Programação de tarefas em um ambiente *flow shop* com m máquinas para a minimização do desvio absoluto total de uma data de entrega comum

Julio Cesar Delgado Vasquez

Texto apresentado
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Ernesto G. Birgin

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, outubro de 2017

Programação de tarefas em um ambiente *flow shop* com m máquinas para a minimização do desvio absoluto total de uma data de entrega comum

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 28/08/2017. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Ernesto Julian Goldberg Birgin (orientador) - IME-USP
- Prof. Dr. Marco Aurélio de Mesquita - EP-USP
- Prof. Dr. Alexandre Cesar Muniz de Oliveira - UFMA

Agradecimentos

A mis padres, hermanos y sobrinas. Sin su apoyo no podría caminar por la vida.

Al profesor Ernesto y su esposa Débora, por su apoyo durante mi estancia en el instituto.

A mis fantásticos amigos, a los compañeros de clases y del trabajo, por los bonitos momentos que pasé con ustedes.

A Dios por hacer nuestro mundo perfecto.

Resumo

Neste trabalho abordamos o problema de programação de tarefas em um ambiente *flow shop* permutacional com mais de duas máquinas. Restringimos o estudo para o caso em que todas as tarefas têm uma data de entrega comum e restritiva, e onde o objetivo é minimizar a soma total dos adiantamentos e atrasos das tarefas em relação a tal data de entrega. É assumido também um ambiente estático e determinístico. Havendo soluções com o mesmo custo, preferimos aquelas que envolvem menos tempo de espera no *buffer* entre cada máquina. Devido à dificuldade de resolver o problema, mesmo para instâncias pequenas (o problema pertence à classe NP-difícil), apresentamos uma abordagem heurística para lidar com ele, a qual está baseada em busca local e faz uso de um algoritmo linear para atribuir datas de conclusão às tarefas na última máquina. Este algoritmo baseia-se em algumas propriedades analíticas inerentes às soluções ótimas. Além disso, foi desenvolvida uma formulação matemática do problema em programação linear inteira mista (PLIM) que vai permitir validar a eficácia da abordagem. Examinamos também o desempenho das heurísticas com testes padrões (*benchmarks*) e comparamos nossos resultados com outros obtidos na literatura.

Palavras-chave: flow shop, permutação, heurística, adiantamento, atraso, data de entrega comum, algoritmo de timing.

Abstract

In this work we approach the permutational flow shop scheduling problem with more than two machines. We restrict the study to the case where all the jobs have a common and restrictive due date, and where the objective is to minimize the total sum of the earliness and tardiness of jobs relative to the due date. A static and deterministic environment is also assumed. If there are solutions with the same cost, we prefer those that involve less buffer time between each machine. Due to the difficulty of solving the problem, even for small instances (the problem belongs to the NP-hard class), we present a heuristic approach to dealing with it, which is based on local search and makes use of a linear algorithm to assign conclusion times to the jobs on the last machine. This algorithm is based on some analytical properties inherent to optimal solutions. In addition, a mathematical formulation of the problem in mixed integer linear programming (MILP) was developed that will validate the effectiveness of the approach. We also examined the performance of our heuristics with benchmarks and compared our results with those obtained in the literature.

Keywords: flow shop, heuristic, earliness, tardiness, common due date, permutation, timing algorithm.

Sumário

1 Introdução **1**
1.1 Revisão bibliográfica . . . 2
1.2 Organização do trabalho . . . 3

2 Definição e Formulação Matemática do Problema **5**
2.1 Definição do problema . . . 5
2.2 Modelos de programação linear inteira mista . . . 9
2.3 Modelagem matemática do problema . . . 10
2.4 Validação da formulação matemática . . . 13

3 Heurísticas Propostas **15**
3.1 Propriedades analíticas das soluções ótimas para o caso de duas máquinas . . . 15
3.2 O algoritmo de *timing* . . . 17
3.3 Busca local . . . 19
3.3.1 Vizinhanças . . . 21
3.3.2 Vizinhança por inserção . . . 21
3.3.3 Vizinhança por troca . . . 22
3.4 Algoritmos de melhora iterativa . . . 22
3.4.1 Maior melhora iterativa . . . 23
3.4.2 Primeira melhora iterativa . . . 23
3.4.3 Busca local reduzida . . . 23
3.5 Método heurístico proposto . . . 24
3.6 Minimização da espera no *buffer* . . . 26
3.7 Experimentos numéricos . . . 28

4 Conclusões **41**
4.1 Perspectivas futuras . . . 41

A Implementação do modelo PLIM **42**

B Cota superior para a data de entrega restritiva **44**

Referências Bibliográficas **45**

Capítulo 1

Introdução

Neste trabalho se estuda o problema de programação de tarefas em um ambiente *flow shop* permutacional com m máquinas, procurando minimizar a soma total dos adiantamentos e atrasos das tarefas em relação a uma data de entrega comum, além de restritiva. O problema pode ser pensado como de uma linha de produção em uma fábrica a qual consiste em um conjunto de máquinas dispostas em série por onde deve passar um certo número de *produtos* para sua fabricação. Tais lotes passam através das máquinas um por um e em uma sequência fixa. Cada um deles tem uma data prometida para ser entregue e o objetivo é encontrar uma ordem particular de processamento dos produtos que propicie a maior pontualidade e, em consequência, a menor quantidade possível de perdas devido aos custos que resultam de terminar de fabricar os produtos antes ou depois de sua data de entrega. Em teoria de programação de tarefas é comum nos referirmos a tais produtos ou ordens de produção simplesmente como de *tarefas*.

Tradicionalmente, os critérios de otimalidade para este tipo de problema só procuravam minimizar o tempo de atraso em que as tarefas eram entregues, permitindo assim reduzir os inconvenientes que surgem de ter tarefas atrasadas. Como é bem conhecido, ao ter atrasos nas datas de entrega dos produtos as empresas podem sofrer penalidades contratuais, perda de clientes e credibilidade, além de prejudicar a imagem da empresa ao procurar novos clientes (Sen e Gupta 1984, Sakuraba *et al.* 2009).

De outro lado, surgem outro tipo de inconvenientes se decidirmos antecipar a fabricação dos produtos. Aumento dos custos de inventário, deterioração dos produtos ou concentração de capital ocioso são alguns exemplos (Biskup e Feldmann, 2001). Então, tendo em conta as desvantagens causadas pela adoção de um único desses esquemas e considerando que às vezes é inevitável ter tarefas adiantadas e atrasadas (por exemplo, quando as datas de entrega das tarefas são muito próximas), surgiu na indústria a necessidade de outros critérios de otimalidade na hora das empresas planejar a execução de suas operações.

Na década de 1950 se originou no Japão uma nova abordagem à gestão chamada *Just In Time* (JIT), que tem sido adotada pela indústria japonesa desde os anos 1970. Taiichi Ohno, o criador da doutrina JIT, define ela como "tornar disponível a peça certa, no momento certo e na quantidade certa, para entrar na montagem" (Ohno, 1982). A filosofia de gestão por traz do JIT é buscar continuamente maneiras de tornar os processos mais eficientes com o objetivo final de produzir bens e serviços sem incorrer em nenhum desperdício.

Portanto, seguindo a filosofia JIT é natural estabelecer um critério de otimalidade que seja compatível com ela, isto é, um critério que procure finalizar a fabricação dos produtos o mais perto possível de sua data de entrega (nem demasiado tarde nem cedo demais). Deste modo surgiu o critério chamado

soma de desvios absolutos (SDA), também conhecido como *soma de adiantamentos e atrasos total*. Este trabalho usa tal critério para procurar soluções ótimas para o problema de programação de tarefas de tipo *flow shop* permutacional.

No início dos anos 1990, o problema *flow shop* permutacional com o critério SDA foi provado ser NP-difícil (Hall *et al.*, 1991) e por conseguinte, é pouco provável encontrar soluções exatas em tempo razoável para instâncias de tamanho mediano e grande. Deste modo, está justificado o uso de métodos heurísticos para sua resolução. Portanto, a proposta deste trabalho é estender para o caso de três ou mais máquinas, as heurísticas projetadas por Sakuraba *et al.* (2009). Tais heurísticas são apoiadas por propriedades analíticas do problema e são baseadas em algoritmos de busca local (Aarts e Lenstra 2003, Michiels *et al.* 2007). Esses algoritmos são técnicas de exploração iterativa de um espaço de soluções e o que procuram em cada iteração é "pular" da solução atual para alguma solução vizinha, isto é, uma solução próxima da solução atual (segundo alguma métrica). Comparamos (para pequenas instâncias) as soluções obtidas pelas heurísticas com as obtidas pelo modelo matemático implementado em um software comercial. Finalmente, avaliamos a eficácia das heurísticas através de um estudo comparativo dos resultados obtidos com bancos de teste conhecidos e outros da literatura.

Começamos apresentando um modelo matemático que será útil para dar uma descrição clara e completa do problema e especificamos de maneira detalhada as variáveis e restrições que compõem o modelo. Depois estudamos algumas características específicas das soluções ótimas e as usamos para estabelecer propriedades analíticas que servem para encontrar estas soluções. Para consolidar ideias, implementamos o modelo na linguagem de modelagem AMPL e resolvemos algumas instâncias que vão nos servir depois como ponto de referência na avaliação das heurísticas propostas.

1.1 Revisão bibliográfica

Um dos primeiros resultados analíticos em relação a problemas do tipo *flow shop* foi apresentado por Johnson (1954). Nesse trabalho o autor descreve um algoritmo de ordem $O(n \log n)$ que encontra uma programação ótima para um *flow shop* de duas máquinas com objetivo de minimizar o tempo total de finalização das tarefas ou *makespan* (C_{max}), conhecido agora como o *algoritmo* ou *regra de Johnson*. Muitos outros trabalhos continuaram com esse esquema de funcionamento, mas mudaram o critério de otimização para outros também regulares, como são o tempo total de conclusão das tarefas, o tempo médio de fluxo e o tempo de atraso máximo , entre outros. Ao mesmo tempo, aparecem novas técnicas para abordar o problema, incluindo abordagens exatas e métodos heurísticos (Sung e Min 2001, Yeung *et al.* 2004, Lauff e Werner 2004, Sakuraba *et al.* 2009).

Em meados da década de 1980, os pesquisadores começaram a dirigir seus esforços à resolução de problemas *flow shop* com funções objetivo não regulares. Uma função objetivo não regular não reflete um comportamento monótono em relação ao tempo de finalização das tarefas. Desse modo surgem desafios adicionais no momento de procurar soluções ótimas. Uma das funções objetivo não regulares mais estudadas na literatura é a soma de adiantamentos e atrasos das tarefas em relação a uma data de entrega comum. Existem vários artigos que servem como compêndios de resultados sobre este tipo de problemas, entre eles Baker e Scudder (1990), Kanet e Sridharan (2000) e Gordon *et al.* (2002).

Inicialmente, grande maioria de trabalhos sobre adiantamentos e atrasos com data de entrega comum foram dirigidos a problemas com uma única máquina. Um deles foi Kanet (1981), o qual descreve as condições necessárias e suficientes para que uma solução seja ótima. Kanet, que assume uma data

de entrega não restritiva, descreve um algoritmo $O(n \log n)$ que consegue encontrar *uma* solução ótima, mas não é capaz de encontrar *todas* as soluções ótimas. Bagchi *et al.* (1986) trabalham também com uma única máquina, mas estendem a análise de Kanet para o caso de uma data de entrega restritiva. Szwarc (1989) aborda também o problema de máquina única e divide sua análise em dois casos: quando a data de início da programação é arbitrária e quando é fixa. Nesse trabalho, o autor propõe um procedimento *branch and bound* para quando a data de começo das tarefas é fixa. Hall *et al.* (1991) consideram o problema de máquina única e data de entrega comum e restritiva e, depois de estabelecer algumas propriedades analíticas das soluções ótimas, baseiam-se em tais propriedades para provar que o problema é NP-difícil. Pouco antes, e independentemente, Hoogeveen e Van de Velde (1991) provaram o mesmo fato.

Sarper (1995) foi o primeiro a tratar o problema estendido a duas máquinas. Nesse trabalho, é dada uma formulação matemática do problema baseada em programação linear inteira mista (PLIM) e propõem-se algumas heurísticas junto com um estudo do desempenho de cada uma delas. Outro trabalho importante, e no qual está baseado o presente trabalho, é o de Sakuraba *et al.* (2009). Nesse trabalho os autores analisam o problema de duas máquinas com data de entrega comum e restritiva, além de apresentar propriedades particulares do problema. Tais propriedades são utilizadas depois para projetar um algoritmo $O(n)$ de atribuição de valores aos tempos de início das tarefas, que eles chamam de "timing" em referência a algoritmos similares para o problema de adiantamentos e atrasos com datas de entrega diferentes (Hendel e Sourd, 2007). Esse algoritmo é usado na segunda etapa das heurísticas SH_1, SH_2 e SH_3 que também propõem e tem a vantagem de obter uma programação ótima em tempo linear para uma sequência de tarefas construída na primeira etapa. Os autores também fazem um estudo da performance das heurísticas propostas e mostram que essas heurísticas superam os resultados encontrados na literatura. Outro trabalho importante é o de Chandra *et al.* (2009), onde estudam o problema geral com m máquinas. Eles classificam sua análise em três casos diferentes segundo o valor da data de entrega e apresentam uma heurística para o caso de uma data de entrega restritiva. Assim, temos outro trabalho que vai nos servir de referência no desenvolvimento da pesquisa.

Ronconi e Birgin (2012) fizeram um estudo dos modelos em programação linear inteira mista (PLIM) mais utilizados para formular problemas de programação de tarefas no ambiente *flow shop*. Tais modelos podem ser usados para gerar soluções ótimas para problemas de tamanho moderado. A conclusão dos autores é que, dentre todos os modelos analisados, o proposto por Wagner (1959) é o que precisa de menos tempo computacional para encontrar soluções exatas quando é implementado em um software comercial, além de precisar de uma menor quantidade de variáveis e restrições que os outros modelos estudados. Outros trabalhos com uma análise similar são Pan (1997) e Stafford *et al.* (2005).

Finalmente, para mencionar alguns trabalhos que tratam aplicações na vida real de problemas *flow shop* temos: aplicações na indústria química (Dudek *et al.*, 1992), indústria de cabos (Dag, 2013), indústria farmacêutica (Chandra *et al.*, 2009), entre outros (Pinedo, 2012).

1.2 Organização do trabalho

O restante deste texto está organizado como segue. No Capítulo 2, descrevemos detalhadamente o problema central, definindo-o matematicamente e posteriormente apresentamos o desenvolvimento de uma formulação matemática em programação linear inteira mista para o problema. O Capítulo 3 faz um resumo dos resultados obtidos por Sakuraba *et al.* (2009) para o caso de duas máquinas e tam-

bém descreve uma proposta para estender as heurísticas para m máquinas. Finalmente, o Capítulo 4 apresenta as conclusões e as perspectivas futuras para o desenvolvimento da pesquisa.

Capítulo 2

Definição e Formulação Matemática do Problema

Neste capítulo apresentamos a notação a ser usada durante o resto do trabalho e também os conceitos que ajudam a descrever com precisão o problema de estudo. Apresentamos também o critério sob o qual consideramos uma programação ser ótima e uma formulação matemática em programação linear inteira mista que será útil para encontrar soluções ótimas de instâncias pequenas do problema. Na parte final, descrevemos um experimento numérico com a finalidade de testar a formulação.

2.1 Definição do problema

Vamos definir claramente o problema de interesse. Todo o trabalho será desenvolvido sob o ambiente de produção *flow shop*. Neste ambiente tem-se *m máquinas* em série as quais devem processar *n tarefas*, conhecidas na língua inglesa como *jobs*. De forma mais rigorosa, seja $\mathscr{M} = (M_1, \ldots, M_m)$ uma sequência fixa de máquinas e $\mathscr{J} = \{J_1, \ldots, J_n\}$ um conjunto de tarefas que precisam ser processadas por tais máquinas. Cada tarefa J_i é composta de exatamente *m operações*, isto é, $J_i = (O_{i1}, \ldots, O_{im}), \forall i \in \{1, \ldots, n\}$ onde a k-ésima operação da tarefa J_i é denotada por O_{ik} e deve ser processada obrigatoriamente pela máquina M_k. Associada a cada operação O_{ik} temos um inteiro positivo $p_{ik} > 0$ que indica o *tempo de processamento* que demanda essa operação. Também, a cada tarefa J_i associamos o vetor $p_i = (p_{i1}, \ldots, p_{im})$ que contém os tempos de processamento de todas as operações da tarefa. Cabe ressaltar que o ambiente *flow shop* tem como característica principal que a ordem das operações de cada tarefa é específica e todas as tarefas passam pelas máquinas na mesma ordem (a operação O_{ik} só pode ser processada na máquina M_k).

O problema tem as seguintes restrições: (a) para toda tarefa J_i, a operação $O_{i,k+1}$ tem que ser iniciada depois da operação $O_{ik}, \forall k \in \{1, \ldots, m-1\}$; (b) cada máquina só pode processar uma única operação em um determinado momento; (c) todas as tarefas estão disponíveis para o processamento desde o tempo 0; (d) o processamento em todas as máquinas é feito sem *interrupção*, ou seja, toda operação é processada desde seu inicio até o final sem ser interrompida; (e) entre cada par de máquinas consecutivas M_k e M_{k+1} existe um *buffer* (ilimitado) que permite armazenar as tarefas completadas por M_k (seguindo uma política de tipo PEPS – *Primeiro que Entra, Primeiro que Sai*) até que M_{k+1} termine de processar a tarefa que a mantém ocupada; (f) cada tarefa pode ser processada em uma máquina de cada vez; (g) quando a operação O_{ik} é finalizada em M_k, a operação $O_{i,k+1}$ está imediatamente disponível para começar em

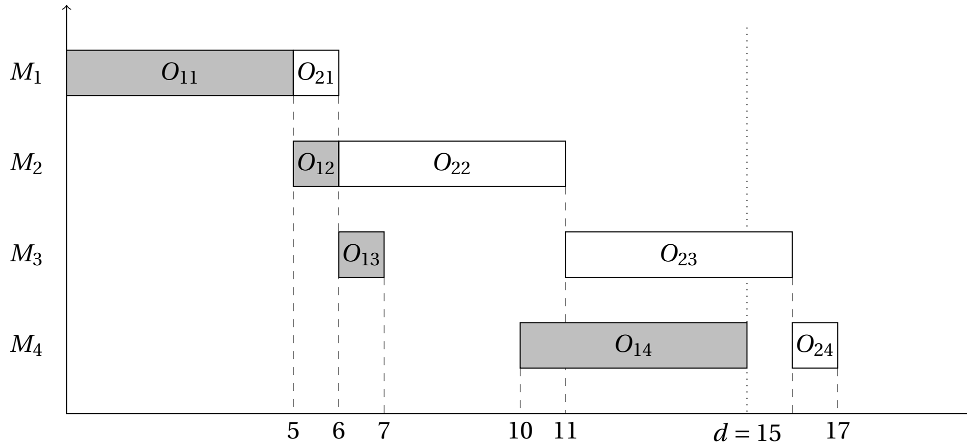


Figura 2.1: *Distribuição das tarefas em um flow shop com 2 tarefas, 4 máquinas e data de entrega* $d = 15$.

M_{k+1} e (h) é assumido um ambiente estático e determinístico, quer dizer que tanto o número de máquinas como o número de tarefas e seus correspondentes tempos de processamento são conhecidos *a priori* e permanecem fixos ao longo do processo.

Para cada O_{ik} definimos a variável inteira c_{ik} que denota o *tempo de conclusão da operação* O_{ik} na máquina M_k. Também, definimos a variável $C_i = c_{im}, \forall i \in \{1,\ldots,n\}$ como o *tempo de conclusão da tarefa* J_i, a qual indica o momento em que todas as operações da tarefa são concluídas. Com cada tarefa J_i nos é dado um parâmetro d_i chamado *data de entrega*, que expressa a data de conclusão prometida. Nosso problema tem a restrição que a data de entrega é *comum* para todas as tarefas, ou seja, $d_i = d, \forall i \in \{1,\ldots,n\}$.

Na Figura 2.1, apresentamos um exemplo fazendo uso de uma ferramenta visual chamada *diagrama de Gantt*, que serve para ilustrar a distribuição das tarefas em cada máquina ao longo do tempo. No exemplo, temos: $m = 4$, $n = 2$, $\mathscr{M} = \{M_1, M_2, M_3, M_4\}$, $\mathscr{J} = \{J_1, J_2\}$, $J_1 = \{O_{11}, O_{12}, O_{13}, O_{14}\}$, $J_2 = \{O_{21}, O_{22}, O_{23}, O_{24}\}$, $p_1 = (p_{11}, p_{12}, p_{13}, p_{14}) = (5,1,1,5)$, $p_2 = (p_{21}, p_{22}, p_{23}, p_{24}) = (1,5,5,1)$ e $d = 15$. Além disso, $C_1 = 15$ e $C_2 = 17$.

Este exemplo é descrito como segue: primeiro é processada em M_1 a operação O_{11} desde o instante 0 até o instante 5. Em seguida, começa o processamento de O_{21} na mesma máquina e, simultaneamente, O_{12} em M_2 desde o instante 5 até o instante 6. O processamento continua do mesmo modo até terminar as operações restantes. Pode-se observar que entre M_3 e M_4 existe um *tempo de inatividade inserido* (TII) entre as operações O_{13} e O_{14} a fim de que a tarefa J_1 termine exatamente na data de entrega.

Uma *sequência de processamento* das tarefas (ou simplesmente *sequência* de tarefas) é a ordem na qual as tarefas são processadas em uma determinada máquina. No exemplo da Figura 2.1 temos que a sequência é a mesma em todas as máquinas: primeiro se processa J_1 e depois J_2. Em geral, para um problema com várias máquinas pode acontecer que a sequência de tarefas em cada máquina seja diferente mesmo que as restrições acima mencionadas fossem satisfeitas. A Figura 2.2 apresenta esse caso e mostra outra possível distribuição das tarefas para o mesmo exemplo. Vemos que em M_2 a sequência das tarefas é diferente das outras máquinas e J_1 deve esperar no *buffer* entre M_1 e M_2 até que M_2 termine de processar J_2 (operação O_{22}).

Para uma única máquina tem-se $n!$ possíveis sequências nas quais as tarefas podem ser processadas (considerando o número de permutações de n objetos) e, no total, para m máquinas existe $(n!)^m$ sequências diferentes nas quais poderiam ser processadas as n tarefas. Neste trabalho, nos limitamos ao caso em que a sequência de processamento em todas as máquinas é a mesma. Com essa restrição se diz

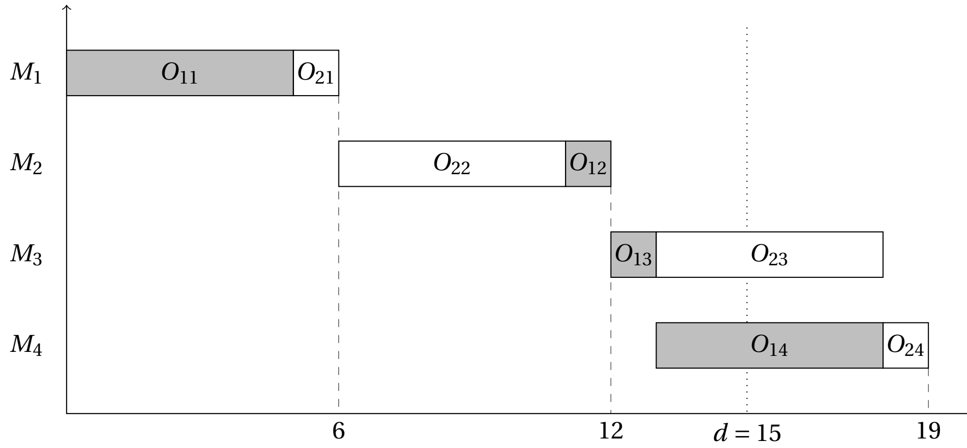


Figura 2.2: *Outra possível distribuição das tarefas para o exemplo.*

que o problema é de *permutação*, também conhecido como *flow shop permutacional*. Assim, o número de sequências possíveis é reduzido a $n!$.

Definimos uma *programação factível* σ (ou simplesmente uma *programação*) como uma atribuição de valores às variáveis $c_{ij}, \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}$ que satisfaz as restrições (a)-(h) acima mencionadas. Por exemplo, a programação σ_1 da Figura 2.1 atribui os seguintes valores: $(c_{11}, c_{12}, c_{13}, c_{14}) = (5, 6, 7, 15)$ e $(c_{21}, c_{22}, c_{23}, c_{24}) = (6, 11, 16, 17)$, enquanto na Figura 2.2 a programação σ_2 é: $(c_{11}, c_{12}, c_{13}, c_{14}) = (5, 12, 13, 18)$ e $(c_{21}, c_{22}, c_{23}, c_{24}) = (6, 11, 18, 19)$. O conjunto Π de todas as programações factíveis recebe o nome de *região factível* ou *viável*.

Quando uma tarefa é completada antes da data de entrega d, diz-se que a tarefa tem um *adiantamento* E_i (*earliness*), o qual é definido como $E_i = d - C_i$. Do mesmo modo, quando a tarefa é completada depois da data de entrega, diz-se que tem um *atraso* T_i (*tardiness*) definido como $T_i = C_i - d$. Cabe ressaltar que esta definição é um pouco ambígua pois aparentemente uma tarefa tem adiantamento negativo se conclui sua execução depois da data de entrega; o mesmos se aplica ao caso de ter tarefas atrasadas. Para evitar esta confusão, trabalhamos com as seguintes definições:

$$E_i = \max\{d - C_i, 0\}, \tag{2.1}$$

$$T_i = \max\{C_i - d, 0\}. \tag{2.2}$$

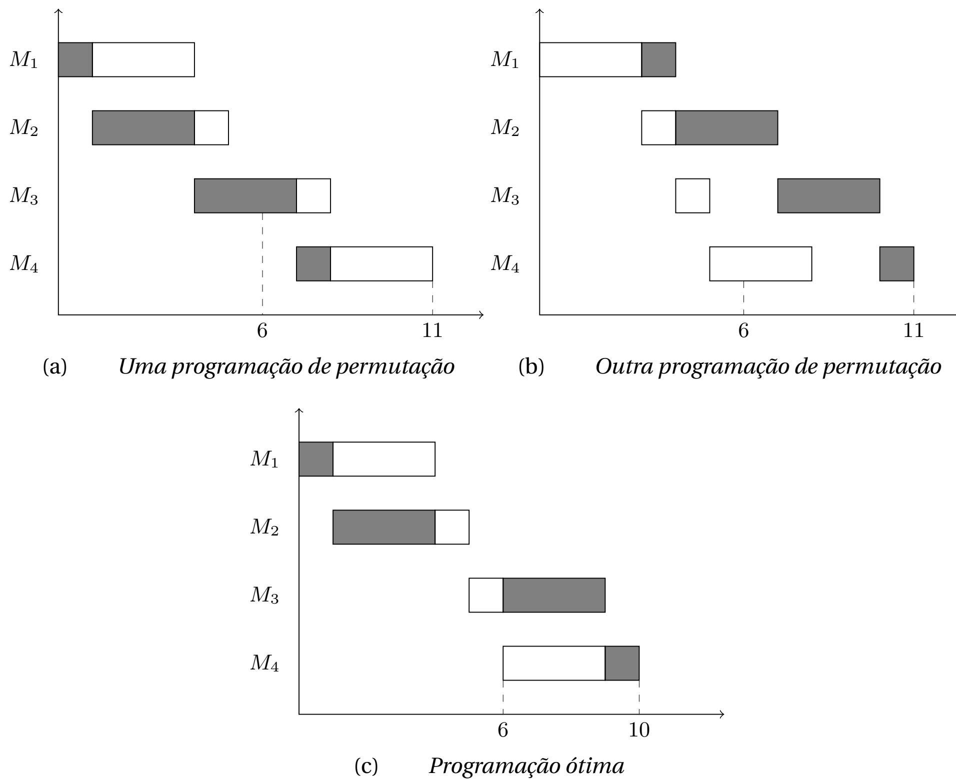
É interessante notar que para qualquer tarefa J_i, $E_i > 0$ implica $T_i = 0$. Igualmente, se $T_i > 0$ então $E_j = 0$. Também resulta da definição que $E_i, T_i \geqslant 0$.

Com a finalidade de diferenciar a "qualidade" das diferentes programações que possam surgir para o problema, definimos um critério de otimalidade que vai de acordo como nosso objetivo principal, isto é, a filosofia JIT. Seja $\sigma \in \Pi$ uma programação qualquer para as n tarefas que compõem o problema. Então, definimos a função objetivo como:

$$f(\sigma) = \sum_{i=1}^{n} (E_i + T_i), \tag{2.3}$$

conhecida como a *soma dos adiantamentos e atrasos* das tarefas ou também como *soma dos desvios absolutos* (SDA). O nosso objetivo principal é encontrar uma programação $\sigma^* \in \Pi$ que minimize a função f, isto é, $f(\sigma^*) \leqslant f(\sigma), \forall \sigma \in \Pi$. Neste caso, dizemos que a programação é *ótima* e pode ser que tal programação não seja única (um caso muito frequente).

Nem toda programação ótima é de permutação. Isto é, pode existir uma programação ótima na qual



(a) *Uma programação de permutação* (b) *Outra programação de permutação*

(c) *Programação ótima*

Figura 2.3: *Programações para o problema $F4||C_{max}$ com duas tarefas.*

a ordem em que as tarefas são processadas em cada máquina seja diferente. A Figura 2.3 mostra um exemplo deste fato (Emmons e Vairaktarakis, 2013). O exemplo tem duas tarefas e quatro máquinas com tempos de processamento $p_1 = (1,3,3,1)$, $p_2 = (3,1,1,3)$, e o objetivo é minimizar o *makespan* C_{max} (tempo de conclusão da última tarefa na última máquina). As duas únicas programações de permutação possíveis (Figuras 2.3(a) e 2.3(b)) tem *makespan* $C_{max} = 11$, enquanto a programação ótima para o problema tem $C_{max} = 10$ (Figura 2.3(c)) e a ordem das tarefas varia de máquina em máquina.

No entanto, não devemos subestimar a importância das programações de permutação pois essas programações são as mais fáceis de implantar na vida real (Kim, 1995).

Com o esquema de classificação para problemas de programação de tarefas proposto por Lawler *et al.* (1982), denotamos o problema de estudo como $Fm|\, prmu, d_i = d | \sum(E_i + T_i)$. Nessa notação pode-se diferenciar três campos (separados pelo símbolo“|”). No primeiro tem-se Fm, onde a letra F indica que trabalhamos em um ambiente *flow shop* e o número de máquinas que temos disponíveis é m. O segundo contém dois termos: *prmu*, indicando que apenas consideramos programações de permutação e $d_i = d$, que a data de entrega será a mesma para todas as tarefas. Por último, no terceiro campo tem-se $\sum(E_i + T_i)$, indicando que o critério de otimalidade adotado é o de minimizar a soma de adiantamentos e atrasos total.

Outro aspecto interessante do problema é que sua complexidade varia de acordo com o valor da data de entrega. Existem dois principais tipos de data de entrega comum, a saber, o restritivo e o não restritivo. No caso *não restritivo*, a data de entrega é uma variável de decisão cujo valor deve-se calcular ou, se é dado, não tem influência sobre a sequência ótima (Feldmann e Biskup, 2003). Isso ocorre, por exemplo, sempre que o valor da data de entrega é maior ou igual à soma dos tempos de processamento de todas as tarefas. Porém, se a data de entrega é dada e tem influência na busca da sequência de tarefas ótima, então é considerada restritiva. Hall *et al.* (1991) provaram que o problema de máquina única

e data de entrega restritiva é NP-difícil e, em consequência, também é o problema com m máquinas. Hoogeveen e Van de Velde (1991) provaram o mesmo fato quase simultaneamente. Neste trabalho vamos considerar apenas datas de entrega restritivas por ser o caso mais interessante e porque já existem na literatura algoritmos polinomiais para o caso não restritivo (Kanet 1981, Bagchi *et al.* 1986).

2.2 Modelos de programação linear inteira mista

Um *programa linear* (PL), com frequência chamado simplesmente *programa*, é um problema que consiste em minimizar ou maximizar uma expressão linear sujeita a um conjunto (finito) de restrições também lineares. Essas restrições podem ser tanto de igualdade (=) quanto desigualdade (⩽, ⩾), e também é frequente ter restrições para o domínio das variáveis do programa. Os valores para essas variáveis são tomados comumente dos reais $\mathbb{R}$. Como exemplo temos o seguinte PL:

$$\begin{array}{rrcl} \min & -5x_1 - 4x_2 - 3x_3 & & \\ \text{s.a} & 2x_1 + 3x_2 + x_3 & \leqslant & 5 \\ & x_1 + x_3 & = & 3 \\ & -4x_1 - x_2 - 2x_3 & \geqslant & -11 \\ & 3x_1 + 4x_2 + 2x_3 & \leqslant & 8 \\ & x_1, x_2, x_3 & \geqslant & 0. \end{array}$$

Aqui, as variáveis do programa são x_1, x_2 e x_3 e a expressão $-5x_1 - 4x_2 - 3x_3$ é chamada *função objetivo*. Em geral, desejamos encontrar valores para as variáveis x_i que atendam todas as restrições e minimizem (ou maximizem) o valor dessa função. Um conjunto de valores para as x_i que satisfaça as restrições é chamado *solução viável* ou simplesmente *solução*. Uma solução que minimiza (ou maximiza) o valor da função objetivo se chama *solução ótima* e o valor da função correspondente se chama *valor ótimo*. Uma solução ótima não necessariamente é única. No exemplo, uma solução seria $(x_1, x_2, x_3) = (1.5, 0, 1.5)$ e uma solução ótima $(x_1, x_2, x_3) = (2, 0, 1)$.

O *método simplex* é um algoritmo clássico para resolver PLs que foi proposto no ano 1947 por George B. Dantzig. Existem trabalhos anteriores ao de Dantzig onde outros autores desenvolveram de maneira independente métodos similares ao simplex para tratar casos especiais de PLs (Dantzig, 1990). Klee e Minty (1972) elaboraram um PL com n variáveis que o método simplex resolve em $2^n - 1$ iterações, provando deste modo sua complexidade exponencial. Pouco tempo depois, Khachiian (1979) mostrou um algoritmo polinomial para resolver PLs chamado *método elipsoidal* e, assim, pôs fim a um dos problemas abertos mais antigos da computação teórica. Para uma introdução mais extensa ao tema consultar Chvátal (1983).

Um *programa linear inteiro misto* é um programa linear onde o domínio de alguma das variáveis é um conjunto discreto (por exemplo $\mathbb{Z}$). Quando todas as variáveis são inteiras então falamos de um *programa linear inteiro puro*. Assim, um *modelo PLIM* é um programa linear inteiro misto que representa matematicamente as características do problema em estudo. Deste modo, as restrições e função objetivo do programa capturam as propriedades e dinâmica do problema.

2.3 Modelagem matemática do problema

O modelo PLIM descrito a seguir foi adaptado do trabalho de Ronconi e Birgin (2012), no qual eles estudam o desempenho computacional de vários modelos PLIM, entre eles um que foi elaborado por Wagner (1959). Nesse trabalho, os autores abordam o problema de minimizar o adiantamento e atraso total em relação a datas de entrega diferentes. Uma de suas conclusões é que o modelo de Wagner pode ser resolvido mais rápido do que os outros quando implementado em um software comercial, sem importar que precise de um maior número de variáveis binárias para sua formulação.

A seguir, descrevemos matematicamente as restrições do problema. Primeiramente, o problema *flow shop* permutacional consiste em atribuir as n tarefas a uma sequência de n posições, portanto, definimos a seguinte variável binaria:

$$x_{ij} = \begin{cases} 1 & \text{se a tarefa } J_i \text{ é atribuída à } j\text{-ésima posição,} \\ 0 & \text{em caso contrário.} \end{cases}$$

A seguinte restrição expressa o fato que apenas podemos atribuir uma tarefa à primeira posição:

$$\sum_{i=1}^{n} x_{i1} = 1.$$

Em geral, isso deve-se cumprir para todas as posições da sequência, isto é,

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n. \tag{2.4}$$

Do mesmo modo, a seguinte restrição indica que a tarefa J_1 deve ser atribuída a uma única posição:

$$\sum_{j=1}^{n} x_{1j} = 1,$$

e em geral

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n. \tag{2.5}$$

Definimos mais algumas variáveis. Seja W_{jk} o tempo que deve esperar a tarefa atribuída à posição j (j-ésima tarefa) no *buffer* ou *fila* que existe entre as máquinas M_k e M_{k+1}. Também, seja I_{jk} o tempo que a máquina M_k permanece inativa depois de processar a j-ésima tarefa enquanto espera à $(j+1)$-ésima tarefa. Devido às restrições físicas do problema, tem-se $W_{jk}, I_{jk} \geqslant 0$. Por último, seja I_0 o tempo de inatividade da primeira máquina até começar a processar a primeira tarefa.

A restrição a seguir pode ser melhor compreendida ao observar a Figura 2.4.

$$C_1 = I_0 + \left(\sum_{i=1}^{n} x_{i1} p_{i1} + W_{11}\right) + \left(\sum_{i=1}^{n} x_{i1} p_{i2} + W_{12}\right) + \cdots + \left(\sum_{i=1}^{n} x_{i1} p_{i,m-1} + W_{1,m-1}\right) + \sum_{i=1}^{n} x_{i1} p_{im}, \tag{2.6}$$

A equação (2.6) calcula o tempo de conclusão da primeira tarefa (C_1) como a soma dos tempos de processamento e espera da tarefa em todas as máquinas, além de I_0. O termo $\sum_{i=1}^{n} x_{i1} p_{ik}$ representa o tempo de processamento da primeira tarefa na máquina M_k. Em geral, $\sum_{i=1}^{n} x_{ij} p_{ik}$ representa o tempo de processamento da j-ésima tarefa na máquina M_k. A equação pode ser expressada de um modo mais

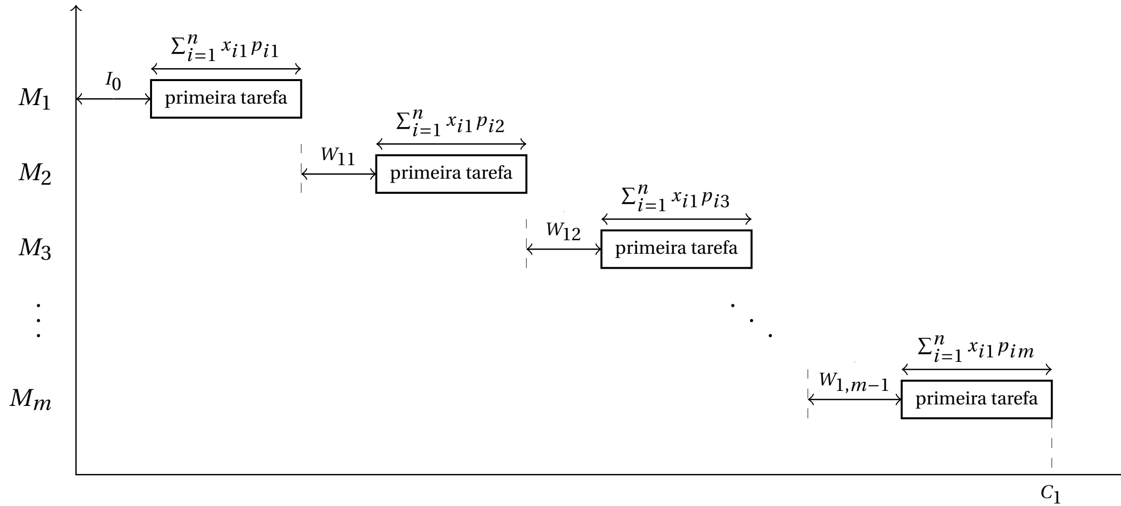


Figura 2.4: *Calculo do tempo de conclusão da primeira tarefa na última máquina.*

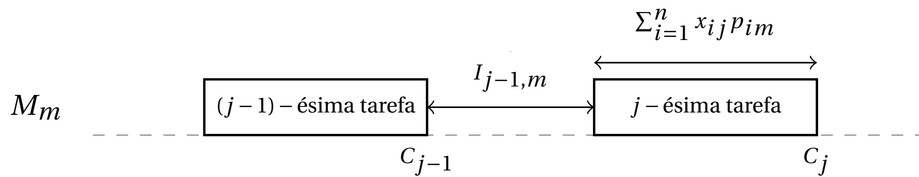


Figura 2.5: *Calculo do tempo de conclusão da j-ésima tarefa na última máquina.*

compacto como

$$C_1 = I_0 + \sum_{k=1}^{m} \left(\sum_{i=1}^{n} x_{i1} p_{ik} \right) + \sum_{i=1}^{m-1} W_{1i}. \tag{2.7}$$

Também, a igualdade:

$$C_j = C_{j-1} + I_{j-1,m} + \sum_{i=1}^{n} x_{ij} p_{im}, \qquad j = 2, \dots, n, \tag{2.8}$$

calcula o tempo de conclusão da j-ésima tarefa ($j = 2, \dots n$) como a soma de três termos: (a) o tempo de conclusão da tarefa anterior, (b) o tempo de inatividade de M_m depois de processar a tarefa anterior e (c) seu tempo de processamento em M_m (Figura 2.5).

Outra restrição importante é:

$$I_{jk} + \sum_{i=1}^{n} x_{i,j+1} p_{ik} + W_{j+1,k} = W_{jk} + \sum_{i=1}^{n} x_{ij} p_{i,k+1} + I_{j,k+1}, \qquad j = 1, \dots, n-1; k = 1, \dots, m-1, \tag{2.9}$$

a qual pode-se interpretar da Figura 2.6 como o fato que a soma dos tempos de inatividade, processamento e espera para cada par de tarefas consecutivas deve ser a mesma, isto é, o tempo decorrido em ambas as máquinas M_k e M_{k+1} entre o instante de término do processamento da j-ésima tarefa em M_k (instante t_1 na figura) e o instante de início do processamento $(j+1)$-ésima tarefa em M_{k+1} (instante t_2) devem ser iguais.

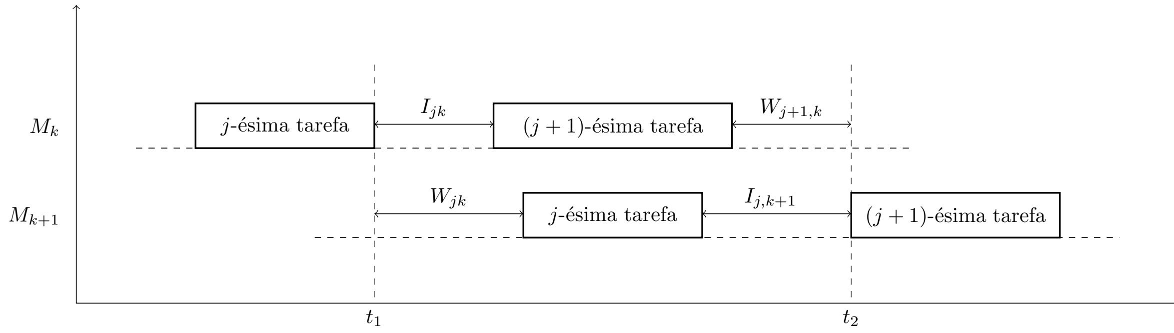


Figura 2.6: *Programação das tarefas em posição j e $j+1$ nas máquinas M_k e M_{k+1}.*

Parâmetro	Descrição
n	Número de tarefas
m	Número de máquinas
p_{ik}	Tempo de processamento da tarefa J_i na máquina M_k
d	Data de entrega comum

Tabela 2.1: *Parâmetros do modelo PLIM.*

Para finalizar, as restrições a seguir são a versão linearizada das equações (2.1) e (2.2):

$$T_j \geqslant C_j - d, \qquad j = 1, \ldots, n, \tag{2.10}$$

$$E_j \geqslant d - C_j, \qquad j = 1, \ldots, n. \tag{2.11}$$

As Tabelas 2.1 e 2.2 mostram um resumo das variáveis e parâmetros usados para a modelagem. Juntando a função objetivo (2.3) e as restrições (2.4), (2.5) e (2.7)-(2.11), temos o seguinte modelo PLIM:

$$\min \quad \sum_{j=1}^{n} \left(E_j + T_j\right) \tag{2.12}$$

s.a

$$C_1 = I_0 + \sum_{k=1}^{m} \left(\sum_{i=1}^{n} x_{i1} p_{ik} \right) + \sum_{i=1}^{m-1} W_{1i}, \tag{2.13}$$

$$C_j = C_{j-1} + I_{j-1,m} + \sum_{i=1}^{n} x_{ij} p_{im}, \qquad j = 2, \ldots, n, \tag{2.14}$$

$$I_{jk} + \sum_{i=1}^{n} x_{i,j+1} p_{ik} + W_{j+1,k} = W_{jk} + \sum_{i=1}^{n} x_{ij} p_{i,k+1} + I_{j,k+1}, \qquad j = 1, \ldots, n-1; k = 1, \ldots, m-1, \tag{2.15}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n, \tag{2.16}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n, \tag{2.17}$$

$$T_j \geqslant C_j - d, \qquad j = 1, \ldots, n, \tag{2.18}$$

$$E_j \geqslant d - C_j, \qquad j = 1, \ldots, n. \tag{2.19}$$

Além disso, temos as seguintes restrições para os domínios das variáveis: $x_{ij} \in \{0,1\}, \forall i, j \in \{1, \ldots, n\}$; $I_0 \geqslant 0$; $E_j, T_j \geqslant 0, \forall j \in \{1, \ldots, n\}$; $W_{jk} \geqslant 0, \forall j \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, m-1\}$; $I_{jk} \geqslant 0, \forall j \in \{1, \ldots, n-1\}, \forall k \in$

Variável	Descrição
x_{ij}	1 se a tarefa J_i é atribuída à j-ésima posição da sequência, 0 em caso contrario
E_j	Adiantamento da j-ésima tarefa
T_j	Atraso da j-ésima tarefa
C_j	Tempo de conclusão da j-ésima tarefa (na última máquina)
W_{jk}	Tempo de espera (no *buffer)* da j-ésima tarefa entre as máquinas M_k e M_{k+1}
I_{jk}	Tempo de inatividade entre as tarefas j e $(j+1)$-ésima na máquina M_k
I_0	Tempo de inatividade da primeira máquina até processar a primeira tarefa

Tabela 2.2: *Variáveis do modelo PLIM.*

	M_1	M_2	M_3	M_4	M_5
J_1	80	41	66	27	93
J_2	94	57	63	16	14
J_3	87	29	84	32	8
J_4	63	6	10	42	70
J_5	18	70	96	69	23

Tabela 2.3: *Tempos de processamento para um* flow shop *com 5 tarefas ($n = 5$) e 5 máquinas ($m = 5$).*

$\{1, \ldots, m\}$.

O número de variáveis binarias, contínuas e restrições são n^2, $2mn+2n-m+1$ e $nm+4n-m+1$, respectivamente.

2.4 Validação da formulação matemática

É importante contar com um modelo PLIM por dois motivos: (a) tem-se uma descrição matemática precisa e completa das restrições que compõem o problema e (b) devido à natureza intratável do mesmo, podemos usar o modelo para encontrar soluções ótimas para instâncias de tamanho pequeno.

O modelo PLIM da seção anterior foi codificado na linguagem de modelagem AMPL (A Mathematical Programming Language, Fourer *et al.* (2002)) e foram resolvidas algumas instâncias pequenas de Chandra *et al.* (2009) usando o IBM ILOG CPLEX Optimization Studio 12.3.6, deste modo, corroborando os resultados obtidos pelos autores. O código fonte do modelo (arquivo *flowshop.mod*) pode ser acessado no endereço http://www.ime.usp.br/~jdelgado/flowshop/.

A título de exemplo, mostramos uma rotina de trabalho no AMPL IDE (http://ampl.com/products/ide/) que resolve a primeira das 50 instâncias de tamanho $n = 5$ e $m = 5$ de Chandra *et al.* (2009). Os tempos de processamento de cada operação para essa instância são mostrados na Tabela 2.3 e a data de entrega restritiva é $d = 334$. A rotina é mostrada no Listado 2.1.

Dos resultados pode-se observar que a sequência ótima é

$$\sigma^* = (J_4, J_5, J_2, J_1, J_3),$$

e seu valor correspondente

$$f(\sigma^*) = 374.$$

No Apêndice A são listados os arquivos *flowshop.mod* e *chandran5m5d334.cplex.dat* com a implementação do modelo e a representação em AMPL do exemplo da Tabela 2.3, respectivamente.

```
ampl: reset;
ampl: model flowshop.mod;
ampl: data chandran5m5d334.cplex.dat;
option solver cplexamp;
solve;
CPLEX 12.6.3.0: optimal integer solution; objective 374
118 MIP simplex iterations
0 branch-and-bound nodes
No basis.
ampl: display C; # Tempos de conclusão das tarefas
C [*] :=
 1  316
 2  339
 3  353
 4  496
 5  504
 ;

x [*,*]
 :   1   2   3   4   5   :=
 1   0   0   0   1   0
 2   0   0   1   0   0
 3   0   0   0   0   1
 4   1   0   0   0   0
 5   0   1   0   0   0
 ;
```

Listado 2.1: *Rotina de execução do exemplo da Tabela 2.3.*

Capítulo 3

Heurísticas Propostas

Neste capítulo descrevemos detalhadamente as heurísticas desenvolvidas por Sakuraba *et al.* (2009) para o caso de duas máquinas. Essas heurísticas resultam da combinação de três conceitos: (a) um método para a construção de uma "boa" sequência de tarefas inicial chamado NEH (Nawaz *et al.*, 1983), (b) alguns esquemas de busca local (completa e reduzida) para gerar sequências de tarefas localmente ótimas e (c) um algoritmo linear chamado algoritmo de *timing*, que dada uma sequência fixa de tarefas, obtém uma programação ótima para a sequência. Da mistura desses conceitos, Sakuraba *et al.* (2009) projetaram várias heurísticas que implementaram e testaram com um conjunto de instâncias de teste de tamanho variado.

O objetivo do presente capítulo é propor uma generalização dessas heurísticas para o caso de m máquinas. Primeiramente, explicamos as propriedades nas quais se baseia o algoritmo de *timing* e descrevemos sua estrutura e modo de funcionamento. Em seguida, revisamos os esquemas de busca local que iremos utilizar na concepção das heurísticas (incluindo o esquema de busca local reduzida usado por Sakuraba *et al.* (2009)). Posteriormente, explicamos detalhadamente a estrutura de trabalho das heurísticas e propomos uma forma de adaptar o algoritmo de *timing* para o caso de m máquinas. Depois, descrevemos um procedimento para obter soluções mais eficientes no sentido que evitam o tempo de permanência desnecessário das tarefas no *buffer* do *flow shop*, e sugerimos como modificar o problema para aproveitar este benefício. Ao final, descrevemos alguns experimentos computacionais que mostram que nossas heurísticas melhoram os resultados obtidos por outros autores que abordaram o mesmo problema; além disso, propomos um conjunto de dados maior baseado no trabalho de Taillard (1993) para estudar mais satisfatoriamente o desempenho das heurísticas e apresentamos os resultados de tais experimentos.

3.1 Propriedades analíticas das soluções ótimas para o caso de duas máquinas

Agora descrevemos algumas propriedades propostas por Sakuraba *et al.* (2009) (para o caso de duas máquinas) que são úteis para projetar o algoritmo de *timing*, descrito na seguinte seção.

Como mencionado por Sarper (1995), dado que o valor da função objetivo depende unicamente dos tempos de conclusão das tarefas na segunda máquina, podemos começar a processar as tarefas na primeira o mais rápido possível. Nesse caso dizemos que a programação tem um *tempo de inatividade inserido* (TII). A Figura 3.1 mostra um exemplo com dados tomados da Tabela 3.1, onde se observa que

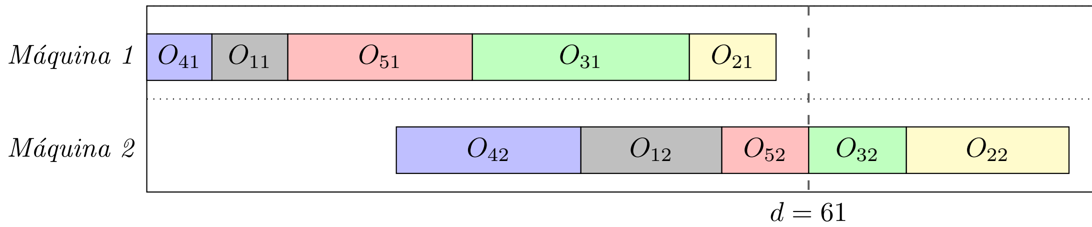


Figura 3.1: *Uma programação com tempos de inatividade inseridos (TII) para cada tarefa.*

	M_1	M_2
J_1	7	13
J_2	8	15
J_3	20	9
J_4	6	17
J_5	17	8

Tabela 3.1: *Tempos de processamento para um* flow shop *com 5 tarefas, 2 máquinas e $d = 61$.*

cada tarefa deve esperar algum tempo no *buffer* entre as máquinas M_1 e M_2.

Propriedade 3.1. *Em uma programação ótima, todas as operações na segunda máquina com tempos de conclusão antes ou sobre a data de entrega d, são processadas sem interrupção (uma imediatamente depois da outra) em um bloco único.*

Demonstração. A prova é por contradição. Vamos supor que existe uma programação ótima σ^* que tem duas operações O_{i2} e O_{j2} na segunda máquina com $c_{i2}, c_{j2} \leqslant d$, $c_{i2} < c_{j2}$ e um tempo de inatividade de t unidades entre elas. Então, poderíamos atrasar o processamento da operação O_{i2} em t unidades (isto é, juntar O_{i2} a O_{j2}) e conseguir outra programação σ com valor objetivo t unidades menor do que σ^*, o qual contradiz sua otimalidade. □

Não acontece o mesmo no caso das operações que terminam depois de d, pois nem sempre podemos adiantar o processamento de uma operação até coincidir com o tempo de conclusão da anterior operação. Isso é devido ao fato de que a tarefa poderia ainda estar sendo processada na máquina anterior.

Propriedade 3.2. *Se o número de operações adiantadas é maior que o número de operações não adiantadas (isto é, operações a tempo ou atrasadas) então podemos gerar um ganho se atrasamos por algum tempo o processamento de um bloco de operações consecutivas na segunda máquina, desde que esse atraso não interfira com a programação das demais tarefas.*

Demonstração. Obtém-se um ganho desde que o número de tarefas adiantadas mantém-se maior que o número de tarefas não adiantadas. O ganho é calculado como o produto de dois fatores: a quantidade do atraso e a diferença entre o número de tarefas adiantadas e não adiantadas. Esse ganho é devido a que existem mais tarefas que diminuem seu adiantamento que tarefas que aumentam seu atraso. □

Propriedade 3.3. *Em uma programação ótima, uma tarefa atrasada deve começar seu processamento na segunda máquina em um instante que é o maior entre seu tempo de conclusão na primeira máquina e o tempo de conclusão da tarefa anterior na segunda máquina.*

Demonstração. Não há razão para começar depois o processamento dessa tarefa pois o que desejamos é diminuir seu atraso. Devido às restrições físicas do problema, o mínimo tempo em que pode começar o processamento da tarefa é a maior dessas duas quantidades. □

Propriedade 3.4. *Toda programação ótima tem no máximo $\lfloor n/2 \rfloor$ tarefas adiantadas.*

Demonstração. A prova também é por contradição. Suponhamos que existe uma programação ótima σ^* com i tarefas adiantadas. Se atrasamos todas as tarefas (adiantadas e não adiantadas) por uma quantidade $\epsilon > 0$, pequena o suficiente para manter a quantidade de tarefas adiantadas, então obtemos uma diminuição de $i\epsilon - (n-i)\epsilon = (2i-n)\epsilon$ na função objetivo. Se $i > n/2$, então esta última quantidade torna-se positiva, o que significa que conseguimos reduzir a função objetivo. Isto contradiz a otimalidade de σ^*. Portanto $i \leqslant n/2$, e por conseguinte $i \leqslant \lfloor n/2 \rfloor$. □

3.2 O algoritmo de *timing*

O algoritmo de *timing* foi projetado originalmente por Sakuraba *et al.* (2009) para resolver o problema de duas máquinas com data de entrega única e penalidades de adiantamento e atraso distintas para cada tarefa, ou em notação de Lawler *et al.* (1982), $F2|prmu, d_i = d|\sum(\alpha_i E_i + \beta_i T_i)$. Porém, a forma em que aborda o problema é resolvendo um problema equivalente. Como afirmam os autores, o tempo de conclusão de uma tarefa na primeira máquina pode ser visto como o instante de liberação da tarefa r_i (*release date*) para que possa iniciar seu processamento na segunda máquina. Então, podemos começar a processar as tarefas na primeira máquina o mais rápido possível (como na Figura 3.1) e visualizar seus tempos de conclusão como se fossem os instantes de liberação para a segunda máquina. Por tanto, o problema é equivalente a um de máquina única com instantes de liberação r_i para as tarefas, isto é, $1|r_i, d_i = d|\sum E_i + T_i$ (em nosso caso $\alpha_i = \beta_i = 1$).

O algoritmo de *timing* (Algoritmo 1) aceita como entrada uma sequência de tarefas fixa (completa ou parcial), além da data de entrega d (restritiva ou não) e os tempos de processamento p_i e liberação r_i das tarefas, e devolve os valores S, C e j. Em virtude da Propriedade 3.1, ao finalizar o algoritmo as variáveis S e C contém os tempos de início e conclusão, respectivamente, do bloco de tarefas nas posições $1, \dots, j-1$ que são executadas de forma consecutiva. As tarefas restantes (posições $j, \dots, n$) são processadas no maior tempo entre seu instante de liberação e o instante de conclusão da tarefa imediatamente anterior.

No algoritmo, as variáveis têm a seguinte interpretação: δ representa o adiantamento máximo do bloco que pode ser atingido sem que nenhuma tarefa inicie antes de seu instante de liberação, α é o número de tarefas a tempo ou adiantadas, β o número de tarefas atrasadas e h representa o índice da primeira tarefa atrasada do bloco.

O funcionamento do algoritmo é como descrito a seguir. Devido à Propriedade 3.1, as tarefas a tempo ou adiantadas são processadas no início como um bloco único (sem interrupções). Na j-ésima iteração, a tarefa na posição j da sequência é inserida na última posição do bloco desde que seu instante de liberação seja menor do que o tempo de conclusão da última tarefa do bloco (Propriedade 3.3). Logo, se $\beta \geq \alpha$, o bloco inteiro é adiantado em uma quantidade de tempo que depende do valor de δ: se δ é maior que o tempo de processamento da primeira tarefa atrasada do bloco (p_h) então o bloco inteiro é adiantado nessa quantidade, e as variáveis α, β e h são atualizadas, senão, o bloco é adiantado δ unidades e o algoritmo termina. É importante ressaltar que este algoritmo é executado em tempo $O(n)$ pois em

Algoritmo 1 Algoritmo de *timing*

Entrada: $n, d \in \mathbb{Z}$ e $p, r \in \mathbb{Z}^n$.

Saída: As tarefas nas posições $1, \ldots, j-1$ são programadas entre S e C consecutivamente (sem interrupções). As tarefas nas posições $j, \ldots, n$ o mais rápido possível depois depois C.

1: $S \leftarrow d$, $C \leftarrow d$, $\delta \leftarrow d$
2: $\alpha \leftarrow 0$, $\beta \leftarrow 0$, $h \leftarrow 1$, $j \leftarrow 1$
3: **enquanto** $j \leqslant n$, $\delta > 0$ e $r_j < C$ **faça**
4: $\quad C \leftarrow C + p_j$
5: $\quad \beta \leftarrow \beta + 1$
6: $\quad \delta \leftarrow \min\{\delta, C - p_j - r_j\}$
7: $\quad$ **enquanto** $\delta > 0$ e $\beta \geqslant \alpha$ **faça**
8: $\quad\quad$ **se** $\delta \geqslant p_h$ **então**
9: $\quad\quad\quad \alpha \leftarrow \alpha + 1$, $\beta \leftarrow \beta - 1$
10: $\quad\quad\quad S \leftarrow S - p_h$, $C \leftarrow C - p_h$, $\delta \leftarrow \delta - p_h$
11: $\quad\quad\quad h \leftarrow h + 1$
12: $\quad\quad$ **senão**
13: $\quad\quad\quad S \leftarrow S - \delta$, $C \leftarrow C - \delta$
14: $\quad\quad\quad \delta \leftarrow 0$
15: $\quad\quad$ **fim se**
16: $\quad$ **fim enquanto**
17: $\quad j \leftarrow j + 1$
18: **fim enquanto**

cada iteração é inserida uma tarefa por vez e as variáveis α e β restringem o laço **enquanto** interno a ser executado uma única vez em cada iteração.

Continuando com o exemplo, os tempos de conclusão $(C_1, C_2, C_3, C_4, C_5) = (53, 85, 70, 40, 61)$ da programação da Figura 3.1 foram obtidos através da execução do algoritmo de *timing* com a sequência de tarefas fixada a $(J_4, J_1, J_5, J_3, J_2)$, a data de entrega $d = 61$ e os tempos de processamento da Tabela 3.1. Os instantes de liberação $(r_4, r_1, r_5, r_3, r_2) = (6, 13, 30, 50, 58)$ foram obtidos ao processar as tarefas na primeira máquina o mais rápido possível. Na Figura 3.2 pode-se observar o modo de proceder do algoritmo. Para a iteração j, se mostra primeiro o instante em que é inserida a tarefa da j-ésima posição da sequência no final do bloco e logo o adiantamento efetuado ao bloco inteiro (se há algum).

Uma análise da definição de E_i e T_i (equações (2.1) e (2.2)) e da função objetivo (equação (2.3)) nos permite reescrever esta última como

$$f(\sigma) = \sum_{i=1}^{n} (E_i + T_i) = \sum_{i=1}^{n} |C_i - d|, \tag{3.1}$$

deste modo, podemos visualizar a função objetivo como a soma de *funções convexas lineares por partes* $f_i(C_i) = |C_i - d|$. Além disso, como as tarefas do primeiro bloco são executadas sem interrupções, e supondo que o bloco consta de q tarefas, podemos calcular os tempos de conclusão C_i de cada tarefa do bloco como $C_i = S + P_i$, onde S é o instante do início do processamento do bloco e P_i é definido como $P_i = \sum_{k=1}^{i} p_k$, $\forall i \in \{1, \ldots, q\}$. Também, os C_i para as tarefas nas posições $q+1, \ldots, n$ são calculadas como $C_i = r_i + p_i$ (Propriedade 3.3). Então, a equação (3.1) é reescrita como

$$f(\sigma) = \sum_{i=1}^{n} |C_i - d| = \sum_{i=1}^{q} |S + P_i - d| + \sum_{i=q+1}^{n} |r_i + p_i - d|. \tag{3.2}$$

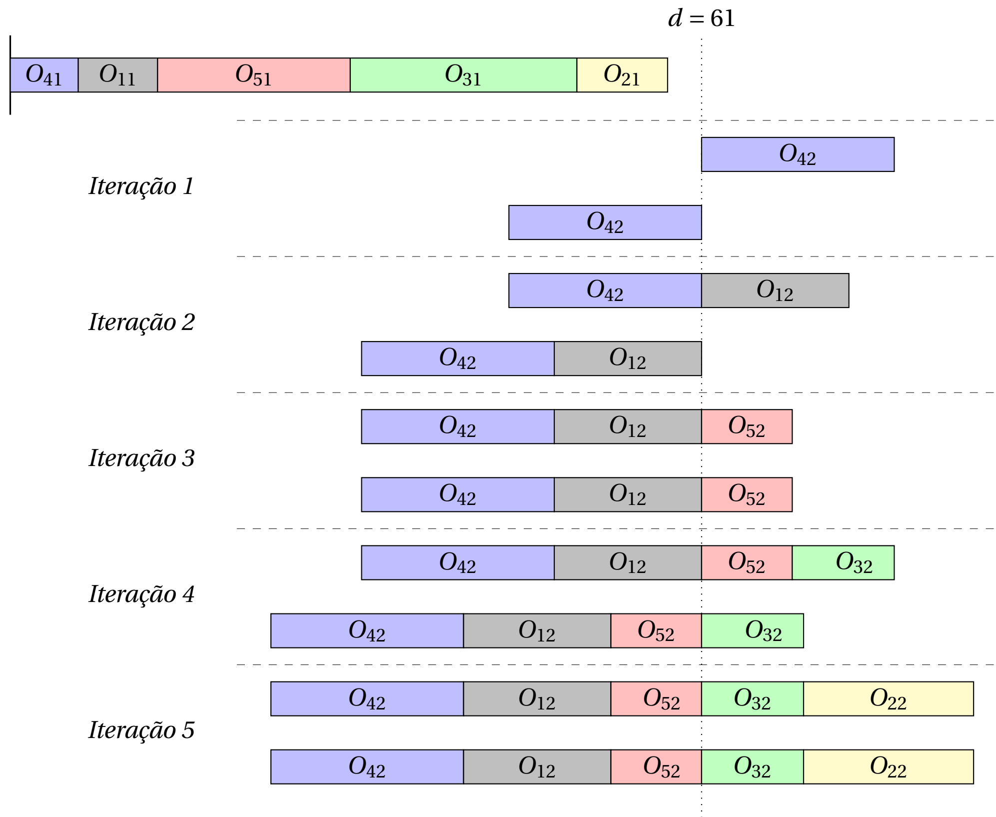


Figura 3.2: *As cinco iterações do algoritmo de* timing *para o exemplo.*

Portanto, dado que os valores p_i, r_i e d são conhecidos *a priori* e que os P_i podem ser calculados a partir dos p_i, a função objetivo depende unicamente da variável S. Então, minimizar $f(\sigma)$ é equivalente a minimizar a função $F(S) = \sum_{i=1}^{q} |S + P_i - d|$, que representa o custo de programar as tarefas do primeiro bloco. Dado que as funções $f_i'(S) = |S + P_i - d|$ são convexas lineares por partes, a soma delas também é convexa linear por partes, garantindo deste modo um mínimo para F. Assim, usando o critério da derivada, não é difícil ver que a função atinge seu valor mínimo em $S = d - P_{h-1}$, onde h é o menor índice tal que $h > q/2 + 1$.

A análise prévia é um caso particular da análise feita por Chrétienne e Sourd (2003) e Sakuraba *et al.* (2009). Os autores aproveitaram as propriedades deste tipo de função (função de custo convexo) para projetar o algoritmo de *timing* para o caso geral (penalidades de adiantamento e atraso diferentes para cada tarefa). Os detalhes podem ser vistos nesses dois trabalhos.

A Figura 3.3 mostra as funções lineares convexas para o exemplo. Neste caso, o bloco é composto por cinco tarefas, com $f_1' = |S + P_1 - 61| = |S - 44|$, $f_2' = |S - 29|$, $f_3' = |S - 21|$, $f_4' = |S - 12|$ e $f_5' = |S + 1|$. A Figura 3.4 mostra a soma das f_i' que também é uma função linear convexa por partes com valor mínimo para $S = 21$.

3.3 Busca local

Uma definição simples para o conceito de busca local é a seguinte: são técnicas baseadas em exploração iterativa de um espaço de soluções; tem-se uma solução inicial a qual tentamos melhorar mediante a exploração iterativa de suas soluções *vizinhas*, as quais são as soluções que estão próximas (segundo alguma métrica) da solução atual. Esse processo é executado enquanto a solução vizinha me-

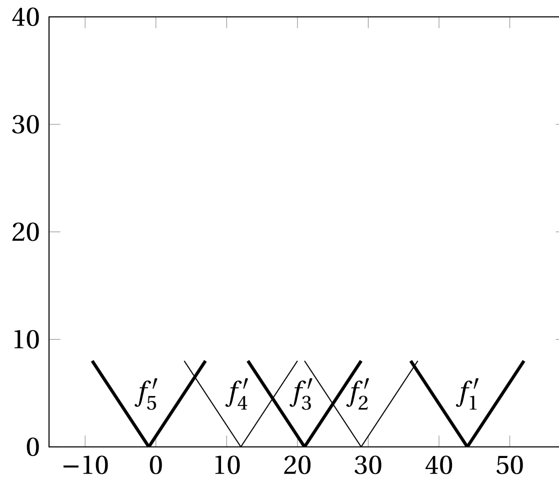


Figura 3.3: *As funções lineares convexas do exemplo.*

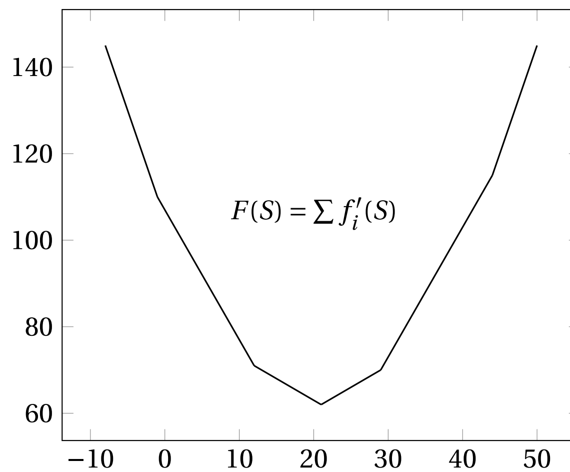


Figura 3.4: *A soma das funções da Figura 3.3.*

lhora o critério que está conduzindo a busca e continua sempre que seja possível encontrar uma solução vizinha que melhore a atual, caso contrario, ele para. A busca local é uma abordagem prática para encontrar soluções "boas" (em tempo razoável) para problemas de otimização combinatória que requerem um tempo proibitivo para obter uma solução exata.

Definição 3.1. Um *problema de otimização combinatória* Π é um problema geral de busca que envolve parâmetros e estruturas discretas, e que é especificado por um conjunto de instâncias. Dizemos que o problema é de *minimização* ou *maximização*, dependendo de qual objetivo é perseguido.

Normalmente, um problema é descrito por meio de uma instância geral e as instâncias comuns são obtidas a partir da instâncias geral, atribuindo valores a seus parâmetros e variáveis.

Definição 3.2. Uma *instância* de um problema de otimização combinatória é um par (S, f) onde S é um conjunto chamado o *espaço solução* e pode ser finito ou infinito numerável. Qualquer $s \in S$ é chamada uma *solução* da instância e a função $f : S \to \mathbb{R}$ é chamada *função custo* do problema. Esta função associa com cada solução s um real $f(s)$ chamado o *custo* da solução, que expressa de algum modo a "qualidade" da solução. O objetivo é encontrar uma solução *ótima global*, isto é, um s^* tal que $\forall s \in S,\ f(s^*) \leqslant f(s)$ (minimização) ou $f(s^*) \geqslant f(s)$ (maximização). Também chamamos o $f^* = f(s^*)$ como o *custo ótimo* da instância.

3.3.1 Vizinhanças

Seja X um conjunto qualquer, denotamos como 2^X seu conjunto potência, definido como $2^X = \{Y \mid Y \subseteq X\}$. No restante do trabalho vamos lidar apenas com problemas de minimização pois um problema de maximização pode ser convertido em um de minimização mediante uma mudança de sinal.

Definição 3.3. Seja (S, f) uma instância de um problema de optimização combinatória. Chamamos *função de vizinhança* a uma função $V : S \to 2^S$. Para todo $s \in S$ dizemos que o conjunto $V(s) \subseteq S$ é uma *vizinhança* de s. Também, dizemos que toda solução $s' \in V(s)$ é *vizinha* de s. O número de elementos do conjunto $V(s)$ é chamado *tamanho* da vizinhança.

Uma "boa" escolha da função de vizinhança é crítica para o desempenho dos métodos de busca local e a estrutura da mesma depende fortemente do problema que estamos tratando. Não é comum ter que definir uma vizinhança enumerando explicitamente seus elementos, senão, definindo pequenas modificações que podem ser aplicadas a uma solução qualquer.

Em um problema de programação de tarefas em ambiente *flow shop* permutacional, uma solução pode ser representada por uma sequência de tarefas

$$s = (J_1, \dots, J_n),$$

que indica a ordem na qual as tarefas vão ser processadas nas máquinas, além do tempo de conclusão em cada máquina de cada uma das tarefas. Com base nesta representação, podemos definir diferentes tipos de vizinhanças dependendo do tipo de manipulação que se aplique à solução atual. Entre as modificações mais comuns se encontram as de troca e inserção, as quais servem para definir dois tipos de funções de vizinhança.

3.3.2 Vizinhança por inserção

Para uma sequência (solução) de tarefas $s = (J_1, \dots, J_n)$, seja $V_{ij}^{\text{INS}}(s)$ a sequência vizinha obtida pela remoção da tarefa na posição i e sua inserção na posição j, isto é,

$$V_{ij}^{\text{INS}}(s) = \begin{cases} (J_1, \dots, J_{i-1}, J_{i+1}, \dots, J_j, J_i, J_{j+1}, \dots, J_n) & \text{se } i < j, \\ (J_1, \dots, J_{j-1}, J_i, J_j, \dots, J_{i-1}, J_{i+1}, \dots, J_n) & \text{se } i > j. \end{cases}$$

Com ajuda de $V_{ij}^{\text{INS}}(s)$, definimos a *vizinhança por inserção* $V_{\text{INS}}(s)$ como

$$V_{\text{INS}}(s) = \{V_{ij}^{\text{INS}}(s) \mid i, j \in \{1, \dots, n\} \land i \neq j\}.$$

Vamos calcular o tamanho desta vizinhança. Se removermos o elemento J_i da sequência temos $n-1$ lugares onde podemos inseri-lo (não pode ser inserido no seu mesmo lugar), portanto temos $n(n-1)$ possíveis sequências. Logo, para cada par de tarefas consecutivas J_i e J_{i+1} obtemos a mesma sequência quando removemos a tarefa J_i e a inserimos depois de J_{i+1} que quando removemos a tarefa J_{i+1} e a inserimos antes de J_i. Como existem $n-1$ pares de tarefas consecutivas, devemos diminuir esta quantidade do número de sequências anterior, o que faz um total de $n(n-1) - (n-1) = (n-1)^2$ sequências para esta vizinhança. Se podemos gerar e avaliar uma sequência vizinha em tempo constante, gerar e avaliar toda a vizinhança vai levar um tempo quadrático.

Algoritmo 2 Melhora iterativa

1: gerar uma solução inicial $s \in S$
2: **repetir**
3: gerar um $s' \in V(s)$
4: **se** $f(s') < f(s)$ **então**
5: $s \leftarrow s'$
6: **fim se**
7: **até** que s seja localmente ótima

3.3.3 Vizinhança por troca

De novo, dada a sequência $s = (J_1, \ldots, J_n)$, seja $V_{ij}^{\text{TR}}(s)$ a sequência vizinha obtida ao trocar as tarefas nas posições i e j,

$$V_{ij}^{\text{TR}}(s) = (J_1, \ldots, J_{i-1}, J_j, J_{i+1}, \ldots, J_{j-1}, J_i, J_{j+1}, \ldots, J_n).$$

De forma semelhante, definimos a *vizinhança por troca* como

$$V_{\text{TR}}(s) = \{V_{ij}^{\text{TR}}(s) \mid i, j \in \{1, \ldots, n\} \wedge i \neq j\}.$$

Calculamos seu tamanho como segue. Primeiro consideramos as trocas de tarefas da forma $(J_i \leftrightarrow J_{i+1})$, isto é, $(J_1 \leftrightarrow J_2), (J_2 \leftrightarrow J_3), \ldots, (J_{n-1} \leftrightarrow J_n)$. No total, temos $n-1$ dessas trocas. Do mesmo modo, as trocas da forma $(J_i \leftrightarrow J_{i+2})$ são $(J_1 \leftrightarrow J_3), (J_2 \leftrightarrow J_4), \ldots, (J_{n-2} \leftrightarrow J_n)$, e fazem um total de $n-2$ trocas possíveis. Continuamos da mesma maneira até considerar as trocas da forma $(J_i \leftrightarrow J_{i+n-1})$, que só tem uma, a saber $(J_1 \leftrightarrow J_n)$. Somando todas as trocas de posição possíveis, temos que o tamanho da vizinhança $V_{\text{TR}}(s)$ é $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2$, que também é uma quantidade quadrática em relação ao número de tarefas.

Como exemplo, as vizinhanças de inserção e troca para a sequência $s = (J_1, J_2, J_3, J_4)$ são:

- $V_{\text{INS}}(s) = \{(J_2, J_1, J_3, J_4), (J_2, J_3, J_1, J_4), (J_2, J_3, J_4, J_1), (J_1, J_3, J_2, J_4), (J_1, J_3, J_4, J_2),$
  $(J_3, J_1, J_2, J_4), (J_1, J_2, J_4, J_3), (J_4, J_1, J_2, J_3), (J_1, J_4, J_2, J_3)\},$
- $V_{\text{TR}}(s) = \{(J_2, J_1, J_3, J_4), (J_1, J_3, J_2, J_4), (J_1, J_2, J_4, J_3), (J_3, J_2, J_1, J_4), (J_1, J_4, J_3, J_2), (J_4, J_2, J_3, J_1)\},$

com tamanhos $|V_{\text{INS}}(s)| = 9$ e $|V_{\text{TR}}(s)| = 6$.

3.4 Algoritmos de melhora iterativa

Uma vez gerada a vizinhança de uma solução (sequência) particular, definimos os esquemas que vão nos permitir escolher a próxima solução (isto é, atualizar a solução). O desempenho da busca vai depender em boa medida de tal escolha.

Definição 3.4. Uma solução $\hat{s} \in S$ é dita *localmente ótima* em relação a uma função de vizinhança V se $f(\hat{s}) \leqslant f(s), \forall s \in V(\hat{s})$.

Um algoritmo de *melhora iterativa* é um procedimento para refinar iterativamente uma solução inicial até que uma solução localmente ótima seja encontrada. O Algoritmo 2 reflete esta maneira de proceder.

Em nosso estudo vamos considerar dois esquemas para a atualização das soluções: maior melhora e primeira melhora, descritos a seguir.

Algoritmo 3 Maior melhora iterativa

1: gerar uma solução inicial $s \in S$
2: **repetir**
3: gerar um $s' \in V(s)$ tal que $f(s') = \min_{s'' \in V(s)} f(s'')$
4: **se** $f(s') < f(s)$ **então**
5: $s \leftarrow s'$
6: **fim se**
7: **até** que s seja localmente ótima

Algoritmo 4 Primeira melhora iterativa

1: gerar uma solução inicial $s \in S$
2: **repetir**
3: gerar um $s' \in V(s)$ segundo um ordenamento predeterminado
4: **se** $f(s') < f(s)$ **então**
5: $s \leftarrow s'$
6: **fim se**
7: **até** que s seja localmente ótima

3.4.1 Maior melhora iterativa

Consiste em explorar a vizinhança procurando a solução que oferece a maior melhora em relação à solução atual. Deve-se observar que nesse esquema examinam-se *todas* as soluções em $V(s)$ e isso pode levar muito tempo se a escolha da função vizinhança não for feita adequadamente. O Algoritmo 3 descreve esse processo.

3.4.2 Primeira melhora iterativa

Este esquema é mais simples devido ao fato de não precisar explorar toda a vizinhança; simplesmente explora $V(s)$ em busca da *primeira* solução que melhore a função custo e prossegue desse modo até encontrar uma solução localmente ótima. No pior dos casos, este esquema pode examinar a mesma quantidade de soluções que o esquema anterior. O Algoritmo 4 explica o processo.

3.4.3 Busca local reduzida

Esta estratégia de busca local foi introduzida por Sakuraba *et al.* (2009) a fim de melhorar a solução obtida em uma etapa de seu algoritmo. Esta busca diz-se "reduzida" no sentido que percorre a sequência de tarefas apenas uma vez. Para cada posição examinada, troca a tarefa nessa posição com a tarefa que vem depois na sequência e que tem o maior ganho possível na função objetivo. O procedimento completo é descrito no Algoritmo 5. Neste trabalho vamos usar essa variação de busca local devido aos bons resultados reportados pelos autores. Cabe ressaltar que a complexidade desta busca é $O(n^3)$ no pior dos casos.

Como exemplo, vamos supor que a sequência de tarefas inicial seja

$$(J_6, J_5, J_2, J_1, J_9, J_7, J_3, J_4, J_8).$$

Então, na primeira iteração do algoritmo é identificada qual das trocas $(J_6 \leftrightarrow J_5), (J_6 \leftrightarrow J_2), (J_6 \leftrightarrow J_1), \ldots, (J_6 \leftrightarrow J_8)$ obtém a maior melhora na função objetivo. Supondo que a maior melhora foi obtida com a troca

Algoritmo 5 Busca local reduzida

Entrada: Uma sequência de tarefas s.
Saída: Uma sequência de tarefas s' localmente ótima.
Passo 1. $i \leftarrow 1$, $MelhorCusto \leftarrow$ custo da sequência inicial (usando o algoritmo de timing).
Passo 2. $MelhorGanho \leftarrow 0$, $j \leftarrow i+1$.
Passo 3. Trocar tarefas nas posições i e j. $NovoGanho \leftarrow MelhorCusto$ menos o custo da nova sequência. Se $NovoGanho > MelhorGanho$ então $w \leftarrow j$ e $MelhorGanho \leftarrow NovoGanho$.
Passo 4. Trocar de novo as tarefas nas posições i e j e $j \leftarrow j+1$. Se $j \leqslant n$, voltar ao Passo 3, senão, ir ao Passo 5.
Passo 5. Se $MelhorGanho > 0$, trocar as tarefas nas posições i e w e fazer $MelhorCusto \leftarrow MelhorCusto - MelhorGanho$. $i \leftarrow i+1$. Se $i < n$, voltar para o Passo 2, senão, o algoritmo termina.

$(J_6 \leftrightarrow J_7)$, então, efetuamos essa troca resultando na sequência

$$(J_7, J_5, J_2, J_1, J_9, J_6, J_3, J_4, J_8).$$

Do mesmo modo, na segunda iteração identificamos qual das trocas $(J_5 \leftrightarrow J_2), (J_5 \leftrightarrow J_1), (J_5 \leftrightarrow J_9), \ldots, (J_5 \leftrightarrow J_8)$ logra a maior melhora na função objetivo e efetuamos essa troca. Supondo de novo que a maior melhora foi obtida com a troca $(J_5 \leftrightarrow J_9)$, a sequência fica

$$(J_7, J_9, J_2, J_1, J_5, J_6, J_3, J_4, J_8).$$

O algoritmo continua do mesmo modo até atingir a penúltima tarefa.

3.5 Método heurístico proposto

O método heurístico proposto neste trabalho é uma extensão para m máquinas das heurísticas desenvolvidas por Sakuraba *et al.* (2009). As heurísticas que eles projetaram geram soluções localmente ótimas baseadas em dois conceitos: (a) um esquema de geração de sequências de tarefas que faz uso de busca local e (b) o algoritmo de timing (Seção 3.2), que obtém uma programação ótima para cada sequência gerada pelo esquema.

A Figura 3.5 mostra o processo completo da abordagem que propomos para a resolução do problema. Na primeira etapa, é construída uma sequência de tarefas inicial mediante a aplicação do método NEH, proposto por Nawaz *et al.* (1983). Esse método foi projetado inicialmente com o objetivo de minimizar o *makespan* no problema *flow shop* clássico, porém, devido ao sucesso mostrado em relação a outros métodos, é usado frequentemente como passo inicial em heurísticas para *flow shops* com funções objetivo distintas do *makespan*, como é o caso do presente trabalho. O Algoritmo 6 apresenta o método NEH, o qual faz uso das regras de sequenciamento da Tabela 3.2 na ordenação inicial das tarefas. A Figura 3.6 mostra um exemplo de execução do método NEH para o caso em que cinco tarefas são ordenadas na sequência $(J_5, J_2, J_4, J_1, J_3)$ por alguma das regras de sequenciamento (em caso de empate, o desempate pode ser feito arbitrariamente). Como primeiro passo, tomamos as tarefas J_5 e J_2, e comparamos qual das sequências parciais (J_2, J_5), (J_5, J_2) logra a menor programação (aplicando o algoritmo de timing a cada uma delas). Neste caso, ganha a sequência (J_5, J_2) e em seguida a tarefa J_4 é inserida nas 3 posições possíveis em busca da melhor sequência parcial. O método continua do mesmo modo até que as n tarefas sejam inseridas na sequência. A sequência final obtida pelo método NEH é neste

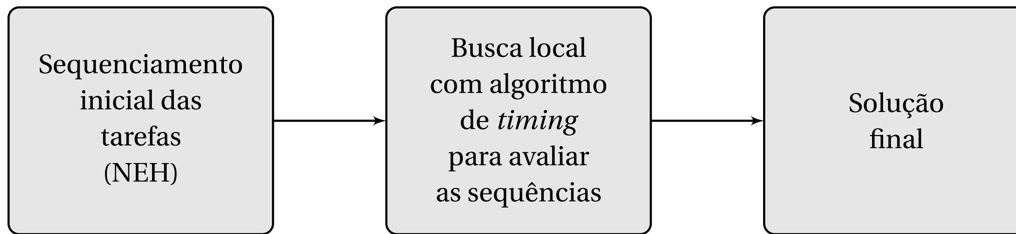


Figura 3.5: *Estrutura de trabalho das heurísticas propostas.*

Algoritmo 6 Algoritmo NEH

Saída: Uma "boa" sequência de tarefas.
1: Ordenar as tarefas por qualquer uma das regras de sequenciamento da Tabela 3.2.
2: Escolher as primeiras duas tarefas e aplicar o algoritmo de *timing* às duas possíveis sequências de tarefas (como se fosse apenas duas tarefas) e salvar a melhor delas.
3: **para** $k \leftarrow 3$ **até** n **faça**
4: Inserir a k-ésima tarefa na posição que minimize a função objetivo parcial dentre as k sequências possíveis.
5: **fim para**

---

caso $(J_5, J_3, J_4, J_1, J_2)$.

Na segunda etapa da abordagem (etapa de busca local), vamos considerar os seguintes esquemas de busca:

[M] Maior melhora iterativa (Seção 3.4.1).

[P] Primeira melhora iterativa (Seção 3.4.2).

[R] Busca local reduzida (Seção 3.4.3).

O processo de gerar soluções vizinhas a partir da solução atual vai ser realizado usando os conceitos de:

[I] Vizinhança por inserção (Seção 3.3.2).

[T] Vizinhança por troca (Seção 3.3.3).

Portanto, definiremos um conjunto de heurísticas completas do seguinte modo. Seja SHE_{pqr} a heurística que consiste em usar:

- o esquema de busca local p,
- o conceito q para a geração de soluções vizinhas e
- a regra de sequenciamento r da Tabela 3.2 para a ordenação inicial das tarefas (no início do NEH).

Por exemplo, SHE_{PT6} é a heurística que no processo de busca local pula à primeira solução que melhora a função objetivo, gera sua vizinhança pelo conceito de troca e usa a regra de sequenciamento 6.

Para o caso em que usamos o esquema de busca local reduzida, nomeamos as heurísticas como SHE_{Rr}, onde r é definida como acima.

Com essa nomenclatura para as heurísticas temos várias opções a examinar, todas elas fazendo uso do algoritmo de timing. O modo em que usamos o algoritmo é o seguinte: uma vez que tenhamos gerado uma sequência de tarefas em algum instante na execução das heurísticas acima mencionadas, o passo seguinte será executar as tarefas nas máquinas $M_1, \ldots, M_{m-1}$ o mais rápido possível. Deste modo,

Regra	Descrição
1	TPL(Tempo de Processamento mais Longo) usando tempos de processamento na primeira máquina
2	TPL usando tempos de processamento na última máquina
3	TPL usando a soma de tempos de processamento em todas as máquinas
4	TPC(Tempo de Processamento mais Curto) usando tempos de processamento na primeira máquina
5	TPC usando tempos de processamento na última máquina
6	TPC usando a soma de tempos de processamento em todas as máquinas

Tabela 3.2: *Descrição das regras de sequenciamento para o procedimento NEH.*

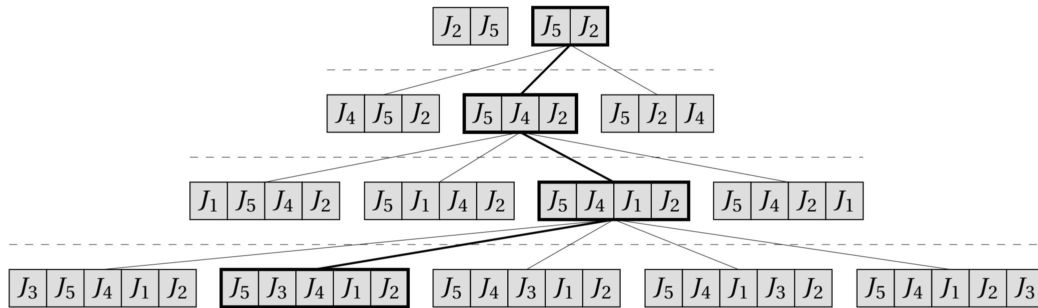


Figura 3.6: *Execução do método NEH para as tarefas na ordem* $(J_5, J_2, J_4, J_1, J_3)$.

o tempo de conclusão de uma tarefa na máquina M_{m-1} (a variável $c_{i,m-1}$) será visualizado como o instante de liberação r_i da tarefa para seu ingresso na máquina M_m.

Assim, para uma sequência qualquer, o Algoritmo 7 calcula os r_i da seguinte forma: cria-se um vetor auxiliar que vai conter os tempos de conclusão de todas as operações de uma tarefa por vez. Depois, em cada iteração, calculam-se os tempos de conclusão das operações da tarefa a seguir. Cada operação inicia seu processamento no maior instante entre o tempo de conclusão da tarefa anterior (na mesma máquina) e o tempo de conclusão da mesma tarefa na máquina anterior. Para a primeira tarefa da sequência, o instante de conclusão de cada operação é calculado simplesmente como a soma cumulativa dos tempos de processamento em cada máquina.

Deste modo, podemos aplicar o algoritmo de timing (Algoritmo 1) com entrada os r_i calculados previamente, além de os tempos de processamento na última máquina, e obter uma programação para a sequência de tarefas *dada* (que sería uma programação ótima se $m \leqslant 2$—ver Sakuraba *et al.* 2009). O processo continua com o esquema de busca local escolhido até atingir uma sequência localmente ótima.

3.6 Minimização da espera no *buffer*

Como uma consequência da definição da função objetivo (Equação 2.3), o custo de uma programação depende unicamente das variáveis E_i e T_i, que por sua vez dependem apenas do tempo de conclusão das tarefas na última máquina (C_i) e a data de entrega. Isso quer dizer que poderiam existir duas ou mais soluções ótimas cujos tempos de conclusão na última máquina sejam idênticos, porém diferindo na forma em que as operações são completadas nas máquinas anteriores.

Como um exemplo deste fato, a Tabela 3.3 mostra os tempos de processamento p_{ij} para um problema sugerido por Chandra *et al.* (2009). O problema consiste de cinco tarefas, cinco máquinas e data de entrega $d = 77$. A Figura 3.7 mostra duas soluções ótimas distintas que têm em comum a programação na última máquina. Também, é possível notar da Figura 3.7(a) que na segunda máquina a tarefa J_5

Algoritmo 7 Calcula os instantes de liberação (release dates) para a última máquina

Entrada: n, m e p (matriz com os tempos de processamento do problema).
Saída: r (vetor com os instantes de liberação r_i para a última máquina).

```
1:  Cria os vetores c e r
2:  c_1 ← p_{1,1}
3:  para i ← 2 até m − 1 faça
4:      c_i ← c_{i−1} + p_{1,i}
5:  fim para
6:  r_1 ← c_{m−1}
7:  para j ← 2 até n faça
8:      c_1 ← c_1 + p_{j,1}
9:      para 2 ← m − 1 até 1 faça
10:         se c_{i−1} ⩾ c_i então
11:             c_i ← c_{i−1} + p_{j,i}
12:         senão
13:             c_i ← c_i + p_{j,i}
14:         fim se
15:         r_j ← c_{m−1}
16:     fim para
17: fim para
```

Tarefas	Máquinas				
	M_1	M_2	M_3	M_4	M_5
J_1	10	8	1	16	6
J_2	20	15	18	8	14
J_3	15	10	14	8	14
J_4	17	10	14	20	8
J_5	11	8	9	5	17

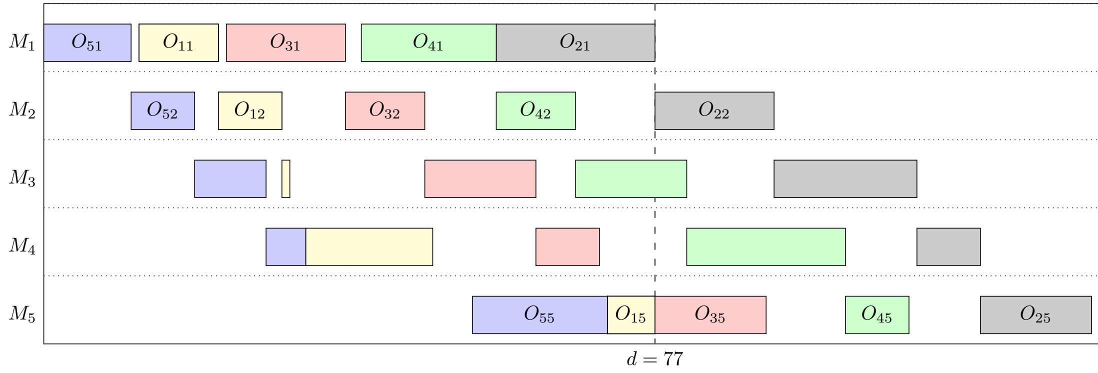
Tabela 3.3: *Tempos de processamento para um* flow shop *com 5 tarefas e 5 máquinas.*

(operação O_{52}) é processada um pouco antes em relação à Figura 3.7(b), sem afetar o valor da função objetivo.

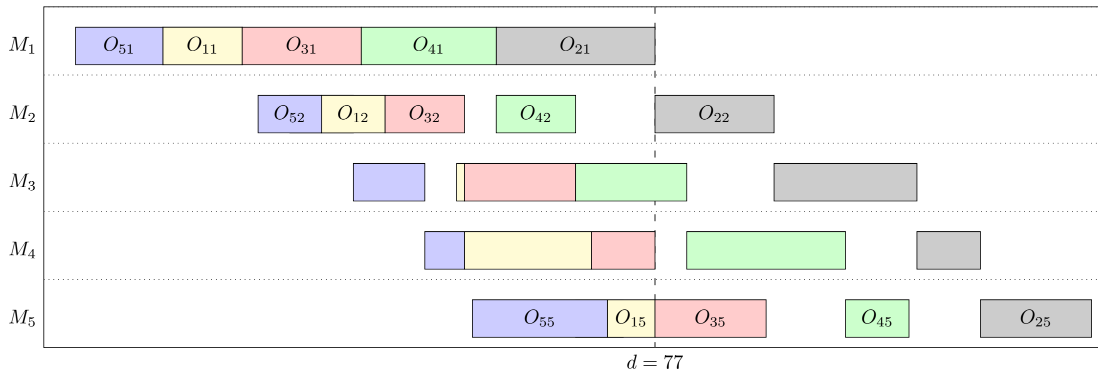
Isso significa que poderíamos “mover” (adiantar ou atrasar) arbitrariamente o início das operações em qualquer das máquinas anteriores à última sem afetar a otimalidade da solução (e respeitando as restrições do problema). No entanto, começar (e portanto, concluir) com antecipação uma operação (em uma máquina diferente da última) que poderia ser deixada para depois resulta em um maior tempo de permanência da tarefa no *buffer*, o que significa um maior gasto devido ao custo envolvido em armazenar as tarefas até que a próxima máquina seja liberada. Portanto, dentre duas soluções ótimas, deveríamos preferir a que diminua o tempo de espera no *buffer*, isto é, a que adia o processamento das tarefas nas máquinas $M_1, M_2, \ldots, M_{m-1}$ tanto quanto possível.

O Algoritmo 8 toma como entrada uma matriz TC que contém os tempos de conclusão de todas as operações de uma solução ótima e modifica esses valores de tal forma que os novos tempos de conclusão das operações nas máquinas $M_1, \ldots M_{m-1}$ seguem o critério descrito no parágrafo anterior.

A Figura 3.7(b) mostra o resultado de aplicar o Algoritmo 8 à programação da Figura 3.7(a). Este algoritmo é descrito brevemente assim: as linhas 2–4 copiam o vetor C para a última coluna da matriz TC; as linhas 5–14 primeiro estabelecem o tempo de conclusão da tarefa na posição n na máquina M_j a ser igual ao tempo de início da mesma tarefa na máquina M_{j+1}. Depois, é estabelecido o valor do tempo



(a) *Uma programação com TII inseridos.*



(b) *Uma programação com tempo mínimo de espera no buffer.*

Figura 3.7: *Duas programações ótimas para o problema* $F5|\,prmu, d_i = 73|\sum_{j=1}^{5}(E_j + T_j)$.

de conclusão da tarefa na posição $n-1$ na máquina M_j a ser o maior instante entre o tempo de início da mesma tarefa na máquina M_{j+1} e o tempo de início da seguinte tarefa na máquina anterior.

Então, este critério de "economia" de tempo na permanência das tarefas no *buffer* poderia (e deveria) influenciar nossa busca de soluções ótimas ao problema pois, deste modo, teríamos um equilíbrio entre soluções com adiantamento e atraso total baixo e custo de armazenamento das tarefas no *buffer* também baixo. Isso implica que devemos reformular o problema (e portanto, a formulação matemática) para ter em conta esta característica. Tanto quanto é do nosso conhecimento, este problema ainda não foi estudado na literatura e poderíamos considerá-lo como um trabalho futuro.

3.7 Experimentos numéricos

Como uma primeira fase na avaliação do método proposto, foram implementadas cinco heurísticas: SHE_{R6}, SHE_{PT6}, SHE_{PI6}, SHE_{MT6} e SHE_{MI6}. Usamos como dados de teste o conjunto completo de instâncias de Chandra *et al.* (2009), disponível no endereço eletrônico dos autores (http://home.iitk.ac.in/~pmehta/flowshop.htm) e também no endereço http://www.ime.usp.br/~jdelgado/flowshop/chandraFlowshop. Os tamanhos das instâncias estudadas por Chandra *et al.* foram:

$$(n,m) \in \left\{ \begin{array}{l} (5,\ 5\ ), (10,\ 5\ ), (20,\ 5\ ), (50,\ 5\ ), (80,\ 5\ ), (100,\ 5\ ), \\ (5,10), (10,10), (20,10), (50,10), (80,10), (100,10), \\ (5,15), (10,15), (20,15), (50,15), (80,15), (100,15), \\ (5,20), (10,20), (20,20), (50,20), (80,20), (100,20) \end{array} \right\}. \quad (3.3)$$

Algoritmo 8 Aderir à data de entrega

Entrada: n, m, C (vetor com os tempos de conclusão C_i de uma programação ótima) e P (matriz com os tempos de processamento do problema).

Saída: TC (matriz com os tempos de conclusão de todas as operações conforme ao critério estabelecido).

1: Cria matriz TC
2: **para** $i \leftarrow 1$ **até** n **faça**
3: $\quad TC_{i,m} \leftarrow C_i$
4: **fim para**
5: **para** $j \leftarrow m-1$ **até** 1 **faça**
6: $\quad TC_{n,j} \leftarrow TC_{n,j+1} - P_{n,j+1}$
7: $\quad$ **para** $i \leftarrow n-1$ **até** 1 **faça**
8: $\quad\quad$ **se** $TC_{i+1,j} - P_{i+1,j} \leqslant TC_{i,j+1} - P_{i,j+1}$ **então**
9: $\quad\quad\quad TC_{ij} \leftarrow TC_{i+1,j} - P_{i+1,j}$
10: $\quad\quad$ **senão**
11: $\quad\quad\quad TC_{ij} \leftarrow TC_{i,j+1} - P_{i,j+1}$
12: $\quad\quad$ **fim se**
13: $\quad$ **fim para**
14: **fim para**

Para cada par (n, m) foram dadas 50 instâncias do problema, com três datas de entrega restritivas por instância. Chandra *et al.* fornecem o valor objetivo *ótimo* unicamente para a data menos restritiva, porém, mostram valores obtidos pelo seu método heurístico para todas as três. Os valores objetivo para as instâncias com valores ótimos fornecidos coincidem, em todas as instâncias, com os valores ótimos obtidos pela nossa formulação matemática, "validando" deste modo o modelo PLIM descrito na Seção 2.3.

Todos os procedimentos apresentados neste capítulo foram implementados na linguagem C++ e compilados com o compilador GNU g++ versão 4.8.2 com a diretiva de otimização -O3 ligada. Os experimentos foram realizados em uma máquina Intel(R) Xeon(R) CPU X5650 2.67 Ghz com 8 GB de RAM e sistema operacional Ubuntu 14.04 (GNU/Linux 3.13.0-45-generic x86_64).

A Tabela 3.4 mostra os resultados numéricos da execução da heurística SHE_{R6} nas 50 instâncias de tamanho $n = 100$ e $m = 20$, com data de entrega menos restritiva. Como para esta data de entrega dispomos dos valores ótimos das instâncias, calculamos o *Gap* de cada heurística com a fórmula

$$\text{Gap} = \frac{V_h - V_o}{V_o} \times 100\%,$$

onde V_h é o valor obtido pela heurística avaliada e V_o o valor ótimo fornecido por Chandra *et al.* Pode-se observar que virtualmente em toda entrada da tabela o Gap da heurística SHE_{R6} é notavelmente melhor que o Gap obtido pelo método proposto por Chandra *et al.* (2009), e o Gap médio para este conjunto de instâncias é 0.111%. Também é deixado em negrito as entradas em que a nossa heurística ganha e as entradas marcadas com "–" indicam que o valor ótimo foi atingido. Por último, o tempo médio de execução em segundos para esta heurística foi 0.141.

Do mesmo modo, as Tabelas 3.5, 3.6, 3.7 e 3.8 mostram os resultados para as heurísticas SHE_{PT6}, SHE_{PI6}, SHE_{MT6} e SHE_{MI6}, respectivamente. Todas as heurísticas mostram melhorias em relação a heurística projetada por Chandra *et al.* Dentre elas, a que mostra os melhores resultados é a heurística SHE_{MT6} com um bom Gap médio de 0.004%, tempo médio de execução de 6.27 segundos (maior do que os outros) e 17 instâncias em que atinge o valor ótimo global.

As tabelas contendo os resultados da aplicação das cinco heurísticas para a totalidade de instâncias testadas por Chandra *et al.* estão disponíveis no endereço http://www.ime.usp.br/~jdelgado/flowshop/resultados. A Tabela 3.9 mostra o resumo dos Gap médios de todos os tamanhos de instância com a finalidade de avaliar o desempenho das heurísticas em relação ao número de tarefas no *flow shop*. Observa-se que quando o número de tarefas é pequeno ($n \leqslant 20$) o método de Chandra *et al.* obtém melhores valores do Gap médio (valores em negrito), mas quando o número de tarefas cresce ($n \geqslant 50$) nossas heurísticas atingem melhores resultados (muito próximo do ótimo no caso de SHE_{MT6}), sugerindo um melhor comportamento assintótico que deve ser testado com mais profundidade.

Devido à falta de um conjunto maior de dados de referência padrão (*benchmark*) para testar as heurísticas (e com a finalidade de estuda-las com mais precisão), o comportamento de *todas* as heurísticas propostas neste trabalho foi analisado com um novo conjunto de instâncias de teste. Este novo conjunto de dados foi criado mediante o processo sugerido por Taillard (1993) para o problema *flow shop* clássico. Taillard apresenta um procedimento baseado em um *gerador congruencial linear* (GCL) (Bratley *et al.*, 2011) que produz inteiros pseudoaleatórios entre 1 e 99 (de acordo com uma distribuição uniforme discreta), para usar eles como tempos de processamento das tarefas. O GCL usado por ele está baseado na fórmula de recorrência

$$X_{j+1} = (16807X_j) \mod (2^{31} - 1)$$

e produz iterativamente um conjunto de números $X_j \in [1, 99]$ a partir de uma "semente" inicial s (Algoritmo 9).

Algoritmo 9 Gera um número pseudoaleatório

Entrada: s (a semente), α e β (os extremos do intervalo $[\alpha, \beta]$).
Saída: Atualiza o valor da semente s e devolve um número entre 1 e 99 (inclusive).

```
 1: função INT-UNI(s, α, β)
 2:     m ← 2^31 − 1
 3:     a ← 7^5
 4:     b ← 127773
 5:     c ← 2836
 6:     k ← ⌊s/b⌋
 7:     s ← a ∗ (s  mod b) − k ∗ c
 8:     se s < 0 então
 9:         s ← s + m
10:     fim se
11:     devolve α + ⌊(s/m) ∗ (β − α + 1)⌋
12: fim função
```

Esses números são gerados sem importar o computador usado, variando apenas em função da semente. Para cada tamanho de instância (isto é, para cada par (n, m)) usamos 10 sementes, todas elas fornecidas no trabalho de Taillard. Portanto, com ajuda do Algoritmo 10 e as 10 sementes, geramos 10 instâncias para cada tamanho de instância.

Os tamanhos de instância tomados de Taillard que vamos examinar são:

$$(n, m) \in \{(100, 5), (100, 10), (100, 20), (200, 10), (200, 20), (500, 20)\}.$$

Para cada instância consideramos quatro datas de entrega restritivas, o que perfaz um total de $10 \times 6 \times 4 = 240$ instâncias de teste. Deste modo, estamos propondo um novo *benchmark*.

Algoritmo 10 Gera tempos de processamento para as instâncias de teste

Entrada: n, m, s_0 (semente inicial).
Saída: P (matriz com os tempos de processamento de uma instância de tamanho $n \times m$).

1: $s \leftarrow s_0$
2: **para** $i \leftarrow n$ **até** 1 **faça**
3:     **para** $j \leftarrow m$ **até** 1 **faça**
4:         $P_{ij} \leftarrow$ INT-UNI(s, 1, 99)
5:     **fim para**
6: **fim para**

Para a geração das datas de entrega restritivas, primeiro calculamos uma cota superior d_s para as datas de entrega restritivas do modo seguinte: aplicamos o procedimento de Bagchi *et al.* (1986) com uma data de entrega não restritiva (pode ser a soma dos tempos de processamento das n tarefas nas m máquinas) para obter uma sequência de tarefas ótima, em seguida calculamos para essa sequência os instantes de liberação para a última máquina (Algoritmo 7) e depois “empurramos” todas as tarefas na última máquina (como um bloco único) para começar seu processamento o mais rápido possível até que alguma tarefa não possa ser mais atrasada (devido a seu instante de liberação); finalmente obtemos d_s como o tempo de conclusão da tarefa na posição $\lceil n/2 \rceil$ da sequência. O procedimento completo é apresentado no Apêndice B.

Agora, seja J_t a tarefa com o menor tempo de processamento total, isto é,

$$J_t = \arg\min_{J_i} \sum_{j=1}^{m} p_{ij}, \quad \forall\, i \in \{1, 2, \ldots, n\},$$

e definimos d_t como

$$d_t = \sum_{j=1}^{m} p_{tj}.$$

Então, não é difícil perceber que para datas de entrega menores que d_t necessariamente todas as tarefas vão estar atrasadas e, deste modo, o problema torna-se em um de minimizar o tempo de fluxo médio no *flow shop*. Portanto, como nosso objetivo é estudar o caso em que se têm tanto tarefas adiantadas como atrasadas, vamos restringir a escolha das datas de entrega ao intervalo (d_t, d_s).

Assim, para cada instância vamos trabalhar com as seguintes datas de entrega:

$$d_i = d_t + \lfloor 0.2i(d_s - d_t) \rfloor, \quad \forall\, i \in \{1, 2, 3, 4\}.$$

Finalmente, devido à dificuldade de obter soluções ótimas para este novo conjunto de instâncias (o tempo de execução no CPLEX torna-se proibitivo), não será possível medir o Gap para os resultados obtidos pelas heurísticas. Portanto, usamos a medida de desempenho IDR (Índice de Desvio Relativo) como sugerido por Kim (1993). O IDR é calculado com a fórmula seguinte:

$$IDR = \frac{VH - MV}{PV - MV}, \tag{3.4}$$

onde VH é o valor obtido pela heurística sendo avaliada e MV e PV sendo o melhor e pior valor obtido, respectivamente, entre todas as heurísticas avaliadas. Em vista disso, o índice tem um valor entre 0 e 1 e um valor mais próximo de 0 indica um melhor desempenho da heurística testada.

A Tabela 3.10 mostra os resultados de aplicar a heurística SHE_{R6} para as 10 instâncias de tamanho

$(200, 10)$ e as 4 datas de entrega restritivas, cujos valores podem ser comparados com os da Tabela 3.11, a qual contém os valores que resultaram de aplicar a heurística SHE_{MT6} para as mesmas 10 instâncias. Desses experimentos podemos concluir que a heurística SHE_{MT6} obtém os melhores resultados mas o tempo de cómputo das soluções aumenta significativamente. A totalidade de tabelas com os resultado dos experimentos se encuentram no endereço http://www.ime.usp.br/~jdelgado/flowshop/resultados/Taillard. A Tabela 3.12 é um resumo do IDR médio para as 5 heurísticas usadas, todos os tamanhos de instância e as quatro datas de entrega. Da tabela se observa que as heurísticas SHE_{MT6} e SHE_{MI6} obtém melhor desempenho em todos os casos e, seguido da heurística SHE_{R6} que obtem também bom desempenho com tempo de cómputo menor.

Instância	d	Tempo (s)	Valor objetivo			Gap(%)	
			Ótimo	Chandra *et al.*	SHE_{R6}	Chandra *et al.*	SHE_{R6}
1	5748	0.139780	81607	82606	81727	1.224	**0.147**
2	5919	0.145986	79799	80398	80055	0.751	**0.321**
3	6024	0.144696	81331	81954	81336	0.766	**0.006**
4	5781	0.135951	81869	82683	81881	0.994	**0.015**
5	5677	0.142962	87412	88749	87471	1.530	**0.067**
6	5594	0.144120	84388	85337	84426	1.125	**0.045**
7	5857	0.139259	80960	82054	80967	1.351	**0.009**
8	5940	0.134307	72794	73417	72804	0.856	**0.014**
9	5546	0.139421	94768	95636	94796	0.916	**0.030**
10	5615	0.144574	92031	93131	92082	1.195	**0.055**
11	5655	0.141649	85911	87024	86005	1.296	**0.109**
12	5796	0.138348	74947	75696	74999	0.999	**0.069**
13	5603	0.149570	87074	87921	87109	0.973	**0.040**
14	5727	0.141366	75524	76737	75551	1.606	**0.036**
15	5355	0.145917	100343	101876	100447	1.528	**0.104**
16	6036	0.141419	76536	77017	76547	0.628	**0.014**
17	5975	0.141583	73436	73974	73448	0.733	**0.016**
18	5748	0.135200	73506	74144	74197	0.868	0.940
19	5541	0.142406	81305	82544	81521	1.524	**0.266**
20	5652	0.142648	87060	88126	87096	1.224	**0.041**
21	5574	0.138446	92117	93545	92180	1.550	**0.068**
22	5828	0.144760	85555	86199	85556	0.753	**0.001**
23	5924	0.143764	85558	86151	85563	0.693	**0.006**
24	5636	0.140210	74766	76946	75180	2.916	**0.554**
25	5742	0.141401	95856	96499	95896	0.671	**0.042**
26	5705	0.143517	89818	91056	89836	1.378	**0.020**
27	5596	0.144852	86696	87864	86710	1.347	**0.016**
28	5345	0.137927	90470	92354	90804	2.082	**0.369**
29	5933	0.135909	80072	80476	80183	0.505	**0.139**
30	5694	0.141084	84056	84764	84058	0.842	**0.002**
31	5471	0.141178	80372	81852	80755	1.841	**0.477**
32	5791	0.137424	91754	92750	91766	1.086	**0.013**
33	5729	0.139995	93117	94285	93176	1.254	**0.063**
34	5879	0.141723	82934	83759	82947	0.995	**0.016**
35	5624	0.141793	77692	78638	77699	1.218	**0.009**
36	5419	0.137144	99851	101111	99854	1.262	**0.003**
37	5922	0.140446	78184	78723	78192	0.689	**0.010**
38	5852	0.139265	82545	83222	82554	0.820	**0.011**
39	5903	0.140457	74000	74515	74055	0.696	**0.074**
40	5793	0.141583	70392	70954	70953	0.798	**0.797**
41	5910	0.142154	88242	88986	88250	0.843	**0.009**
42	5737	0.139450	83151	84266	83167	1.341	**0.019**
43	5450	0.140608	92279	93526	92289	1.351	**0.011**
44	5968	0.140560	72504	73307	72699	1.108	**0.269**
45	5882	0.138945	92359	92842	92362	0.523	**0.003**
46	5527	0.140432	86767	88088	86782	1.522	**0.017**
47	5493	0.135775	82972	84524	83081	1.871	**0.131**
48	5742	0.141562	87852	88392	87886	0.615	**0.039**
49	5743	0.139764	91759	92358	91779	0.653	**0.022**
50	6161	0.137280	77519	78064	77519	0.703	–
média		0.140811				1.120	0.111

Tabela 3.4: *Resultado numérico da heurística SHE_{R6} para as 50 instâncias de Chandra et al. com $n = 100$ e $m = 20$.*

Instância	d	Tempo (s)	Valor objetivo			Gap(%)	
			Ótimo	Chandra *et al.*	SHE_{PT6}	Chandra *et al.*	SHE_{PT6}
1	5748	0.137149	81607	82606	82268	1.224	**0.810**
2	5919	0.134879	79799	80398	80244	0.751	**0.558**
3	6024	0.135922	81331	81954	81374	0.766	**0.053**
4	5781	0.137383	81869	82683	81911	0.994	**0.051**
5	5677	0.136954	87412	88749	87750	1.530	**0.387**
6	5594	0.139575	84388	85337	84559	1.125	**0.203**
7	5857	0.134984	80960	82054	81064	1.351	**0.128**
8	5940	0.133474	72794	73417	72867	0.856	**0.100**
9	5546	0.132542	94768	95636	94983	0.916	**0.227**
10	5615	0.135009	92031	93131	92486	1.195	**0.494**
11	5655	0.167936	85911	87024	86758	1.296	**0.986**
12	5796	0.134881	74947	75696	75353	0.999	**0.542**
13	5603	0.143363	87074	87921	87297	0.973	**0.256**
14	5727	0.135115	75524	76737	75743	1.606	**0.290**
15	5355	0.144490	100343	101876	100855	1.528	**0.510**
16	6036	0.136784	76536	77017	76770	0.628	**0.306**
17	5975	0.136515	73436	73974	73544	0.733	**0.147**
18	5748	0.133834	73506	74144	74606	0.868	1.496
19	5541	0.133766	81305	82544	81751	1.524	**0.549**
20	5652	0.136059	87060	88126	87472	1.224	**0.473**
21	5574	0.133191	92117	93545	92641	1.550	**0.569**
22	5828	0.135026	85555	86199	85597	0.753	**0.049**
23	5924	0.136676	85558	86151	85615	0.693	**0.067**
24	5636	0.134955	74766	76946	75624	2.916	**1.148**
25	5742	0.134505	95856	96499	96060	0.671	**0.213**
26	5705	0.134342	89818	91056	90021	1.378	**0.226**
27	5596	0.137181	86696	87864	86839	1.347	**0.165**
28	5345	0.166236	90470	92354	91400	2.082	**1.028**
29	5933	0.133517	80072	80476	80711	0.505	0.798
30	5694	0.137322	84056	84764	84109	0.842	**0.063**
31	5471	0.135057	80372	81852	82150	1.841	2.212
32	5791	0.134159	91754	92750	92054	1.086	**0.327**
33	5729	0.135808	93117	94285	93412	1.254	**0.317**
34	5879	0.138992	82934	83759	82971	0.995	**0.045**
35	5624	0.134471	77692	78638	77722	1.218	**0.039**
36	5419	0.139626	99851	101111	99880	1.262	**0.029**
37	5922	0.134822	78184	78723	78252	0.689	**0.087**
38	5852	0.140347	82545	83222	82595	0.820	**0.061**
39	5903	0.134385	74000	74515	74123	0.696	**0.166**
40	5793	0.135782	70392	70954	71971	0.798	2.243
41	5910	0.141105	88242	88986	88293	0.843	**0.058**
42	5737	0.135148	83151	84266	83311	1.341	**0.192**
43	5450	0.134791	92279	93526	92382	1.351	**0.112**
44	5968	0.133716	72504	73307	73091	1.108	**0.810**
45	5882	0.136262	92359	92842	92415	0.523	**0.061**
46	5527	0.135185	86767	88088	86878	1.522	**0.128**
47	5493	0.134319	82972	84524	83900	1.871	**1.118**
48	5742	0.135320	87852	88392	87958	0.615	**0.121**
49	5743	0.136343	91759	92358	91969	0.653	**0.229**
50	6161	0.140348	77519	78064	77555	0.703	**0.046**
média		0.137391				1.120	0.426

Tabela 3.5: *Resultado numérico da heurística SHE_{PT6} para as 50 instâncias de Chandra et al. com $n = 100$ e $m = 20$.*

			Valor objetivo			Gap(%)	
Instância	d	Tempo (s)	Ótimo	Chandra *et al.*	SHE_{PI6}	Chandra *et al.*	SHE_{PI6}
1	5748	0.195122	81607	82606	82268	1.224	**0.810**
2	5919	0.194466	79799	80398	80244	0.751	**0.558**
3	6024	0.193339	81331	81954	81374	0.766	**0.053**
4	5781	0.191549	81869	82683	81911	0.994	**0.051**
5	5677	0.196922	87412	88749	87750	1.530	**0.387**
6	5594	0.197005	84388	85337	84559	1.125	**0.203**
7	5857	0.206767	80960	82054	81064	1.351	**0.128**
8	5940	0.200757	72794	73417	72867	0.856	**0.100**
9	5546	0.196030	94768	95636	94983	0.916	**0.227**
10	5615	0.196448	92031	93131	92486	1.195	**0.494**
11	5655	0.270255	85911	87024	86722	1.296	**0.944**
12	5796	0.197844	74947	75696	75353	0.999	**0.542**
13	5603	0.199466	87074	87921	87297	0.973	**0.256**
14	5727	0.196560	75524	76737	75743	1.606	**0.290**
15	5355	0.197588	100343	101876	100855	1.528	**0.510**
16	6036	0.199294	76536	77017	76770	0.628	**0.306**
17	5975	0.195469	73436	73974	73544	0.733	**0.147**
18	5748	0.191380	73506	74144	74606	0.868	1.496
19	5541	0.194837	81305	82544	81751	1.524	**0.549**
20	5652	0.196114	87060	88126	87472	1.224	**0.473**
21	5574	0.194349	92117	93545	92641	1.550	**0.569**
22	5828	0.196561	85555	86199	85597	0.753	**0.049**
23	5924	0.197453	85558	86151	85615	0.693	**0.067**
24	5636	0.197009	74766	76946	75624	2.916	**1.148**
25	5742	0.194736	95856	96499	96060	0.671	**0.213**
26	5705	0.195138	89818	91056	90021	1.378	**0.226**
27	5596	0.197015	86696	87864	86839	1.347	**0.165**
28	5345	0.266119	90470	92354	92173	2.082	**1.882**
29	5933	0.198164	80072	80476	80711	0.505	0.798
30	5694	0.198661	84056	84764	84109	0.842	**0.063**
31	5471	0.195470	80372	81852	82150	1.841	2.212
32	5791	0.192995	91754	92750	92054	1.086	**0.327**
33	5729	0.194711	93117	94285	93412	1.254	**0.317**
34	5879	0.194205	82934	83759	82971	0.995	**0.045**
35	5624	0.195708	77692	78638	77722	1.218	**0.039**
36	5419	0.195537	99851	101111	99880	1.262	**0.029**
37	5922	0.194833	78184	78723	78252	0.689	**0.087**
38	5852	0.195028	82545	83222	82595	0.820	**0.061**
39	5903	0.202998	74000	74515	74123	0.696	**0.166**
40	5793	0.203376	70392	70954	71971	0.798	2.243
41	5910	0.200226	88242	88986	88293	0.843	**0.058**
42	5737	0.199706	83151	84266	83311	1.341	**0.192**
43	5450	0.197069	92279	93526	92382	1.351	**0.112**
44	5968	0.191730	72504	73307	73091	1.108	**0.810**
45	5882	0.195218	92359	92842	92415	0.523	**0.061**
46	5527	0.194536	86767	88088	86878	1.522	**0.128**
47	5493	0.195150	82972	84524	83900	1.871	**1.118**
48	5742	0.202836	87852	88392	87958	0.615	**0.121**
49	5743	0.193434	91759	92358	91969	0.653	**0.229**
50	6161	0.192668	77519	78064	77555	0.703	**0.046**
média		0.199397				1.120	0.442

Tabela 3.6: *Resultado numérico da heurística SHE_{PI6} para as 50 instâncias de Chandra et al. com $n = 100$ e $m = 20$.*

			Valor objetivo			Gap(%)	
Instância	d	Tempo (s)	Ótimo	Chandra *et al.*	SHE_{MT6}	Chandra *et al.*	SHE_{MT6}
1	5748	9.697066	81607	82606	81607	1.224	–
2	5919	7.827184	79799	80398	79802	0.751	**0.004**
3	6024	3.129453	81331	81954	81331	0.766	–
4	5781	2.407303	81869	82683	81872	0.994	**0.004**
5	5677	8.190974	87412	88749	87413	1.530	**0.001**
6	5594	3.442431	84388	85337	84400	1.125	**0.014**
7	5857	3.895969	80960	82054	80960	1.351	–
8	5940	4.560248	72794	73417	72796	0.856	**0.003**
9	5546	7.520481	94768	95636	94776	0.916	**0.008**
10	5615	12.287986	92031	93131	92035	1.195	**0.004**
11	5655	8.742528	85911	87024	85914	1.296	**0.003**
12	5796	8.568687	74947	75696	74955	0.999	**0.011**
13	5603	7.283606	87074	87921	87077	0.973	**0.003**
14	5727	5.519233	75524	76737	75527	1.606	**0.004**
15	5355	6.721058	100343	101876	100353	1.528	**0.010**
16	6036	6.444898	76536	77017	76536	0.628	–
17	5975	5.146166	73436	73974	73436	0.733	–
18	5748	9.673332	73506	74144	73508	0.868	**0.003**
19	5541	10.960909	81305	82544	81310	1.524	**0.006**
20	5652	7.166207	87060	88126	87061	1.224	**0.001**
21	5574	5.500885	92117	93545	92120	1.550	**0.003**
22	5828	3.015198	85555	86199	85555	0.753	–
23	5924	3.848924	85558	86151	85560	0.693	**0.002**
24	5636	9.569704	74766	76946	74768	2.916	**0.003**
25	5742	5.680564	95856	96499	95856	0.671	–
26	5705	6.496537	89818	91056	89821	1.378	**0.003**
27	5596	4.879809	86696	87864	86697	1.347	**0.001**
28	5345	12.068610	90470	92354	90470	2.082	–
29	5933	8.075770	80072	80476	80073	0.505	**0.001**
30	5694	4.078793	84056	84764	84056	0.842	–
31	5471	12.537009	80372	81852	80384	1.841	**0.015**
32	5791	5.751170	91754	92750	91754	1.086	–
33	5729	5.256849	93117	94285	93124	1.254	**0.008**
34	5879	1.956182	82934	83759	82944	0.995	**0.012**
35	5624	2.702011	77692	78638	77694	1.218	**0.003**
36	5419	2.962949	99851	101111	99851	1.262	–
37	5922	4.963626	78184	78723	78184	0.689	–
38	5852	3.011891	82545	83222	82547	0.820	**0.002**
39	5903	4.565621	74000	74515	74003	0.696	**0.004**
40	5793	12.418498	70392	70954	70411	0.798	**0.027**
41	5910	3.743291	88242	88986	88242	0.843	–
42	5737	4.679572	83151	84266	83151	1.341	–
43	5450	5.285601	92279	93526	92281	1.351	**0.002**
44	5968	9.189751	72504	73307	72507	1.108	**0.004**
45	5882	3.839853	92359	92842	92359	0.523	–
46	5527	3.858474	86767	88088	86767	1.522	–
47	5493	10.253253	82972	84524	82976	1.871	**0.005**
48	5742	4.917429	87852	88392	87853	0.615	**0.001**
49	5743	6.236647	91759	92358	91761	0.653	**0.002**
50	6161	2.727070	77519	78064	77519	0.703	–
média		6.265145				1.120	0.004

Tabela 3.7: *Resultado numérico da heurística SHE_{MT6} para as 50 instâncias de Chandra et al. com $n = 100$ e $m = 20$.*

Instância	d	Tempo (s)	Valor objetivo			Gap(%)	
			Ótimo	Chandra *et al.*	SHE_{MI6}	Chandra *et al.*	SHE_{MI6}
1	5748	6.807712	81607	82606	81684	1.224	**0.094**
2	5919	4.763162	79799	80398	79960	0.751	**0.202**
3	6024	0.831476	81331	81954	81371	0.766	**0.049**
4	5781	0.462484	81869	82683	81911	0.994	**0.051**
5	5677	3.161784	87412	88749	87509	1.530	**0.111**
6	5594	1.520644	84388	85337	84434	1.125	**0.055**
7	5857	0.812317	80960	82054	81012	1.351	**0.064**
8	5940	0.796417	72794	73417	72840	0.856	**0.063**
9	5546	3.169144	94768	95636	94847	0.916	**0.083**
10	5615	1.533050	92031	93131	92080	1.195	**0.053**
11	5655	5.164639	85911	87024	85979	1.296	**0.079**
12	5796	2.195710	74947	75696	74972	0.999	**0.033**
13	5603	3.202414	87074	87921	87111	0.973	**0.042**
14	5727	1.153088	75524	76737	75594	1.606	**0.093**
15	5355	2.934202	100343	101876	100448	1.528	**0.105**
16	6036	1.161758	76536	77017	76588	0.628	**0.068**
17	5975	0.813531	73436	73974	73492	0.733	**0.076**
18	5748	3.460741	73506	74144	73613	0.868	**0.146**
19	5541	4.137301	81305	82544	81403	1.524	**0.121**
20	5652	2.978685	87060	88126	87112	1.224	**0.060**
21	5574	1.835802	92117	93545	92174	1.550	**0.062**
22	5828	0.867637	85555	86199	85596	0.753	**0.048**
23	5924	0.471464	85558	86151	85615	0.693	**0.067**
24	5636	8.849142	74766	76946	74814	2.916	**0.064**
25	5742	3.719367	95856	96499	95902	0.671	**0.048**
26	5705	1.522931	89818	91056	89861	1.378	**0.048**
27	5596	2.229025	86696	87864	86715	1.347	**0.022**
28	5345	4.358099	90470	92354	90543	2.082	**0.081**
29	5933	2.876853	80072	80476	80122	0.505	**0.062**
30	5694	0.813403	84056	84764	84105	0.842	**0.058**
31	5471	5.005082	80372	81852	80625	1.841	**0.315**
32	5791	1.496258	91754	92750	91779	1.086	**0.027**
33	5729	2.491286	93117	94285	93214	1.254	**0.104**
34	5879	0.822022	82934	83759	82967	0.995	**0.040**
35	5624	0.491841	77692	78638	77722	1.218	**0.039**
36	5419	1.162649	99851	101111	99877	1.262	**0.026**
37	5922	1.160164	78184	78723	78247	0.689	**0.081**
38	5852	0.849446	82545	83222	82574	0.820	**0.035**
39	5903	1.861605	74000	74515	74044	0.696	**0.059**
40	5793	4.320858	70392	70954	70683	0.798	**0.413**
41	5910	0.467607	88242	88986	88293	0.843	**0.058**
42	5737	2.205144	83151	84266	83178	1.341	**0.032**
43	5450	2.278105	92279	93526	92302	1.351	**0.025**
44	5968	3.723839	72504	73307	72600	1.108	**0.132**
45	5882	1.859236	92359	92842	92376	0.523	**0.018**
46	5527	2.177062	86767	88088	86805	1.522	**0.044**
47	5493	7.320204	82972	84524	82995	1.871	**0.028**
48	5742	1.853266	87852	88392	87891	0.615	**0.044**
49	5743	2.860497	91759	92358	91790	0.653	**0.034**
50	6161	1.168245	77519	78064	77551	0.703	**0.041**
média		2.483568				1.120	0.076

Tabela 3.8: *Resultado numérico da heurística SHE_{MI6} para as 50 instâncias de Chandra et al. com $n = 100$ e $m = 20$.*

		Gap médio (%)					
n	m	Chandra *et al.*	SHE_{R6}	SHE_{PT6}	SHE_{PI6}	SHE_{MT6}	SHE_{MI6}
5	5	**0.000**	1.034	6.234	6.559	0.597	2.021
5	10	**0.000**	1.502	6.621	6.844	0.592	2.201
5	15	**0.235**	4.109	8.876	10.146	2.648	1.765
5	20	**0.000**	2.332	6.530	6.697	1.701	1.349
10	5	**0.09**4	1.234	4.564	4.610	0.490	1.462
10	10	**0.081**	0.771	2.573	2.861	0.439	1.185
10	15	**0.099**	1.795	4.616	4.608	0.395	1.563
10	20	**0.074**	1.571	4.122	4.163	0.454	1.526
20	5	**0.058**	0.493	2.117	2.141	0.125	0.623
20	10	**0.025**	0.564	2.229	2.174	0.149	0.715
20	15	**0.031**	0.429	1.532	1.532	0.208	0.508
20	20	**0.045**	0.435	1.711	1.711	0.154	0.532
50	5	0.309	0.167	0.925	0.925	**0.025**	0.183
50	10	0.183	0.212	0.712	0.705	**0.028**	0.185
50	15	0.125	0.148	0.655	0.655	**0.019**	0.156
50	20	0.142	0.277	1.041	1.041	**0.016**	0.229
80	5	0.889	0.081	0.391	0.399	**0.005**	0.080
80	10	0.659	0.136	0.632	0.632	**0.007**	0.102
80	15	0.623	0.086	0.425	0.424	**0.007**	0.088
80	20	0.633	0.081	0.429	0.421	**0.007**	0.091
100	5	1.721	0.072	0.356	0.355	**0.004**	0.059
100	10	1.163	0.097	0.371	0.367	**0.003**	0.071
100	15	1.163	0.067	0.304	0.312	**0.005**	0.052
100	20	1.120	0.111	0.426	0.442	**0.004**	0.076

Tabela 3.9: *Comparação do Gap médio de Chandra et al. e as heurísticas estudadas.*

	d_1		d_2		d_3		d_4	
Instância	Tempo(s)	Solução	Tempo	Solução	Tempo	Solução	Tempo	Solução
1	0.501	718382	0.489	521162	0.495	401790	0.495	342789
2	0.532	711743	0.529	519929	0.521	393973	0.553	328186
3	0.775	707310	0.693	515954	0.624	391739	0.600	341087
4	0.639	711055	0.596	520637	0.578	393354	0.698	334956
5	0.535	705311	0.550	506788	0.518	376454	0.530	318163
6	0.635	682546	0.613	501154	0.547	380758	0.548	334602
7	0.742	705591	0.565	517665	0.622	396947	0.607	346184
8	0.635	704987	0.808	516263	0.568	401073	0.694	350947
9	0.548	704238	0.518	512795	0.538	392889	0.552	344083
10	0.578	685086	0.611	498938	0.576	381860	0.648	335971

Tabela 3.10: *Resultados da heurística SHE_{R6} para as 10 instâncias de tamanho $n = 200$ e $m = 10$ com 4 datas de entrega restritivas.*

	d_1		d_2		d_3		d_4	
Instância	Tempo(s)	Solução	Tempo	Solução	Tempo	Solução	Tempo	Solução
1	79.179	707330	130.801	512868	165.669	387367	197.248	337112
2	112.131	702684	334.194	504926	381.630	380969	529.967	321740
3	275.911	693114	424.153	501255	473.620	385843	511.393	336483
4	92.096	704504	164.962	502405	169.299	382975	235.605	327208
5	401.099	690176	472.850	486956	558.986	366150	516.394	312313
6	265.641	674603	407.440	484730	431.812	374161	542.198	328566
7	244.077	699277	469.050	499861	627.984	385551	660.006	340937
8	300.322	692311	446.579	502660	488.634	389740	713.373	345108
9	290.146	694102	352.999	502346	493.371	383920	521.366	338737
10	269.626	676303	415.644	482334	616.928	369511	576.776	331696

Tabela 3.11: *Resultados da heurística SHE_{MT6} para as 10 instâncias de tamanho $n = 200$ e $m = 10$ com 4 datas de entrega restritivas.*

			IDR (Índice de Desvio Relativo)				
	n	m	SHE_{R6}	SHE_{PT6}	SHE_{PI6}	SHE_{MT6}	SHE_{MI6}
d_1	100	5	0.352	0.928	0.864	**0.046**	0.076
	100	10	0.326	0.861	0.891	0.094	**0.081**
	100	20	0.455	0.975	0.848	**0.078**	0.089
	200	10	0.291	0.843	0.962	**0.110**	0.142
	200	20	0.451	0.976	0.817	**0.052**	0.157
	500	20	0.387	0.912	0.885	0.104	**0.088**
d_2	100	5	0.289	0.841	0.886	**0.080**	0.124
	100	10	0.344	0.839	0.826	**0.075**	0.096
	100	20	0.440	0.987	0.902	0.119	**0.105**
	200	10	0.401	0.923	0.894	0.095	**0.093**
	200	20	0.309	0.853	0.925	0.084	**0.074**
	500	20	0.402	0.976	0.936	**0.043**	0.108
d_3	100	5	0.228	0.864	0.897	**0.061**	0.098
	100	10	0.203	0.833	0.851	**0.112**	0.151
	100	20	0.350	0.914	0.836	**0.064**	0.076
	200	10	0.291	0.844	0.827	**0.055**	0.070
	200	20	0.236	0.902	0.853	**0.116**	0.122
	500	20	0.267	0.890	0.936	0.117	**0.114**
d_4	100	5	0.491	0.927	0.814	**0.063**	0.083
	100	10	0.481	0.946	0.803	**0.110**	0.125
	100	20	0.465	0.879	0.831	**0.042**	0.116
	200	10	0.437	0.866	0.917	**0.106**	0.146
	200	20	0.393	0.911	0.881	**0.073**	0.148
	500	20	0.423	0.878	0.896	**0.054**	0.091

Tabela 3.12: *Comparação do IDR médio para cada tamanho de instância e as quatro datas de entrega.*

Capítulo 4

Conclusões

Neste trabalho estudamos o problema de programação de tarefas em um ambiente *flow shop* permutacional com o objetivo de minimizar a soma total dos adiantamentos e atrasos das tarefas em relação a uma data de entrega comum e restritiva. Apresentamos também uma formulação matemática em programação linear inteira mista para o problema. O objetivo deste modelo é estabelecer uma definição precisa do problema e obter soluções exatas para instâncias pequenas do mesmo. A formulação matemática não consegue diferenciar soluções que têm tarefas com tempo desnecessário no *buffer* de aquelas que não, portanto, é apresentado também um procedimento para minimizar aquele tempo.

Considerando a dificuldade de resolver o problema em estudo (pois pertence à classe NP-difícil), é apresentado um método heurístico usado para obter soluções de boa qualidade em um tempo razoável. O método heurístico faz bom uso de um algoritmo linear de atribuição de tempos de conclusão para sequências de tarefas chamado algoritmo de *timing*, o qual aproveita certas propriedades analíticas inerentes às soluções ótimas. Também, o método usa técnicas de busca local (completa e reduzida). Basicamente, o método desenvolvido neste trabalho é uma extensão para m máquinas do trabalho feito por Sakuraba *et al.* (2009).

O método heurístico foi implementado e testado inicialmente com resultados positivos, o que motivou um teste mais intensivo como seguinte passo. Os resultados obtidos sugerem que nosso método heurístico é bastante competitivo na busca de soluções de boa qualidade.

4.1 Perspectivas futuras

Como perspectivas futuras deste trabalho são propostas as seguintes atividades:

- Considerar o problema *flow shop* permutacional com pesos para os adiantamentos e atrasos (*weighted earliness and tardiness flow shop problem*).
- Incluir na função objetivo o critério de minimização do trabalho em processo (work-in-process, WIP).

Apêndice A

Implementação do modelo PLIM

Apresentamos a implementação do modelo matemático (arquivo *flowshop.mod*) da Seção 2.3 na linguagem de modelado AMPL e a representação dos dados da Tabela 2.3 para sua execução pelo CPLEX (arquivo *chandran5m5d334.cplex.dat*).

- Arquivo *flowshop.mod*:

```
param n;
param m;

param p {i in 1..n, j in 1..m};e

param d;

var x {i in 1..n, j in 1..n} binary;
var C {j in 1..n};
var W {j in 1..n,   k in 1..m-1} >= 0;
var I {j in 1..n-1, k in 1..m  } >= 0;
var T {j in 1..n} >= 0;
var E {j in 1..n} >= 0;
var I0 >= 0;

minimize objective:

    sum {j in 1..n} ( E[j] + T[j] );

subject to

R1:

    C[1] = I0 + sum {k in 1..m, i in 1..n} ( x[i,1] * p[i,k] ) +
                sum {i in 1..m-1} W[1,i];

R2 {j in 2..n}:

    C[j] = C[j-1] + I[j-1,m] + sum {i in 1..n} ( x[i,j] * p[i,m] );

R3 {j in 1..n-1, k in 1..m-1}:

    I[j,k] + sum {i in 1..n} ( x[i,j+1] * p[i,k]   ) + W[j+1,k] =
    W[j,k] + sum {i in 1..n} ( x[i,j]   * p[i,k+1] ) + I[j,k+1];

R4 {j in 1..n}:

    sum {i in  1..n} x[i,j] = 1;
```

```
R5 {i in 1..n}:

    sum {j in  1..n} x[i,j] = 1;

R6 {j in 1..n}:

    T[j] >= C[j] - d;

R7 {j in 1..n}:

    E[j] >= d - C[j];
```

- Arquivo *chandran5m5d334.cplex.dat*:

```
param n := 5;
param m := 5;
param d := 334;
param p(tr): 1 2 3 4 5 :=
1  80 94 87 63 18
2  41 57 29 6  70
3  66 63 84 10 96
4  27 16 32 42 69
5  93 14 8  70 23;
```

Apêndice B

Cota superior para a data de entrega restritiva

Podemos obter uma cota superior para a data de entrega restritiva se calcularmos a menor data de entrega a partir da qual pode-se obter uma programação ótima mediante o procedimento de Bagchi *et al.* (1986). Esse procedimento, desenvolvido para o problema de máquina única, lista as tarefas em ordem não crescente de seus tempos de processamento e as programa da seguinte forma: as tarefas na posição impar são programadas na mesma ordem como um bloco único (sem interrupções) de tal forma que a última tarefa da sequência completa seu processamento exatamente na data de entrega. As tarefas na posição par são programadas na ordem reversa também sem interrupções e com a primeira tarefa começando seu processamento justamente na data de entrega.

O procedimento completo para gerar uma cota superior para a data de entrega restritiva é o seguinte:

Passo 1. Efetuamos o algoritmo de Bagchi *et al.* (1986) usando os tempos de processamento na última máquina (M_m) e uma data de entrega não restritiva (por exemplo $d = \sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}$) para determinar uma sequência de tarefas ótima.

Passo 2. Com a sequência criada, as tarefas são programadas nas máquinas $M_1, \ldots, M_{m-1}$ o mais rápido possível.

Passo 3. As tarefas são programadas na máquina M_m (última máquina) sem interrupções, começando do tempo de conclusão da última tarefa na máquina M_{m-1}.

Passo 4. Para cada tarefa calculamos a diferença entre o tempo de início na máquina M_m e o tempo de conclusão em M_{m-1}. A programação na última máquina é adiantada em um intervalo igual ao menor valor de tais diferenças.

Passo 5. A cota superior para a data de entrega restritiva é definida como o tempo de conclusão da tarefa em posição $\lceil n/2 \rceil$.

Referências Bibliográficas

Aarts e Lenstra (2003) E. Aarts e J. K. Lenstra. Local search in combinatorial optimization, 2003. Citado na pág. 2

Bagchi *et al.* (1986) U. Bagchi, R. S. Sullivan e Y. L. Chang. Minimizing mean absolute deviation of completion times about a common due date. *Naval Research Logistics Quarterly*, 33(2):227–240. ISSN 1931-9193. doi: 10.1002/nav.3800330206. URL http://dx.doi.org/10.1002/nav.3800330206. Citado na pág. 3, 9, 31, 44

Baker e Scudder (1990) K. R. Baker e G. D. Scudder. Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38(1):22–36. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/171295. Citado na pág. 2

Biskup e Feldmann (2001) D. Biskup e M. Feldmann. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Comput. Oper. Res.*, 28(8):787–801. ISSN 0305-0548. doi: 10.1016/S0305-0548(00)00008-3. URL http://dx.doi.org/10.1016/S0305-0548(00)00008-3. Citado na pág. 1

Bratley *et al.* (2011) P. Bratley, B. L. Fox e L.E. Schrage. *A guide to simulation*. Springer Science & Business Media. Citado na pág. 30

Chandra *et al.* (2009) P. Chandra, P. Mehta e D. Tirupati. Permutation flow shop scheduling with earliness and tardiness penalties. *International Journal of Production Research*, 47(20):5591–5610. Citado na pág. 3, 13, 26, 28, 29

Chrétienne e Sourd (2003) P. Chrétienne e F. Sourd. Pert scheduling with convex cost functions. *Theoretical Computer Science*, 292(1):145 – 164. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/S0304-3975(01)00220-1. URL http://www.sciencedirect.com/science/article/pii/S0304397501002201. Citado na pág. 19

Chvátal (1983) V. Chvátal. *Linear Programming*. Series of books in the mathematical sciences. W. H. Freeman. ISBN 9780716715870. Citado na pág. 9

Dag (2013) S. Dag. An application on flowshop scheduling. *Alphanumeric Journal*, 1:47–56. Citado na pág. 3

Dantzig (1990) G. B. Dantzig. A history of scientific computing. chapter Origins of the Simplex Method, páginas 141–151. ACM, New York, NY, USA. ISBN 0-201-50814-1. doi: 10.1145/87252.88081. URL http://doi.acm.org/10.1145/87252.88081. Citado na pág. 9

Dudek *et al.* (1992) R. A. Dudek, S. S. Panwalkar e M. L. Smith. The lessons of flowshop scheduling research. *Operations Research*, 40(1):7–13. doi: 10.1287/opre.40.1.7. URL http://dx.doi.org/10.1287/opre.40.1.7. Citado na pág. 3

Emmons e Vairaktarakis (2013) H. Emmons e G. Vairaktarakis. *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications.* International Series in Operations Research and Management Science (Book 182). Springer; 2013 edition (September 14, 2012). ISBN 9781461451518. Citado na pág. 8

Feldmann e Biskup (2003) M. Feldmann e D. Biskup. Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Comput. Ind. Eng.*, 44(2):307–323. ISSN 0360-8352. doi: 10.1016/S0360-8352(02)00181-X. URL http://dx.doi.org/10.1016/S0360-8352(02)00181-X. Citado na pág. 8

Fourer *et al.* (2002) R. Fourer, D. M. Gay e Kernighan B. W. *AMPL - A Modeling language for mathematical Programming.* Duxbury Press, 2th ed. ISBN 0-534-38809-4. Citado na pág. 13

Gordon *et al.* (2002) V. Gordon, J. M. Proth e C. Chu. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, 139(1):1 – 25. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/S0377-2217(01)00181-3. URL http://www.sciencedirect.com/science/article/pii/S0377221701001813. Citado na pág. 2

Hall *et al.* (1991) N. G. Hall, W. Kubiak e S. P. Sethi. Earliness–tardiness scheduling problems, ii: Deviation of completion times about a restrictive common due date. *Operations Research*, 39(5):847–856. Citado na pág. 2, 3, 8

Hendel e Sourd (2007) Y. Hendel e F. Sourd. An improved earliness-tardiness timing algorithm. *Computers and Operations Research*, 34(10):2931 – 2938. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2005.11.004. URL http://www.sciencedirect.com/science/article/pii/S0305054805003527. Citado na pág. 3

Hoogeveen e Van de Velde (1991) J. A. Hoogeveen e S. L. Van de Velde. Scheduling around a small common due date. *European Journal of Operational Research*, 55(2):237–242. Citado na pág. 3, 9

Johnson (1954) S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68. ISSN 1931-9193. doi: 10.1002/nav.3800010110. URL http://dx.doi.org/10.1002/nav.3800010110. Citado na pág. 2

Kanet (1981) J. J. Kanet. Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly*, 28(4):643–651. ISSN 1931-9193. doi: 10.1002/nav.3800280411. URL http://dx.doi.org/10.1002/nav.3800280411. Citado na pág. 2, 9

Kanet e Sridharan (2000) J. J. Kanet e V. Sridharan. Scheduling with inserted idle time: Problem taxonomy and literature review. *Operation Research*, 48(1):99–110. ISSN 0030-364X. doi: 10.1287/opre.48.1.99.12447. URL http://dx.doi.org/10.1287/opre.48.1.99.12447. Citado na pág. 2

Khachiian (1979) L. G. Khachiian. Polynomial algorithm in linear programming. Em *Akademiia Nauk SSSR, Doklady*, volume 244, páginas 1093–1096. Citado na pág. 9

Kim (1993) Y. D. Kim. Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of the Operational Research Society*, 44(1):19–28. ISSN 1476-9360. doi: 10.1057/jors.1993.3. URL http://dx.doi.org/10.1057/jors.1993.3. Citado na pág. 31

Kim (1995) Y. D. Kim. Minimizing total tardiness in permutation flowshops. *European Journal of Operational Research*, 85(3):541–555. Citado na pág. 8

Klee e Minty (1972) V. Klee e G. J. Minty. How good is the simplex algorithm? Em O. Shisha, editor, *Inequalities*, volume III, páginas 159–175. Academic Press, New York. Citado na pág. 9

Lauff e Werner (2004) V. Lauff e F. Werner. Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey. *Mathematical and Computer Modelling*, 40(5): 637 – 655. ISSN 0895-7177. doi: http://dx.doi.org/10.1016/j.mcm.2003.05.019. URL http://www.sciencedirect.com/science/article/pii/S0895717704803285. Citado na pág. 2

Lawler *et al.* (1982) E.L. Lawler, J.K. Lenstra e A.H.G. Rinnooy Kan. Recent developments in deterministic sequencing and scheduling: a survey. Em M.A.H. Dempster, J.K. Lenstra e A.H.G. Rinnooy Kan, editors, *Deterministic and stochastic scheduling*, volume 84 of *NATO Advanced Study Institutes Series*, páginas 35–73. Springer Netherlands. ISBN 978-94-009-7803-4. doi: 10.1007/978-94-009-7801-0_3. URL http://dx.doi.org/10.1007/978-94-009-7801-0_3. Citado na pág. 8, 17

Michiels *et al.* (2007) W. Michiels, E. Aarts e J. Korst. *Theoretical Aspects of Local Search (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 3540358536. Citado na pág. 2

Nawaz *et al.* (1983) M. Nawaz, E E. Enscore e I Ham. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *Omega*, 11(1):91 – 95. ISSN 0305-0483. doi: http://dx.doi.org/10.1016/0305-0483(83)90088-9. URL http://www.sciencedirect.com/science/article/pii/0305048383900889. Citado na pág. 15, 24

Ohno (1982) T. Ohno. How the toyota production system was created. *Japanese Economy*, 10(4):83–101. URL http://EconPapers.repec.org/RePEc:mes:jpneco:v:10:y:1982:i:4:p:83-101. Citado na pág. 1

Pan (1997) C. H. Pan. A study of integer programming formulations for scheduling problems. *International Journal of Systems Science*, 28(1):33–41. doi: 10.1080/00207729708929360. URL http://dx.doi.org/10.1080/00207729708929360. Citado na pág. 3

Pinedo (2012) M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 4th ed. ISBN 0387789340, 9780387789347. Citado na pág. 3

Ronconi e Birgin (2012) D. P. Ronconi e E. G. Birgin. Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness. Em Roger Z. Ríos-Mercado e Yasmín A. Ríos-Solís, editors, *Just-in-Time systems*, volume 60 of *Springer Optimization and Its Applications*, páginas 91–105. Springer New York. ISBN 978-1-4614-1122-2. doi: 10.1007/978-1-4614-1123-9_5. URL http://dx.doi.org/10.1007/978-1-4614-1123-9_5. Citado na pág. 3, 10

Sakuraba *et al.* (2009) C.S. Sakuraba, D. P. Ronconi e F. Sourd. Scheduling in a two-machine flowshop for the minimization of the mean absolute deviation from a common due date. *Computers & Operations Research*, 36(1):60 – 72. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2007.07.005. URL

http://www.sciencedirect.com/science/article/pii/S0305054807001335. Part Special Issue: Operations Research Approaches for Disaster Recovery Planning. Citado na pág. 1, 2, 3, 15, 17, 19, 23, 24, 26, 41

Sarper (1995) H. Sarper. Minimizing the sum of absolute deviations about a common due date for the two-machine flow shop problem. *Applied Mathematical Modelling*, 19(3):153–161. ISSN 0307-904X. doi: http://dx.doi.org/10.1016/0307-904X(94)00022-X. URL http://www.sciencedirect.com/science/article/pii/0307904X9400022X. Citado na pág. 3, 15

Sen e Gupta (1984) T. Sen e S. K. Gupta. A state-of-art survey of static scheduling research involving due dates. *Omega*, 12(1):63 – 76. ISSN 0305-0483. doi: http://dx.doi.org/10.1016/0305-0483(84)90011-2. URL http://www.sciencedirect.com/science/article/pii/0305048384900112. Citado na pág. 1

Stafford *et al.* (2005) F. E. Stafford, T. F. Tseng e D. J. N. Gupta. Comparative evaluation of milp flowshop models. *Journal of the Operational Research Society*, 56(1):88–101. ISSN 1476-9360. doi: 10.1057/palgrave.jors.2601805. URL http://dx.doi.org/10.1057/palgrave.jors.2601805. Citado na pág. 3

Sung e Min (2001) C.S. Sung e J.I. Min. Scheduling in a two-machine flowshop with batch processing machine(s) for earliness/tardiness measure under a common due date. *European Journal of Operational Research*, 131(1):95 – 106. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/S0377-2217(99)00447-6. URL http://www.sciencedirect.com/science/article/pii/S0377221799004476. Citado na pág. 2

Szwarc (1989) W. Szwarc. Single-machine scheduling to minimize absolute deviation of completion times from a common due date. *Naval Research Logistics (NRL)*, 36(5):663–673. ISSN 1520-6750. doi: 10.1002/1520-6750(198910)36:5<663::AID-NAV3220360510>3.0.CO;2-X. URL http://dx.doi.org/10.1002/1520-6750(198910)36:5<663::AID-NAV3220360510>3.0.CO;2-X. Citado na pág. 3

Taillard (1993) E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/0377-2217(93)90182-M. URL http://www.sciencedirect.com/science/article/pii/037722179390182M. Citado na pág. 15, 30

Wagner (1959) H. M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140. ISSN 1931-9193. doi: 10.1002/nav.3800060205. URL http://dx.doi.org/10.1002/nav.3800060205. Citado na pág. 3, 10

Yeung *et al.* (2004) W. K. Yeung, C. Oguz e T. C. E. Cheng. Two-stage flowshop earliness and tardiness machine scheduling involving a common due window. *International Journal of Production Economics*, 90(3):421 – 434. ISSN 0925-5273. doi: http://dx.doi.org/10.1016/S0925-5273(03)00044-6. URL http://www.sciencedirect.com/science/article/pii/S0925527303000446. Production Control and Scheduling. Citado na pág. 2