_{t,m}$ o tempo "total"de processamento de uma tarefa[1]. Busca-se determinar quais variáveis levam necessariamente a soluções inferiores à uma incumbente. Neste sentido, considera-se que deseja-se minimizar o tempo de ciclo (ou maximizar a taxa de produção) da linha dado o número de estações disponíveis. Para tal, considera-se c o atual tempo de ciclo de regime permanente da linha, associado à resposta incumbente.

Definem-se ainda as matrizes auxiliares P^*, ΔI, ΔB e ΔF. O matriz P^*_{ij} é a matriz de precedência generalizada, que pode ser obtida aplicando o algoritmo de Warshal [11] a partir de uma matriz com valores 1 para elemento a_{ij} tal que haja um vetor (t_i, t_j) no conjunto P. Dado que as relações de precedência são transitivas, a matriz resultante apresentará todas as relações diretas e indiretas de precedência. Essa matriz é usada para definir as outas três: ΔI_{ij} apresenta os tempos de processamento das tarefas entre t_i e t_j, ΔB_{ij} apresenta os tempos de processamento das tarefas anteriores a t_i e t_j, ΔF_{ij} apresenta os tempos de processamento das tarefas posteriores a t_i e t_j. As mesmas podem ser calculadas de acordo com as Equações 1, 2 e 3, observando-se os conectores lógicos *E* ($\wedge$) e *OU* ($\vee$) e obtendo-se os valores para todo i e j.

$$\Delta I_{ij} = \sum_{k \in T} (P^*_{ik} \wedge P^*_{kj}) \cdot d_{t_k} \tag{1}$$

$$\Delta B_{ij} = \sum_{k \in T/\{t_i, t_j\}} (P^*_{ki} \vee P^*_{kj}) \cdot d_{t_k} \tag{2}$$

$$\Delta F_{ij} = \sum_{k \in T/\{t_i, t_j\}} (P^*_{ik} \vee P^*_{kj}) \cdot d_{t_k} \tag{3}$$

[1]Aqui se considera que cada modelo de produto tem demanda igual, caso contrário deve-se fazer uma soma ponderada pelas demandas de cada produto.

Por fim, os vetores N_i e A_i representam, respectivamente, *tails* (caudas) e *heads* (cabeças) de cada tarefa [12]. Cuja interpretação é um limitante inferior do número necessário de estações posteriores[2] a cada tarefa t_i para atender o tempo de ciclo c. Estes podem ser recursivamente calculados ordenando as tarefas sucessoras a t_i em ordem decrescente de *tails*. Seja $(j_1, ...j_K)$ um destes ordenamentos para as tarefas sucessoras de t_i. O *tail* da tarefa t_i pode ser calculado de acordo com a Equação 4.

$$N_i = \max_{k \in 1..K} \left(N_{j_k} + \sum_{1 \leq l \leq k} \frac{d_{t_{j_k}}}{c} \right) \tag{4}$$

Uma vez que esse primeiro valor de N_i é computado, o mesmo deve ser arredondado para $[N_i]^+$ quando as seguintes duas condições forem verificadas: $N_i < [N_i]^+$ e $[N_i + d_{t_i}]^+ > [N_i]^+$. Um arredondamento análogo também se aplica a A_i.

Os vetores de *heads* & *tails* são úteis para a determinação de quais são as primeiras e últimas estações às quais uma tarefa pode ser alocada. Sejam E_{t_i} e L_{t_i} tais estações respectivamente, os mesmos são computados conforme as Equações 5 e 6, para cada tarefa t_i:

$$E_{t_i} = [N_i + d_{t_i}]^+ \tag{5}$$

$$L_{t_i} = |S| + 1 - [A_i + d_{t_i}]^+ \tag{6}$$

Estes limites de alocação[3] são computados a partir de um valor de tempo de ciclo c [12]. Se o objetivo for minimizar o tempo de ciclo, então, a cada nova solução incumbente, os valores acima descritos podem ser recalculados e levar a fortalecimentos dos limites mencionados.

III. Alocações Dual-Infactíveis

Na teoria da dualidade, infactibilidade dual está associada a sub-otimalidade primal [9]. Desta forma, podem ser consideradas dual-infactíveis e portanto eliminadas todas as alocações tarefa-estação (associadas às variáveis $x_{t,s}$) que impliquem um tempo de ciclo maior que o valor incumbente de c. Como no início da otimização de uma instância do problema de balanceamento não se sabe o valor ótimo de tempo de ciclo c^*, propõe-se recalcular os limites acima descritos e a cada solução incumbente obtida adicionar a *Lazy Constraint* descrita pela Equação 7:

$$x_{t,s} = 0 \qquad \forall\, t \in T \,\forall\, s \in S : \; s < E_t \cup s > L_t \tag{7}$$

Essa restrição descarta totalmente a variável garantidamente sub-ótima de consideração para o procedimento de busca. Desta

[2]posteriores para *tails*, anteriores para *heads*.

[3]Note que caso uma tarefa tenha E_{t_j} maior que o L_{t_i} de outra tarefa, então ainda que não haja relação de precedência entre t_i e t_j, na prática, t_i sempre será alocada antes t_j. Isso permite adicionar artificialmente esta precedência e re-computar os valores acima descritos de forma fortalecida.

forma, espera-se fortalecer o limitante e facilitar a obtenção de novas soluções via heurísticas do próprio *solver*: Uma vez que a variável necessariamente leva a respostas dominadas, retirá-la do problema deve levar a mais sucesso em busca locais.

Da mesma forma que a Restrição 7 acima permite considerar dual-infactíveis alocações de uma tarefa a uma estação, é possível ainda considerar dual-infactíveis alocações simultâneas de duas tarefas à mesma estação. Seja s a estação em consideração, e duas tarefas t_1 e t_2 tais que $E_{t_1} \leq s \leq L_{t_1}$ e $E_{t_2} \leq s \leq L_{t_2}$. Por essa hipótese ambas as tarefas individualmente são alocáveis a s sem gerar obrigatoriamente uma solução sub-ótima. Há, no entanto, três condições necessárias para que ambas as tarefas sejam realizadas na mesma estação sem gerar soluções dual-infactíveis: é preciso que todas as tarefas antecessoras de ambas estejam alocadas até aquela estação, que todas as tarefas entre ambas estejam alocadas na mesma estação e que todas as tarefas sucessoras de ambas estejam alocadas a partir daquela estação. Estas três condições necessárias geram os três casos em que as alocações simultâneas são infactíveis, os casos são dados pelas Inequações 8, 9 e 10. Note que essas condições existem valem para todo par de tarefas t_1 e t_2.

$$s < \frac{d_{t_1} + d_{t_2} + \Delta B_{t_1 t_2}}{c} \tag{8}$$

$$d_{t_1} + d_{t_2} + \Delta I_{t_1 t_2} > c \tag{9}$$

$$|S| + 1 - s < \frac{d_{t_1} + d_{t_2} + \Delta F_{t_1 t_2}}{c} \tag{10}$$

Munidos destas três condições é possível escrever a restrição de pares de alocações dual-infactíveis dado pela Inequação 11. Naturalmente seria possível descrever restrições para trios de alocações, mas isso tenderia a gerar um número muito grande de restrições (proporcional ao cubo do número de tarefas e ao número de estações). Foi considerado suficiente portanto, limitar as restrições aos pares de alocações.

$$x_{t_1,s} + x_{t_2,s} \leq 1 \quad \forall\, t_1, t_2, s \mid (8),(9) \text{ ou } (10) \tag{11}$$

Naturalmente, por mudar a cada novo valor de c essas restrições também podem ser adicionadas dinamicamente via *lazy constraints*.

IV. Experimentos - Dados e Configurações

O modelo de balanceamento ciente de sequência apresentado por [6] foi adaptado para considerar múltiplas sequências de produto. Ao invés de minimizar o tempo de ciclo de uma única sequência, minimiza-se a média dos tempos de ciclo das sequências consideradas. Tal valor médio é usado como c para a definição das *lazy constraints*.

Instâncias de modelo simples apresentadas por [13] foram adaptadas para gerar instâncias de modelo misto. Foram escolhidas as instâncias de maior valor de força de ordenamento (*order strength*[4]) para os testes. Instâncias foram tomadas cinco a cinco, o tempo de processamento de cada instância foi usado como tempo de cada modelo de produto. Para cada conjunto de tempos de processamento, cinco instâncias foram geradas, uma para o diagrama de precedência das instâncias de modelo simples. Ao todo, 25 conjuntos de dados de balanceamento de linha foram usados. O número de estações estabelecido foi 10 para todos os testes.

Cinco bateria de testes foram realizadas: na primeira, apenas uma sequência de produtos foi considerada; na segunda, duas sequências; na terceira, quatro sequências; na quarta, oito sequências; e na quinta, 24 sequências[5]. Para cada sequência adicionada, o número de variáveis contínuas incrementa em $3 \cdot |M| \cdot |S|$ cujo valor nos casos estudados é 150. Da mesma forma o número de restrições é incrementado por $3 \cdot |M| \cdot |S| - |M| - |S|$ cujo valor nos casos estudados é 135. O modelo matemático completo para este problema e que justifica estes números foi apresentado por [6].

Naturalmente, por possuírem mais variáveis e restrições, nos casos com mais sequências pode-se esperar um maior tempo de resolução de relaxações lineares. Assim, é esperado que as *lazy constraints* sejam mais impactantes para casos maiores.

Todos os testes foram executados em um computador com Intel(R) Core(TM) i7-3610QM (2.3 GHz) com 8.0 Gigabytes de memória RAM. O *solver* empregado foi o Gurobi [10], versão 7.0 e as implementações de *lazy constraints* foram programadas em vb.net através de *Callbacks*[6].

V. Resultados e Discussões

Ao aplicar as técnicas anteriormente descritas, foi observado que o comportamento do *solver* foi diferente para cada conjunto de dados. A Tabela I apresenta as médias de tempos de processamento (medido em segundos), número médio de nós abertos e número médio de iterações por nó. Cada linha da mesma reporta resultados no conjunto de testes com um número diferente de sequências cuja performance média foi otimizada.

Tabela I: Resultados Gerais

	Configurações Padrão			Com *Lazy Constraints*		
N.Seq.	Tempo	N. Nós	Iter/Nó	Tempo	N. Nós	Iter/Nó
1	5	2336	22	9	1745	26
2	25	7050	44	23	5767	34
4	41	5572	89	34	5379	51
8	70	4801	162	51	4402	105
24	292	6706	481	178	5579	385

[4]Medida da força das relações de precedência, onde zero significa nenhuma relação de precedência e um representa uma única sequência possível para a execução das tarefas

[5]Para as primeiras quatro baterias foram usadas as sequências: (1,2,3,4,5), (5,4,3,2,1), (3,4,2,5,1), (3,2,4,1,5), (2,3,1,4,5), (2,1,3,5,4), (4,3,5,2,1), (4,5,3,1,2). Todas essas são sequências ciclicamente distintas das demais. Para a última, foram usadas todas as sequências cíclicas possíveis para cinco peças distintas.

[6]Objetos que permite execução de código do usuário durante a otimização por parte do *solver*.

Nota-se que para os casos menores, as técnicas descritas implicaram um aumento de tempo computacional. Isso ocorreu apesar da redução do número de nós e de ter-se observado apenas um pequeno aumento no número de iterações por nó. Pode-se inferir que o uso das *callbacks* implica uma alteração do modo de busca do Gurobi, podendo levar a reduções de performance. Note, no entanto, que para os casos maiores as técnicas descritas reduziram os tempos computacionais médios. Quanto maiores as instâncias, maior o ganho de performance. Note que houve uma redução significativa do número de nós estudados e também do número de iterações por nó. Isso sugere que para instâncias grandes, apesar de mais restrições serem adicionadas, dado que o número base de restrições do problema já é grande, o método Simplex tem mais facilidade de resolver as relaxações lineares após a adição das *lazy constraints* propostas.

Tabela II: Diferenças médias por conjunto de dados

	Reduções de Tempo		Reduções de Nós	
N.Seq.	Percentual	Número	Percentual	Número
1	-	0	25.3%	14
2	8.0%	14	18.2%	17
4	17.1%	20	3.5%	12
8	27.1%	20	8.3%	15
24	39.4%	25	16.8%	19

O comportamento observado não foi uniforme dentro de cada conjunto de dados de teste. A Tabela II apresenta as médias das diferenças em número de nós e tempo computacional e o número de instâncias (dentre as 25) em que uma diferença favorável ocorreu devido às *lazy constraints*. Note que em diversos casos o emprego das mesmas não acarretou em ganho de desempenho.

Estas diferenças de performance podem ser devidas à características próprias do *solver*. Apesar de "globalmente"o impacto das *lazy constraints* ter sido positivo, é interessante verificar o que ocorre nas instâncias em que o mesmo foi negativo. Para tal, dividiu-se para cada instância resolvida o maior pelo menor valor de tempo de processamento (o mesmo foi feito para o número de nós). Desta forma pode-se tomar a média das razões para os casos em o impacto foi positivo e para os casos em que o impacto é negativo. Estes valores estão reportados na Tabela III.

Tabela III: Razões médias entre resultados por instância

	Inst. c/ Reduções		Inst. c/ Aumentos	
N.Seq.	Tempo	N. Nós	Tempo	N. Nós
1	-	58.5%	51.0%	31.5%
2	23.7%	41.2%	21.3%	25.9%
4	27.5%	37.1%	20.0%	25.9%
8	34.6%	44.4%	15.3%	34.2%
24	42.8%	37.5%	-	20.0%

Excetuando o primeiro conjunto de dados, em que as *lazy constraints* levaram consistentemente a pioras de tempo, é possível afirmar que o impacto observado é majoritariamente positivo. Para as instâncias em que houve ganho em tempo ou número de nós explorados, o ganho foi proporcionalmente maior que a perda ocorrida em instâncias em que houve perda em tempo ou número de nós explorados.

VI. Trabalhos Futuros

Duas direções de possíveis trabalhos futuros foram identificadas pelos autores. A primeira consiste em fortalecer a execução do modelo matemático através de uma *solução heurística inicial*. A segunda em fortalecer as *lazy constraints* desenvolvidas.

Uma solução heurística inicial pode não apenas auxiliar o processo de busca do *solver*, mas também permitiria usar as técnicas de redução aqui apresentadas para iniciar o modelo com diversas alocações dual-infactíveis já devidamente proibidas. A depender da qualidade da solução inicial (e da distância até limitantes inferiores) é possível até mesmo decidir pela utilização das *lazy constraints* ou não: Foi identificado que o *solver* sacrifica algumas de suas funções para poder usar tais restrições. Se uma análise preliminar indicar que a maioria das restrições pode ser adicionadas no modelo base a partir de uma solução heurística, é possível optar por usar apenas as restrições disponíveis graças a solução inicial e seguir o processo de busca com as ferramentas padrão do *solver*. Alternativamente pode-se aplicar as *lazy constraints* para um valor arbitrário de c, o qual se acredite com razoável confiança ser válido, ainda que não se possua uma solução cujo tempo de ciclo seja c. As técnicas aqui desenvolvidas são válidas sempre que o valor de c for válido, não é necessário para tal estar munido de uma solução que o confirme.

Por fim, deve-se comentar que é possível a cada nova solução incumbente de tempo de ciclo c^*, aplicar as técnicas aqui descritas para o valor de c igual a $c^* - 1$. Neste caso, se o modelo residual for infactível é possível afirmar que não há solução melhor ou igual a $c^* - 1$, implicando otimalidade da incumbente. Como valores menores de c tendem a gerar listas maiores de alocações dual-infactíveis, é possível esperar *fortalecimento* das restrições por meio desta alteração, ao menos em casos específicos. No entanto, ao fazer isso, é possível que a solução incumbente seja considerada "infactível"pelo modelo após a inclusão da *lazy constraint*, o que requer um armazenamento manual desta solução e um esforço complementar para prevenir o modelo de buscar por outras incumbentes com solução igual a c^*. É possível ainda, para casos maiores gerar relações de dual-infactibilidade para trios de alocações tarefas, ou ainda para alocações de tarefas em estações distintas.

VII. Conclusões

Lazy constraints são restrições que podem ser adicionadas ao modelo durante o processo de otimização. São ferramentas disponibilizadas por *solvers* de estado da arte tais quais o CPLEX [14] e o Gurobi [10]. Tais *solvers* permitem a adição destas restrições por parte do usuário através de *callbacks*. Neste artigo, uma aplicação das mesmas é apresentada para o problema de balanceamento de linha com o objetivo de minimização de tempo de ciclo.

A cada nova solução incumbente obtida, é possível nesta classe de problemas, inferir que algumas das possibilidades inicialmente consideradas levaram à soluções sub-ótimas. Desta forma, a adição de restrições que, durante a execução, descartem estas variáveis é interessante. Neste artigo foram apresentadas duas variantes de restrições que podem descartar tais opções sub-ótimas.

Testes computacionais indicaram que para problemas pequenos as restrições podem acarretar em maiores tempos computacionais. Uma possível razão para tal aumento se encontra na necessária desabilitação de procedimentos padrão do *solver*. O uso de *Callbacks* por si só parece alterar quais procedimentos estão disponíveis durante a otimização.

No entanto, para casos maiores, foram observadas reduções em tempo de processamento. Com efeito, quanto maior as instâncias, mais expressiva foi a redução de tempo de processamento obtida pelas técnicas aqui apresentadas. Possíveis razões para tal são: a possibilidade de mais rapidamente descartar nós no campo de busca e a menor quantidade de nós cuja exploração se mostra necessária durante a otimização.

Direções para trabalhos futuros são também discutidas, visando incorporar as técnicas aqui descritas como parte do pré-processamento do problema. Outra possibilidade consiste ainda no fortalecimento das restrições aqui apresentadas.

REFERÊNCIAS

[1] M. E. Salveson, "The assembly line balancing problem," *The Journal of Industrial Engineering*, vol. 6, pp. 18–25, 1955.

[2] A. Scholl and C. Becker, "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing," *European Journal of Operational Research*, vol. 168, pp. 666–693, 2006.

[3] N. Boysen, M. Fliedner, and A. Scholl, "Assembly line balancing: Which model to use when?" *Intern. Journal of Production Economics*, vol. 111, pp. 509–528, 2008.

[4] O. Battaïa and A. Dolgui, "A taxonomy of line balancing problems and their solution approaches," *International Journal of Production Economics*, vol. 142, pp. 259–277, 2013.

[5] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing," *European Journal of Operational Research*, vol. 168, pp. 694–715, 2006.

[6] T. C. Lopes, C. G. S. Sikora, and L. Magatão, "Buffer and Cyclical Product Sequence Aware Assembly Line Balancing Problem: Model and Steady-State Balancing Case Study," in *Annals of the XLVIII SBPO*, Vitória-ES, Brazil, 2016, pp. 3458–3469.

[7] T. C. Lopes, C. G. S. Sikora, A. S. Michels, and L. Magatão, "A New Model for Simultaneous Balancing and Cyclical Sequencing of Asynchronous Mixed-Model Assembly Lines with Parallel Stations," in *Annals of the XLIX SBPO*, 2017, pp. 1–12.

[8] N. Boysen, M. Fliedner, and A. Scholl, "Sequencing mixed-model assembly lines: Survey, classification and model critique," *European Journal of Operational Research*, vol. 192, pp. 349–373, 2009.

[9] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, 10th ed. Heidelberg: Mc Graw Hill, 2015.

[10] Gurobi, *Gurobi Optimizer Reference Manual*. Gurobi, 2016.

[11] J. L. Gersting, *Fundamentos Matemáticos para a Ciência da Computação*, 3rd ed., LTC, Ed. LTC, 1995.

[12] A. Scholl, *Balancing and sequencing assembly lines*, 2nd ed. Heidelberg: Physica, 1999.

[13] A. Otto, C. Otto, and A. Scholl, "Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing," *European Journal of Operational Research*, vol. 228, no. 1, pp. 33–45, 2013.

[14] Cplex, *IBM ILOG CPLEX User's Manual for CPLEX*. IBM, 2014.