UNIVERSIDADE DE SÃO PAULO
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

MAURICIO IWAMA TAKANO

Uma contribuição para o problema de programação de operações *flow shop* com *buffer* zero e tempos de *setup* dependente da sequência e da máquina

SÃO CARLOS
2016

MAURICIO IWAMA TAKANO

Uma contribuição para o problema de programação de operações *flow shop* com *buffer* zero e tempos de *setup* dependente da sequência e da máquina

Tese apresentada ao programa de pós-graduação em Engenharia de Produção da Escola de Engenharia de São Carlos, da Universidade de São Paulo para obtenção do título de doutor em Engenharia de Produção

Área de concentração:
Processos e Gestão de Operações

Orientador: Prof. Dr. Marcelo Seido Nagano

SÃO CARLOS
2016

Takano, Mauricio Iwama
T136u Uma contribuição para o problema de programação de operações flow shop com buffer zero e tempos de setup dependente da sequência e da máquina / Mauricio Iwama Takano; orientador Marcelo Seido Nagano. São Carlos, 2016.

Tese (Doutorado) - Programa de Pós-Graduação em Engenharia de Produção e Área de Concentração em Processos e Gestão de Operações -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2016.

1. Programação da produção. 2. Flow shop. 3. Bloqueio. 4. Buffer zero. 5. Setup dependente. 6. Métodos heurísticos. 7. Programação inteira mista. 8. Branch-and-Bound. I. Título.

FOLHA DE JULGAMENTO

Candidato: Tecnólogo **MAURICIO IWAMA TAKANO.**

Título da tese: "Uma contribuição para o problema de programação de operações *flow shop* com *buffer* zero e tempos de *setup* dependente da sequência e da máquina ".

Data da defesa: 03/08/2016.

Comissão Julgadora: **Resultado:**

Prof. Dr. **Marcelo Seido Nagano** — Aprovado
(Orientador)
(Escola de Engenharia de São Carlos/EESC)

Prof. Dr. **Walther Azzolini Junior** — APROVADO
(Escola de Engenharia de São Carlos/EESC)

Prof. Dr. **Marcelo Bertolete Carneiro** — Aprovado
(Escola de Engenharia de São Carlos/EESC)

Profª. Drª. **Juliana Keiko Sagawa** — Aprovada
(Universidade federal de São Carlos/UFSCar)

Prof. Titular **Marcio Mattos Borges de Oliveira** — APROVADO
(Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto/FEARP-USP)

Coordenadora do Programa de Pós-Graduação em Engenharia de Produção
Profa. Associada **Daisy Aparecida do Nascimento Rebelatto**

Presidente da Comissão de Pós-Graduação:
Prof. Associado **Luis Fernando Costa Alberto**

Dedico este trabalho à minha mãe Luiza Massako Iwama Takano e ao meu pai Harumi Takano (*in memorian*), que sempre me apoiaram, incentivaram e acreditaram em mim.

AGRADECIMENTOS

À minha mãe Luiza Massako Iwama Takano e ao meu pai Harumi Takano (*in memorian*) que me deram toda estrutura e suporte para que eu pudesse estar aqui realizando este trabalho, tanto espiritual como também financeiramente. Um agradecimento especial à minha esposa Karina Assolari Takano, por sempre me apoiar e me incentivar, sempre compreendendo todas as dificuldades passadas durante os estudos, nunca deixando de estar ao meu lado mesmo nos momentos mais difíceis.

Aos meus amigos, pelo grande apoio e paciência, me incentivando nos estudos não me deixando desviar de meus objetivos e sempre compreendendo quando me ausentava para realizar minhas pesquisas e trabalhos.

Ao meu orientador Marcelo Seido Nagano pelo enorme auxílio nos estudos e no conhecimento, pela sua grande paciência para me orientar sempre que me encontrei perdido e por compartilhar de seus conhecimentos para que eu pudesse crescer e desenvolver este trabalho da melhor forma que pude.

À Escola de Engenharia de São Carlos (EESC) pelo grande suporte dado ao disponibilizar todos seus laboratórios e ferramentas que necessitei durante a execução do trabalho e também pelo apoio intelectual de todos os professores que sempre estiveram presente durante todo o processo deste trabalho.

À Universidade Tecnológica Federal do Paraná (UTFPR) por acreditar em minha capacidade de desenvolvimento e criação e também por permitir me afastar de minhas atividades por um período para que eu pudesse me dedicar integralmente aos estudos e concluir da melhor forma possível este trabalho.

RESUMO

TAKANO, Mauricio I. **Uma contribuição para o problema de programação de operações *flow shop* com *buffer* zero e tempos de setup dependente da sequência e da máquina**. 2010. 127 f. Tese (Doutorado) - Departamento de Engenharia de Produção, Universidade de São Paulo, São Carlos, 2016.

O problema do sequenciamento da produção diz respeito à alocação das tarefas nas máquinas em um ambiente de fabricação, o qual vem sendo amplamente estudado. O sequenciamento pode variar em tamanho e complexidade dependendo do tipo de ambiente onde ele é aplicado, do número e tipos de restrições tecnológicas e da função objetivo do problema. A utilização de métodos de decisão para a solução de problemas de sequenciamento na indústria depende de modelos que sejam capazes de oferecer soluções para os problemas reais, que geralmente envolvem diversas restrições, os quais devem ser considerados simultaneamente. No presente trabalho o problema de sequenciamento da produção em ambientes *flow shop* permutacionais, com bloqueio com *buffer* zero, e com tempos de *setup* dependente da sequência e da máquina, com o objetivo de minimização do *makespan* é estudado, sendo este considerado um problema *NP*-Completo. O problema é pouco explorado na literatura. No presente trabalho é apresentado um procedimento de cálculo para o *makespan* e três métodos de solução para o problema: quatro limitantes inferiores para o procedimento *Branch-and-Bound*; quatro modelos *MILP*, sendo dois deles adaptados; e 28 modelos heurísticos construtivos adaptados para o problema. Os métodos desenvolvidos baseiam-se em propriedades matemáticas do problema que são apresentadas neste trabalho como limitante inferior e limitante superior. Dentre todos os modelos *MILP*, o modelo adaptado *RBZBS*1 obteve os melhores resultados para os problemas menores e o modelo desenvolvido *TNZBS*1 obteve os melhores desvios relativos médios do *makespan* para os problemas maiores, que não foram resolvidos dentro do limite de tempo computacional estipulado. O limitante inferior para o *Branch-and-Bound* LB_{TN2} foi melhor que os demais tanto no tempo computacional e no número de nós explorados como também no número de problemas não resolvidos e no desvio relativo médio do *makespan*. Foi realizado uma comparação entre o melhor modelo *MILP* e o melhor limitante inferior para o *Branch-and-Bound*, sendo que o último obteve melhores resultados para os problemas testados. Entre os métodos heurísticos adaptados, o PF foi o que obteve, de uma forma geral, os melhores resultados em todas as fases.

Palavras-chave: Programação da produção. *Flow shop*. Bloqueio. *Buffer* zero. *Setup* dependente. Métodos heurísticos. Programação inteira mista. *Branch-and-Bound*.

ABSTRACT

TAKANO, Mauricio I. **A contribution to the flow shop problem with zero buffer and sequence and machine dependent setup times**. 2016. 127 f. Theses (Doctorate) - Department of Production Engineering, Universidade de São Paulo, São Carlos, 2016.

Production scheduling is defined as a problem of allocating jobs in machines in a production environment and it has been largely studied. The scheduling can vary in difficulty and complexity depending on the environment, the variety and types of technological restraints and the objective function of the problem. The use of decision making methods to solve scheduling problems in the industry needs models that are capable to solve real problems, that usually involve a big variety of restraints that have to be simultaneously studied. At the present work the scheduling problem in a permutational flow shop environment, considering blocking with zero buffer, and sequence and machine dependent setup times, with the objective of minimizing makespan is studied, which is considered a NP-Complete problem and little explored in literature. The work presents a calculation procedure for the makespan and three solution methods for the problem: four lower bounds for the Branch-and-Bound procedure; four *MILP* models, two of which are adapted; and 28 constructive heuristic methods adapted to the problem. The methods developed are based on mathematical properties of the problem that are presented in this work as a lower bound and an upper bound. Among all the *MILP* models, the adapted model *RBZBS*1 was the one to obtain the best results for the smaller problems, and the developed model *TNZBS*1 obtained the smallest mean relative deviation of the makespan for the bigger problems that were not solved within the specified computational time limit. The lower bound for the Branch-and-Bound LB_{TN2} obtained smaller computational times and number of explored nodes as well as the number of unsolved problems and the mean relative deviation for the makespan than all other lower bounds. Also, a comparison among the best *MILP* model and the best lower bound for the Branch-and-Bound was performed, being that the last obtained better results for the tested problems. Among the adapted heuristic methods, the PF heuristic was the one that obtained, in general, the better results in all phases.

Keywords: Production scheduling. Flow shop. Blocking. Zero buffer. Dependent setup. Heuristic methods. Mixed integer program. Branch-and-Bound.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIAÇÕES

B&B	*Branch-and-Bound*
FIFO	*First In First Out*
GA	*Genetic Algorithm*
IP	*Integer Programming*
LP	*Linear Programming*
LPT	*Largest Processing Time*
LS	*Local Search*
MILP	*Mixed Integer-Linear Programming*
MINLP	*Mixed Integer Non-Linear Programming*
MM	*MinMax*
NLP	*Non-Linear Programming*
PCP	Planejamento e Controle da Produção
PF	*Profile Fitting*
PO	Pesquisa Operacional
PSO	*Particle Swarm Optimization*
SM	*Search Metaheuristic*
SPT	*Shortest Processing Time*
TS	*Tabu Search*

LISTA DE SÍMBOLOS

n	Número de tarefas.
m	Número de máquinas.
$j_1, j_2, ..., j_n$	Conjunto de tarefas que devem ser alocadas nas máquinas.
$k_1, k_2, ..., k_m$	Conjunto de máquinas que realizarão as tarefas.
$i_1, i_2, ..., i_n$	Conjunto de tarefas que precedem diretamente as tarefas j_1, j_2, ..., j_n na sequência de fabricação.
P_{jk}	Tempo de processamento da tarefa *j* na máquina *k*.
S_{ijk}	Tempo de *setup* da tarefa *j*, precedida diretamente pela tarefa *i*, na máquina *k* (para problemas com *setup* dependente da sequência de fabricação).
C_{jk}	Data de término da tarefa *j* na máquina *k*.
D_{jk}	Data em que uma tarefa libera a máquina após o término do processamento. $D_{\sigma k} \geq C_{\sigma k}$: se não houver bloqueio $D_{\sigma k} = C_{\sigma k}$, caso contrário $D_{\sigma k} > C_{\sigma k}$.
R_{jk}	Data de término do *setup* da máquina *k* para execução da tarefa *j*.
O_{jk}	Tempo de ociosidade da máquina *k*, *i.e*. Intervalo de tempo entre o término do *setup* para execução da tarefa *j* e o início da operação da tarefa *j* na máquina *k*.
B_{jk}	Tempo de bloqueio da tarefa *j* na máquina *k*, *i.e.* o intervalo de tempo entre o término da tarefa *j* na máquina *k* e o início da operação da mesma tarefa na máquina (*k*+1).
α	Índice que descreve o tipo de ambiente de produção.
β	Índice que descreve as restrições tecnológicas do problema de sequenciamento.
γ	Índice que descreve qual é o critério de avaliação.

SUMÁRIO

1. INTRODUÇÃO

As empresas têm cada vez mais investido em processos de melhoria, devido à alta competitividade no mercado e à globalização,. Para operar de forma eficiente elas precisam planejar e controlar as tarefas a serem realizadas, aumentando, desta forma, a produtividade e, com isso, reduzindo os custos de produção e, desta forma, possivelmente deixando seus produtos e serviços mais atrativos aos consumidores. Inserido nesse contexto, a área de planejamento e controle da produção (*PCP*) é responsável por coordenar e aplicar os recursos produtivos de forma a atender da melhor maneira possível aos planos estabelecidos nos níveis estratégico, tático e operacional (Tubino, 2007).

Em um sistema de manufatura as tarefas entram em um ambiente de produção como matéria-prima e saem como produtos prontos para o consumo do cliente (interno ou externo). A fabricação de um produto geralmente envolve diversos processos, que são realizados em máquinas ou estações de trabalho que fazem parte do ambiente de produção. Essas máquinas ou estações de trabalho são organizadas fisicamente nos ambientes de produção da maneira mais eficiente possível para processar as tarefas. Segundo Groover (2011) o ambiente de produção mais adequado para cada empresa depende, dentre outras coisas, do tipo e da quantidade de tarefas a serem realizadas.

O sequenciamento da produção, uma das atividades exercidas pelo *PCP*, é responsável por adequar as necessidades de fabricação aos recursos disponíveis pela empresa. O sequenciamento é, segundo Pinedo (2008), um processo de tomada de decisão que é utilizado em muitas empresas de manufatura ou de serviços. Quando aplicado em ambientes de fabricação, o sequenciamento realiza a otimização da sequencia de produção, ou seja, ajuda a definir quais tarefas devem ser realizadas primeiro e quais devem ser realizadas em seguida, de forma a otimizar o uso dos recursos do ambiente.

Dentre as diversas informações que devem ser consideradas para se realizar o processo de sequenciamento da produção pode-se citar o tempo de preparação da máquina ou tempo de *setup*. O *setup* da máquina é a preparação que deve ser realizada para que a máquina possa iniciar o processamento de uma tarefa

(por exemplo troca da ferramenta de corte, troca do molde, troca do fluido de corte, troca do sistema de fixação, *etc.*).

Em muitos estudos o tempo de *setup* é considerado incluído ao tempo de processamento das tarefas (Lee; Dicesare, 1994; Wang *et al.*, 2004; Jerald *et al.*, 2005; Low *et al.*, 2005; Ecker; Gupta, 2005; Saidi-Mehrabad; Fattahi, 2007; Kim; Jeong, 2007). Isso, apesar de facilitar a obtenção de uma solução, limita algumas estratégias da produção, como por exemplo, a possibilidade de realizar o *setup* da máquina quando ela estiver disponível, aguardando a tarefa (ou seja, a preparação da máquina ser realizada antes mesmo da tarefa terminar de ser processada na máquina anterior).

Para se obter melhores resultados alguns autores utilizaram tempo de *setup* separado do tempo de processamento, que permite um planejamento da produção mais sensível e realístico (Gupta *et al.*, 1997; Chang *et al.*, 2004; Biskup; Herrmann, 2008; Boiko, 2008; Araujo; Nagano, 2011; Moccellin; Nagano, 2011; Vallada; Ruiz, 2011; Li *et al.*, 2011; Gómez-Gasquet *et al.*, 2012; González *et al.*, 2012; Lee *et al.*, 2012; Nagano; Mesquita, 2012; Sabouni; Logendran, 2013; Xi; Jang, 2012).

Esse tempo de *setup* pode ser dependente da sequencia de fabricação, da máquina ou de ambos. Quando dependente apenas da sequência, o tempo de *setup* é diferente para cada par de tarefas, ou seja, depende de qual tarefa acabou de ser processada e qual tarefa irá iniciar seu processamento, independente da máquina onde será realizado o processamento. Quando dependente apenas da máquina, o tempo de *setup* é diferente para cada máquina, independente de qual tarefa estava sendo processada e qual vai iniciar o seu processamento. Quando dependente da sequência de fabricação e da máquina, o tempo de *setup* é diferente para cada par de tarefas em cada uma das máquinas.

O tempo de *setup* dependente da máquina e da sequência de fabricação pode ser utilizado como um caso geral do problema com tempo de *setup* dependente apenas da sequência de fabricação ou dependente apenas da máquina, pois se pode considerar os tempos de *setup* de todas as máquinas iguais, dependentes apenas da sequência de fabricação, ou pode-se considerar o tempo de *setup* igual para todo par de tarefas, dependendo apenas da máquina.

Outra característica muito comum em ambientes *flow shop* é o bloqueio, que pode ocorrer em ambientes que não possuem estoque intermediário (*buffer* zero) ou que possuem estoque intermediário limitado (Figura 1). O bloqueio de uma máquina

ocorre toda vez que ela termina o processamento de uma tarefa, e o estoque intermediário entre essa máquina e máquina seguinte está cheio, impedindo que a tarefa saia da máquina. Enquanto a tarefa fica na máquina, esta fica impedida de receber a tarefa seguinte da sequência para iniciar seu processamento, diz-se então que ela está bloqueada. Na figura 1 é representado um emxemplo em um ambiente com *buffer* zero e *setup* dependente da tarefa e da máquina, nela a máquina *k* termina o processamento da tarefa *j*+1, porém a máquina *k*+1 ainda não está pronta para receber a tarefa, pois ela ainda está realizando o *setup*. Como não há um estoque intermediário, a máquina *k* fica impedida de iniciar o *setup* da tarefa *j*+2, ou seja, ela fica bloqueada até que a máquina *k*+1 termine o *setup* para poder iniciar o processamento da tarefa *j*+1.

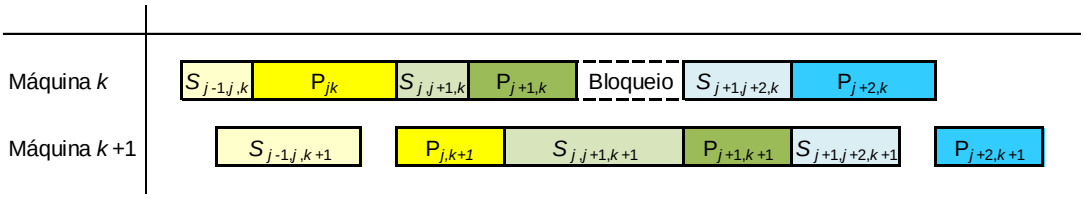

Figura 1 – Ocorrência do bloqueio em um ambiente *flow shop* permutacional

1.1. Motivação e Justificativa para Execução da Pesquisa

Segundo Maccarthy e Liu (1993), os métodos de programação da produção não têm sido muito explorados na prática, devido a dois principais problemas: 1. A teoria e os métodos de solução são desconhecidos ou não entendido pelos responsáveis por fazer os planejamentos da produção nas indústrias; 2. Os ambientes ideais assumidos nos problemas de sequenciamento não são suficientemente parecidos com os encontrados na prática.

O presente trabalho se baseia no segundo problema indicado por Maccarthy e Liu (1993), propondo o sequenciamento da produção em um ambiente *flow shop* com *n* tarefas e *m* máquinas e tempo de *setup* separado do tempo de processamento das tarefas e dependente da sequência e da máquina, sem estoque intermediário, podendo ocorrer bloqueio, que é um cenário bastante comum nas indústrias de manufatura, trabalhando com o objetivo de minimizar o tempo total de produção (ou *makespan*).

A importância de todo processo de melhoria da produção deve-se principalmente ao alto nível de competitividade nas indústrias, grande parte devido à globalização. A redução do *makespan* aumenta a capacidade de produção da fábrica e melhora a utilização dos equipamentos, auxiliando na redução dos custos de fabricação e tornando-a mais competitiva no mercado global (Pinedo, 2008).

O problema de sequenciamento da produção com tempo de *setup* separado do tempo de produção e dependente da sequência de fabricação e da máquina é demonstrado como sendo *NP*-Completo por Gupta e Darrow (1986), mesmo para problema em ambientes *flow shop* permutacional com apenas duas máquinas. Caraffa *et al.* (2001) e Ronconi (2004) descrevem o problema de sequenciamento da produção com *n* tarefas e *m* máquinas e com bloqueio como sendo *NP-Hard*. Os problemas tratados neste trabalho são problemas de sequenciamento da produção em ambientes *flow shop*, que não apenas possuem tempo de *setup* separado do tempo de processamento e dependente da sequencia de fabricação e da máquina, como também possuem bloqueio e oferecem a possibilidade de trabalhar com qualquer quantidade de tarefas e máquinas. Pode-se verificar, por comparação, que os problemas tratados no presente trabalho são problemas *NP*-Completo, exigindo bastante conhecimento e estudo por parte do autor.

Como foi apontado por Maccarthy e Liu (1993), uma das principais dificuldades encontradas quando se tenta aplicar o problema de sequenciamento em ambientes reais de produção é a grande diferença entre os ambientes ideais assumidos na teoria e os ambientes encontrados na prática. O presente trabalho contribui para que essa diferença seja cada vez menor, propondo um ambiente de produção bastante comum na prática.

O problema em questão é ainda muito pouco explorado na literatura. Conforme revisão da literatura realizada, não foram encontrados trabalhos que estudem o problema proposto neste trabalho. O presente trabalho pretende desenvolver novos métodos para o problema em específico, tornando este trabalho original e inovador. Contribuindo, desta forma, com novas abordagens para o problema que poderá também ser utilizado para o desenvolvimento de trabalhos futuros.

1.2. Objetivo Geral

O objetivo deste trabalho é desenvolver e avaliar três métodos de solução capazes de obter soluções para problemas de sequenciamento da produção em um ambiente *flow shop* permutacional com *setup* separado do tempo de processamento das tarefas e dependente da máquina e da sequencia de fabricação, com capacidade zero de estoque intermediário (*buffer* zero), podendo ocorrer o bloqueio nas máquinas. Os métodos de solução desenvolvidos são três: 1. Quatro limitantes inferiores para um algoritmo *Branch-and-Bound*; 2. Dois modelos *MILP* e a adaptação de outros dois modelos *MILP* e; 3. A adaptação de 28 métodos heurísticos construtivos.

1.3. Objetivos Específicos

Para se alcançar o objetivo proposto serão necessários:

- Revisar a bibliografia a respeito de trabalhos relacionados a problemas de sequenciamento da produção em ambientes *flow shop* com bloqueio;
- Elaborar um procedimento para o cálculo do *makespan* em ambientes *flow shop* com tempo de *setup* separado, dependente da sequencia de fabricação, e com bloqueio;
- Elaborar um limitante superior (*upper bound*) para o tempo de máquina ociosa e um limitante inferior (*lower bound*) para o tempo de bloqueio da máquina;
- Utilizar os limitantes superior e inferior para elaborar os limitantes inferiores (*lower bounds*) do *makespan* para resolver problemas em ambientes *flow shop* permutacional com tempo de *setup* separado, dependente da máquina e da sequencia de fabricação, e com bloqueio, com o objetivo de minimização do *makespan*;
- Utilizar os limitantes inferiores do *makespan* para formular o método *Branch-and-Bound* para o problema;

- Desenvolver os modelos de programação linear inteira mista (*MILP*) para resolver problemas em ambientes *flow shop* permutacional com tempo de *setup* separado, dependente da máquina e da sequencia de fabricação, e com bloqueio, com o objetivo de minimização do *makespan*;
- Adaptar os melhores métodos heurísticos construtivos de problemas mais simples para obterem a solução do *makespan* para o problema em ambientes *flow shop* permutacional com tempo de *setup* separado, dependente da máquina e da sequencia de fabricação, e com bloqueio, com o objetivo de minimização do *makespan*;
- Avaliar os métodos por meio de testes e comparação com outros métodos reportados/adaptados da literatura.

1.4. Estrutura do Trabalho

O trabalho é divido de forma a documentar todo o seu desenvolvimento. Inicia-se o primeiro capítulo com a introdução, fornecendo-se uma visão geral do que é estudado no trabalho, quais são os objetivos e os motivos do seu desenvolvimento.

No segundo capítulo é apresentada a forma de classificação dos problemas de sequenciamento da produção, definindo qual é o problema estudado neste trabalho. Apresentam-se também as expressões para o cálculo do *makespan* e os métodos de solução para o para o problema estudado.

No terceiro capítulo é apresentado o que tem sido apresentado na literatura para solução de problemas similares ao problema estudado neste trabalho, demonstrando a evolução dos métodos de solução.

No capítulo quatro são apresentadas duas propriedades estruturais do problema estudado: um limitante superior para o tempo de ociosidade; e um limitante inferior para o tempo de bloqueio.

No quinto capítulo é apresentado o método de pesquisa utilizado para os estudos, como são realizados os testes e como são avaliadas as soluções obtidas pelos métodos.

Nos capítulos seis, sete e oito são apresentados os modelos de solução desenvolvidos e os modelos de solução adaptados para o problema estudado. No

capítulo seis são apresentados os modelos *MILP*, no capítulo sete os modelos *Branch-and-Bound* e no capítulo oito os métodos heurísticos construtivos.

No capítulo nove são apresentados os resultados obtidos nos testes computacionais, bem como as comparações entre os métodos de solução. E, por fim, no capítulo 10 são apresentadas as conclusões do trabalho, o que se obteve de resultados e uma discussão de possíveis trabalhos futuros.

2. PROGRAMAÇÃO DA PRODUÇÃO

O problema da programação da produção diz respeito à alocação das tarefas nas máquinas em um ambiente de fabricação. O sequenciamento pode variar em tamanho e complexidade dependendo do tipo de ambiente onde ele é aplicado, o número e tipos de restrições tecnológicas e os critérios de avaliação do problema.

Graham *et al.* (1979) propuseram pela primeira vez um padrão de descrição de problemas de sequenciamento de produção baseado em três índices α, β e γ. O primeiro índice α descreve o tipo de ambiente de produção. O segundo índice β descreve as restrições tecnológicas do problema. O terceiro índice γ descreve qual é o critério de avaliação.

Os possíveis tipos de ambiente do **campo α** são (Pinedo, 2008; Maccarthy; Liu, 1993):

- **Máquina única** (1): É o tipo mais simples possível de ambiente. Há apenas uma única máquina para realizar todas as tarefas;
- **Máquinas paralelas** (*Pm*): Existem *m* máquinas em paralelo. Cada tarefa necessita ser realizada em apenas uma das máquinas e qualquer máquina pode fazer a mesma tarefa;
- ***Flow shop*** (*Fm*): Existem *m* máquinas, todas as tarefas devem seguir o mesmo padrão de fluxo (*i.e.* todas as tarefas devem ser realizadas primeiro na máquina 1, depois na máquina 2, e assim por diante);
- ***Job Shop*** (*Jm*) Existem *m* máquinas e cada tarefa tem um fluxo próprio de fabricação;
- ***Open Shop*** (*Om*): Existem *m* máquinas, todas as tarefas devem ser realizadas em todas as máquinas, porém nenhuma delas possui um fluxo de fabricação (*i.e.* cada tarefa deve passar por todas as máquinas, porém independem da ordem em que visitarão cada uma delas);
- ***Flow shop* com máquinas múltiplas** (*FFc*): Existem *c* estações de trabalho, cada uma com *g* máquinas em paralelo. Todas as tarefas devem seguir o mesmo padrão de fluxo;

- ***Job Shop* com máquinas múltiplas** (*FJc*): Existem *c* estações de trabalho, cada uma com *g* máquinas em paralelo. Cada tarefa tem um fluxo próprio de fabricação.

A relação existente entre cada tipo de ambiente é apresentada na Figura 2.

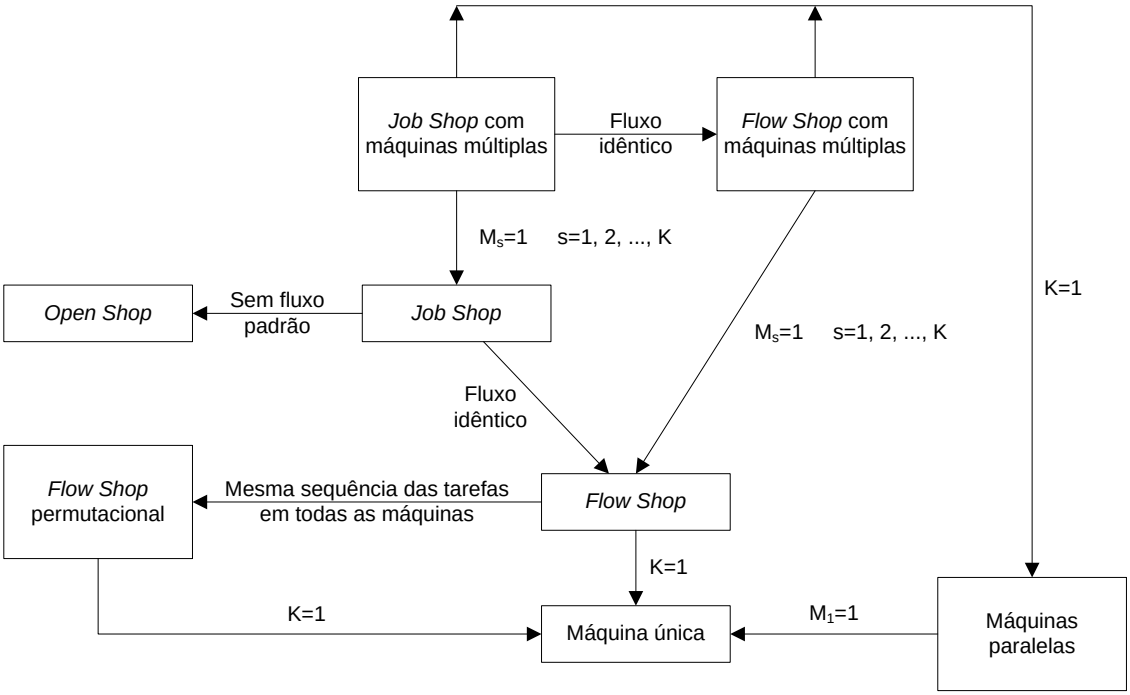

Figura 2 – Relação entre os diversos tipos de ambiente
Fonte: adaptado de Maccarthy e Liu (1993).

Algumas das possíveis restrições tecnológicas do campo β são (Pinedo, 2008):

- **Data de liberação** (r_j): Uma tarefa *j* qualquer não pode começar a ser realizada antes da sua data de liberação r_j;
- **Preempção** (*prmp*): Uma tarefa *j* qualquer não precisa necessariamente permanecer em uma máquina até ser completada, ela pode ser interrompida a qualquer momento e outra tarefa pode ser iniciada na máquina. Porém, a tarefa interrompida deve, em algum momento futuro, ser completada pela mesma máquina ou por uma outra máquina (em caso de máquinas paralelas);

- **Restrição de precedência** (*prec*): Uma tarefa *j* qualquer necessita que uma ou mais tarefas sejam realizadas antes que ela possa ser iniciada;
- ***Setup* dependente da sequência de fabricação** (S_{ijk}): S_{ijk} define o tempo de *setup* da tarefa *j*, precedida diretamente pela tarefa *i*, na máquina *k*;
- **Famílias de tarefas** (*fmls*): Existem *F* diferentes famílias de tarefas e cada uma delas possui *n* tarefas. Tarefas da mesma família podem possuir tempos de processamento diferentes entre si, porém, se realizadas uma após a outra não necessitam de parada para *setup*. Quando uma tarefa de outra família for ser realizada na máquina haverá necessidade de *setup*. Se o tempo de *setup* for dependente da sequência de fabricação, então é simbolizada por S_{ghk}, onde *g* é a família da tarefa que precede diretamente a tarefa da família *h* na máquina *k*. Se o tempo de *setup* depende apenas da família que vai começar a ser processada, então é simbolizada por S_{hk};
- **Processamento em lotes** (*batch*(*b*)): Uma única máquina consegue executar *b* tarefas ao mesmo tempo. O tempo de processamento das tarefas de um lote podem ser diferentes, porém, o lote inteiro só termina suas tarefas quando a tarefa mais demorada do lote estiver terminada. Ou seja, o tempo total de processamento do lote é igual ao tempo de processamento da tarefa mais demorada do lote;
- **Quebras** (*brkdwn*): Essa restrição indica que uma máquina pode não estar disponível o tempo todo;
- **Restrição de eligibilidade de máquinas** (q_j): Essa restrição pode existir em ambientes de máquinas em paralelo. Ela indica que nem todas as máquinas *q* de uma estação de trabalho podem executar uma tarefa *j* qualquer;
- **Permutação** (*prmu*): Essa restrição pode aparecer apenas em ambientes do tipo *flow shop*. Ela indica que o ambiente de fabricação segue a filosofia FIFO, ou seja, a ordem das tarefas se mantem sempre igual até o final;
- **Bloqueio** (*block*): O bloqueio pode ocorrer em ambientes do tipo *job shop* ou *flow shop*. Se um ambiente possui um estoque intermediário limitado entre duas máquinas, e este encontra-se cheio e a máquina subsequente

ocupada, então a máquina anterior a este estoque fica impossibilitada de liberar uma tarefa completa. Isso faz com que a máquina fique bloqueada, ou seja, impossibilitada de começar a ser preparada ou processar a tarefa seguinte. Um caso especial de bloqueio é o *buffer* zero, (*i.e.* quando não há estoque intermediário).;

- **Sem espera** (*nwt*): Essa restrição também ocorre em ambientes *flow shop*. Ambientes com essa restrição possuem tarefas que não podem ficar em espera entre máquinas sucessoras. Isso faz com que em alguns casos o tempo de início de uma tarefa na primeira máquina deva ser adiado para garantir que a tarefa possa passar por todas as máquinas sem ter que ficar esperando;
- **Recirculação** (*rcrc*): Essa restrição pode ocorrer em ambientes do tipo *Job Shop* ou *Job Shop* com máquinas múltiplas. Ela indica que uma tarefa pode ser processada em uma mesma máquina mais de uma vez.

Alguns possíveis critérios de avaliação do campo γ são (Pinedo, 2008):

- ***Makespan*** (C_{max}): Visa minimizar o tempo de conclusão da última operação da sequencia;
- ***Lateness* máxima** (L_{max}): Visa minimizar a pior violação das datas de entrega;
- **Tempo de fluxo** ($\sum C_j$): Tem por objetivo minimizar a soma do tempo de conclusão de todas as tarefas;
- **Tempo total ponderado de conclusão** ($\sum w_j C_j$): O peso da tarefa nesse caso é geralmente associado ao custo. O objetivo é, nesse caso, minimizar os custos de se manter as tarefas paradas (por exemplo, custo de manter as tarefas em estoques intermediários);
- **Tempo total ponderado de conclusão descontado** ($\sum w_j(1-e^{-rCj})$): O objetivo também é minimizar os custos de se manter as tarefas paradas, porém, neste caso o custo aumenta a uma taxa r ($0<r<1$) a cada unidade de tempo. Ou seja, quando a tarefa j é concluída no tempo t, o custo ponderado descontado de conclusão da tarefa j é $w_j(1-e^{-rt})$;

- **Tempo total ponderado de atraso** ($\sum w_j T_j$): Visa minimizar o custo do tempo de atraso de entrega das tarefas (por exemplo, multas por dia de atraso);
- **Quantidade ponderada de tarefas atrasadas** ($\sum w_j U_j$): Visa minimizar o custo de atraso das tarefas. Nesse caso o peso da tarefa é o custo de se atrasar a entrega de alguma tarefa independente do tempo de atraso.

Outros objetivos menos comuns podem ainda ser associados aos problemas de sequenciamento (por exemplo, tempo total ponderado de adiantamento, tempo total de adiantamento somado ao tempo total de atraso, tempo total ponderado de adiantamento somado ao tempo total ponderado de atraso). Além disso, podem existir problemas com múltiplos objetivos (por exemplo, minimizar simultaneamente o *makespan* e o tempo total ponderado de atraso).

Conforme a notação apresentada por Graham *et al.* (1979) o problema estudado neste trabalho é definido da seguinte forma:

- *Fm / prmu, S_{ijk}, block / Cmax*: Descreve um ambiente *flow shop* permutacional com *m* máquinas, com *setup* separado e dependente da sequência de fabricação e da máquina e com a possibilidade de haver bloqueio, onde o objetivo do problema é minimizar o *makespan*. Problema que será tratado neste trabalho;

2.1. Expressão do cálculo do *Makespan* para o problema estudado

O *makespan* é a data de término da última tarefa na última máquina (Pinedo, 2008) e, para o seu cálculo, deve-se inicialmente calcular a data de término do *setup* de cada máquina para cada tarefa (R_{jk}). Em alguns casos as máquinas necessitam realizar o *setup* mesmo para a primeira tarefa da sequencia (j=1). Nesses casos, considerando que todas as máquinas estão disponíveis ao início do processo (*i.e.* no tempo zero), o *setup* das máquinas pode ser realizado ao mesmo tempo em todas as máquinas para j=1, ou seja, a primeira tarefa da sequencia. Portanto o valor de

R_{1k} (data de término do *setup* da máquina para a primeira tarefa) é dado pela equação 1:

$$R_{1k} = S_{01k} \tag{1}$$

para $k = 1,..., m$.

Onde S_{01k} é o tempo de *setup* da primeira tarefa da sequencia na máquina *k*.

Para as demais tarefas *j* = {2, ..., *n*} o *setup* de cada máquina pode ser iniciado logo que a tarefa anterior termina de ser processada na máquina. Portanto o valor de R_{jk} para as demais tarefas pode ser obtido pela equação:

$$R_{jk} = C_{j-1,k} + S_{ijk} \tag{2}$$

para $j = 2,..., n \text{ e } k = 1,..., m$.

A data de término de processamento das tarefas na primeira máquina, considerando o tempo de bloqueio (ou seja, o momento que a tarefa libera a primeira máquina), pode ser obtida pela seguinte equação:

$$C_{j1} = \max\left(R_{j2}, R_{j1} + P_{j1}\right) \tag{3}$$

Nas máquinas $k = 2,..., m-1$:

$$C_{jk} = \max\left(R_{j,k+1}, C_{j,k-1} + P_{jk}\right) \tag{4}$$

E na última máquina:

$$C_{jm} = C_{j,m-1} + P_{jm} \tag{5}$$

O *makespan* ($C_{\max}$) nada mais é do que a data de término da última tarefa da sequência na última máquina, ou seja:

$$C_{\max} = C_{nm} \tag{6}$$

Para demonstrar os procedimentos para o cálculo do *makespan* considere o exemplo com quatro tarefas e três máquinas, ou seja, $n = 4$ e $m = 3$. Os tempos de processamentos (P_{jk}) das tarefas $j = [1, 2, 3, 4]$ nas máquinas $k = [1, 2, 3]$ são dados na Tabela 1.

Tabela 1 – Tempo de processamento das tarefas nas máquinas

P_{jk}	$j1$	$j2$	$j3$	$j4$
$k1$	5	3	3	*4*
$k2$	5	4	4	3
$k3$	3	2	5	3

Os tempos de *setup* (S_{ijk}) das tarefas $j = [1, 2, 3, 4]$ precedidas diretamente pelas tarefas $i = [1, 2, 3, 4]$ nas máquinas $k = [1, 2, 3]$ são apresentados na Tabela 2. Na tabela, o tempo de *setup* quando $i = j$ equivale ao tempo S_{0jk}, ou seja, o tempo de *setup* quando a tarefa *j* for a primeira tarefa da sequência.

Tabela 2 – Tempo de *setup* dos pares de tarefas nas máquinas

$k1$					$k2$					$k3$				
S_{ij1}	$j1$	$j2$	$j3$	$j4$	S_{ij2}	$j1$	$j2$	$j3$	$j4$	S_{ij3}	$j1$	$j2$	$j3$	$j4$
$i1$	4	15	15	13	$i1$	7	14	4	7	$i1$	5	5	8	15
$i2$	12	8	6	10	$i2$	11	1	10	6	$i2$	4	10	2	3
$i3$	3	1	5	8	$i3$	9	8	14	3	$i3$	7	8	8	9
$i4$	5	10	7	6	$i4$	3	4	7	9	$i4$	8	11	3	13

Considerando a sequência de fabricação $\sigma = [3, 1, 4, 2]$, com a equação 1 pode-se calcular a data de término do *setup* da primeira tarefa da sequência ($\sigma 1$) nas máquinas (R_{1k}). Para máquina $k1$

$$R_{11} = S_{011} = 5,$$

para máquina $k2$

$$R_{12} = S_{012} = 14,$$

e para máquina $k3$

$$R_{13} = S_{013} = 8.$$

Com os valores de R_{1k} pode-se calcular, por meio da equação 3, a data de conclusão do processamento (C_{jk}) da primeira tarefa ($\sigma 1$) na máquina $k1$ (C_{11}), que é dada por:

$$C_{11} = \max(R_{12}, R_{11} + P_{11}) = \max(14{,}5 + 3) = 14,$$

na máquina $k2$ (C_{12}), com a equação 4

$$C_{12} = \max(R_{13}, C_{11} + P_{12}) = \max(8{,}14 + 4) = 18,$$

e na máquina $k3$ (C_{13}), com a equação 5

$$C_{13} = C_{12} + P_{13} = 18 + 5 = 23.$$

Definidos os valores de C_{1k} é possível calcular a data de término do *setup* da segunda tarefa ($\sigma 2$) nas máquinas (R_{2k}), utilizando a equação 2. Para máquina $k1$

$$R_{21} = C_{11} + S_{121} = 14 + 3 = 17,$$

para máquina $k2$,

$$R_{22} = C_{12} + S_{122} = 18 + 9 = 27,$$

e para máquina $k3$

$$R_{23} = C_{13} + S_{123} = 23 + 7 = 30.$$

Com as equações 3, 4 e 5 podem-se calcular C_{jk} da segunda tarefa da sequência ($\sigma 2$) nas máquinas $k1$, $k2$ e $k3$ respectivamente.

$$C_{21} = \max(R_{22}, R_{21} + P_{21}) = \max(27{,}17 + 5) = 27,$$
$$C_{22} = \max(R_{23}, C_{21} + P_{22}) = \max(30{,}27 + 5) = 32, \text{ e}$$
$$C_{23} = C_{22} + P_{23} = 32 + 3 = 35.$$

De maneira análoga são realizados os cálculos para a terceira e a quarta tarefa da sequência ($\sigma 3$ e $\sigma 4$). Inicialmente, os cálculos para R_{3k}.

$$R_{31} = C_{21} + S_{231} = 27 + 13 = 40,$$
$$R_{32} = C_{22} + S_{232} = 32 + 7 = 39, \text{ e}$$
$$R_{33} = C_{23} + S_{233} = 35 + 15 = 50.$$

Em seguida, os cálculos para C_{3k}.

$$C_{31} = \max(R_{32}, R_{31} + P_{31}) = \max(39{,}40 + 4) = 44,$$
$$C_{32} = \max(R_{33}, C_{31} + P_{32}) = \max(50{,}44 + 3) = 50, \text{ e}$$
$$C_{33} = C_{32} + P_{33} = 50 + 3 = 53.$$

Seguido dos cálculos para R_{4k}.

$$R_{41} = C_{31} + S_{341} = 44 + 10 = 54,$$
$$R_{42} = C_{32} + S_{342} = 50 + 4 = 54, \text{ e}$$

$$R_{43} = C_{33} + S_{343} = 53 + 11 = 64.$$

Finalizando com os cálculos para C_{4k}.

$$C_{41} = \max(R_{42}, R_{41} + P_{41}) = \max(54{,}54 + 3) = 57,$$
$$C_{42} = \max(R_{43}, C_{41} + P_{42}) = \max(64{,}57 + 4) = 64, \text{ e}$$
$$C_{43} = C_{42} + P_{43} = 64 + 2 = 66.$$

Utilizando a equação 6, pode-se calcular o *makespan* ($C_{\max}$).

$$C_{\max} = C_{43} = 66$$

Na Figura 3 é apresentado o gráfico de GANTT resultante da sequência, nela é possível verificar os valores calculados para R_{jk}, C_{jk} e $C_{\max}$.

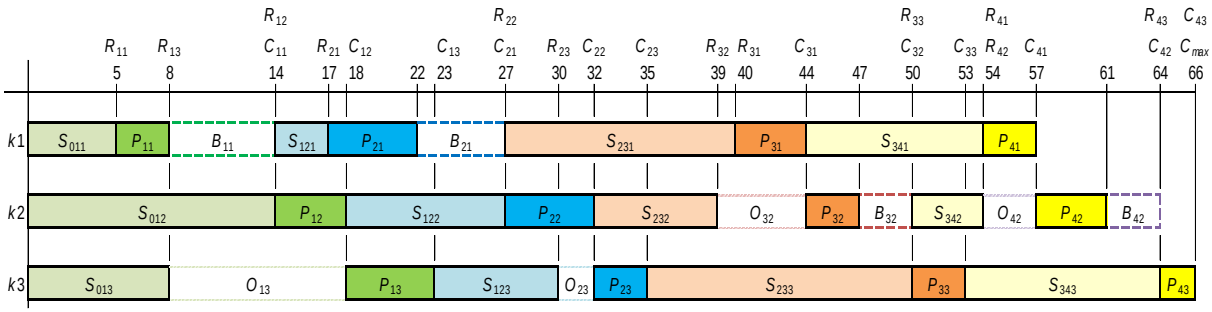

Figura 3 – Gráfico de GANTT para o problema exemplo

Para se obter as soluções viáveis ou ótimas dos problemas de sequenciamento existem diversos métodos que podem ser utilizados. Uma revisão de alguns métodos de solução é apresentada no subitem 2.2 a seguir.

2.2. Métodos de Solução

A pesquisa operacional (PO) é a aplicação de métodos científicos com a finalidade de se obter as soluções de um determinado problema que melhor satisfaçam os objetivos de uma organização como um todo (Ackoff; Sasieni, 1971). Segundo Hillier e Lieberman (1974), a PO ganhou destaque durante a segunda guerra mundial e foi inserida no contexto industrial logo após seu término, devido a dificuldades enfrentadas pelas empresas causadas pelo aumento da competitividade.

Para o problema de PO é necessário criar o modelo matemático que geralmente assume forma de equações ou inequações (Ackoff; Sasieni, 1971; Hillier; Lieberman, 1974). A função objetivo é a equação que fornece o valor da variável que se quer otimizar. As restrições do problema são equações, ou inequações, que garantem que os resultados obtidos sejam os mais próximos do real. Segundo Hillier e Lieberman (1974) a ideia do modelo de PO é maximizar (ou minimizar) a função objetivo, ao mesmo tempo que todas as restrições são respeitadas.

Para se construir um modelo de PO é necessário primeiro identificar e definir qual é o problema abordado, em seguida analisar e coletar informações referentes a esse problema, para posteriormente construir a representação matemática do problema. Com o modelo matemático definido é necessário encontrar uma solução para o problema. Esta solução deve ser tanto quanto possível a melhor solução. Ackoff e Sasieni (1971) evidenciam que o modelo matemático nunca é uma representação perfeita do problema, pois podem existir fatores que, apesar de importantes, são difíceis de serem quantificados (por exemplo, aspectos comportamentais). Portanto a solução ótima alcançada deve ser testada e analisada antes de ser implementada, mensurando também o impacto causado pela falta desses fatores não considerados no modelo. Assim como todo método, os resultados obtidos após a implementação da PO também devem ser acompanhados para garantir a sustentabilidade das melhorias implantadas. Ackoff e Sasieni (1971) definem que as etapas para a construção de modelos de PO são:

- Definição da situação-problema;
- Formulação de um modelo matemático;

- Solução do modelo e obtenção da melhor solução;
- Teste do modelo e avaliação da solução;
- Consideração dos fatores imponderáveis;
- Implementação e acompanhamento da solução (manutenção).

A solução do modelo pode ser realizada de diversas formas. Os métodos aplicados podem ser heurísticos ou exatos.

Segundo Montevechi *et al.* (2002) os métodos heurísticos geram soluções aceitáveis sem grandes esforços computacionais. Entretanto o método exato, utiliza as equações e inequações do modelo matemático do problema, fazendo a procura pela solução ótima.

2.2.1. Métodos heurísticos

Os métodos heurísticos para problemas de sequenciamento da produção podem ser compostos, segundo Framinan *et al.* (2004), em três fases distintas:

- **Fase I**: Desenvolvimento do índice;
- **Fase II**: Construção da solução;
- **Fase III**: Melhoria da solução.

Segundo Framinan *et al.* (2004) cada método pode ser composto de uma ou mais destas fases. As três fases são descritas a seguir.

Fase I: Desenvolvimento do índice

Nesta fase é selecionada, em um conjunto de tarefas, aquela tarefa que terá prioridade de processamento, com base em informações das tarefas (Framinan *et al.*, 2004).

Essa seleção das prioridades das tarefas é feita por meio de regras de prioridade. No caso do sequenciamento, as regras de prioridade consistem em

especificar um padrão para selecionar a ordem em que as tarefas devem ser realizadas e onde cada tarefa deverá ser processada. Existem diferentes regras de prioridade que podem ser usadas para o sequenciamento da produção, porém existem regras que são mais apropriadas para cada tipo de problema, visto que ainda não existe nenhuma regra de prioridade que seja eficiente para todas as situações (Tubino, 2007).

Podem ser citados como exemplos de regras classificadas nesta fase:

- ***Shortest Processing Time*** (SPT): Dado um conjunto de *n* tarefas a serem realizadas em *m* máquinas, a regra de prioridade SPT ordena-as conforme a ordem não decrescente dos seus tempos de processamento. A regra de prioridade SPT tende a minimizar o tempo médio de fluxo em problemas de máquina única e de máquinas paralelas;
- ***Longest Processing Time*** (LPT): Dado um conjunto de *n* tarefas a serem realizadas em *m* máquinas, a regra de prioridade LPT as ordena conforme a ordem não crescente dos seus tempos de processamento. A regra de prioridade LPT tende a equilibrar a carga de trabalho das máquinas em problemas de máquinas paralelas;
- **Regra de Johnson** (Johnson, 1954): Dado um conjunto de *n* tarefas a serem realizadas em duas máquinas, a regra de Johnson ordena-as da seguinte forma: primeiramente procura-se dentre todos os tempos de processamento o menor valor de tempo de processamento; se este valor for correspondente ao tempo de processamento da primeira máquina, a tarefa correspondente é ordenada como a primeira tarefa a ser realizada; se este valor for correspondente ao tempo de processamento da segunda máquina, a tarefa correspondente é ordenada como a última tarefa a ser realizada; os tempos de processamento da tarefa selecionada são eliminados em ambas as máquinas; o processo é repetido até que todas as tarefas tenham sido ordenadas.

Fase II: Construção da solução

Nesta fase a solução é construída por meio de tentativas de se inserir uma ou mais tarefas em uma ou mais posições de um sequenciamento parcial até que

todas as tarefas tenham sido alocadas em uma posição (Framinan *et al.*, 2004). As tarefas são divididas em dois subconjuntos: as tarefas já sequenciadas; e as tarefas que ainda não foram sequenciadas. Nesta fase deve-se definir qual tarefa será inserida na sequencia parcial e em que posição da sequencia parcial ela será inserida.

A primeira decisão, ou seja, a seleção da tarefa que será inserida na sequencia parcial pode ser feita de duas formas:

1. De acordo com o índice formado na fase I (por exemplo, as tarefas são selecionadas na ordem em que elas foram ordenadas na sequencia pela regra de prioridade);
2. Várias tarefas são testadas e a escolha de qual tarefa será inserida na sequencia parcial depende de uma determinada regra pré-estabelecida: por exemplo, uma tarefa é posicionada em uma determinada posição da sequencia parcial de um problema, calcula-se a somatória dos tempos de ociosidade que a alocação desta tarefa gera naquela posição; posteriormente uma segunda tarefa é posicionada nesta mesma posição da sequencia parcial do problema, novamente calcula-se a somatória dos tempos de ociosidade que a alocação desta tarefa gera naquela posição; este procedimento é repetido até que todas as tarefas tenham sido testadas; a tarefa que gerou a menor somatória dos tempos de ociosidade é selecionada.

A segunda decisão, ou seja, a posição a qual a tarefa selecionada será inserida pode ser: fixa (por exemplo, a *j*-ésima tarefa escolhida para ser inserida na sequencia parcial do problema é posicionada na *j*-ésima posição da sequencia); ou pode ser determinada após diversos testes. Assim como na seleção da tarefa a ser inserida, para determinar a posição onde a tarefa será inserida, uma regra de prioridade deve ser definida, por exemplo, a posição em que a tarefa seja posicionada que proporcione o menor *makespan* para a sequência parcial das tarefas.

Fase III: Melhoria da solução

Nesta fase uma solução já existente (solução inicial) é melhorada por meio de um procedimento. Segundo Stützle (1998), os métodos de melhoria para o problema de minimização do *makespan* podem ser divididos em busca local descendente e meta-heurísticos.

A busca local descendente percorre a vizinhança de uma ou mais soluções iniciais, buscando outra solução com melhor valor. Se uma solução melhor for encontrada, esta é tomada como sendo a nova solução inicial e o processo recomeça.

As meta-heurísticas são métodos de melhoria, onde são gerados uma ou mais soluções, a partir de uma ou mais soluções iniciais, que são comparadas entre si e testadas para verificação quanto à sua factibilidade. Algumas meta-heurísticas são:

- ***Cluster Searching*** (Oliveira; Lorena, 2004) emprega agrupamento (*clustering*) para localizar áreas promissoras de busca. Uma área pode ser vista como um subespaço de busca definido por uma relação de vizinhança no espaço de soluções. A área de busca é inicialmente posicionada aleatoriamente no espaço de busca, e com o tempo ela tende a se ”movimentar” em direção aos subespaçoes mais promissores da proximidade;
- ***Simulated Annealing*** (Kirkpatrick *et al*., 1983) é uma técnica de busca probabilística baseado em uma analogia com a mudança de estado do material quando resfriado após ser aquecido acima da sua temperatura de recristalização, equivalente a um processo de têmpera ou de recozimento;
- **Busca Tabu** (Glover, 1986) é um método de busca que se utiliza de uma lista chamada de "Lista Tabu", a qual tem a finalidade de evitar que o programa fique preso em um máximo ou mínimo local;
- **Algoritmo Genético** (Goldberg, 1989) é um método de busca probabilística baseado na teoria da evolução natural, onde o mais “forte” possui mais chances de sobreviver. O método utiliza conceitos de pais, filhos, clonagem, crossover, mutação e população, entre outros;

- **Colônia de formiga** (Dorigo, 1992) se baseia na tentativa de reprodução da habilidade das formigas, quase cegas, em encontrar o caminho mais curto entre seus ninhos e a comida;
- ***Particle Swarm Optimization*** (Kennedy; Eberhart, 1995), assim como os algoritmos genéticos, consiste numa abordagem de Computação Evolucionária que tenta imitar a "evolução" de uma população de indivíduos. Esta técnica se baseia no comportamento social de pássaros e peixes para controlar a geração de "populações" de soluções.

2.2.2. Métodos exatos

Segundo Hillier e Lieberman (1974) o método exato pode ser classificado como:

- **Problemas envolvendo Programação Linear** (*LP*[1]): são problemas nos quais as expressões matemáticas (representandas pela função objetivo e as restrições) envolvidas são lineares e onde todas as variáveis podem assumir valores não inteiros reais;
- **Problemas envolvendo Programação Inteira** (*IP*[2]): são problemas onde todas as variáveis de decisão só podem assumir valores inteiros e as expressões matemáticas são lineares;
- **Problemas envolvendo Programação Linear-Inteira Mista** (*MILP*[3]): são problemas similares aos de programação inteira, mas onde algumas variáveis de decisão podem assumir valores não inteiros;
- **Problemas envolvendo Programação Não-Linear** (*NLP*[4]): são problemas que possuem ao menos uma expressão matemática (a função objetivo ou alguma das restrições) não linear (ou seja, com produto de

---

1 *Linear Programming*
2 *Integer Programming*
3 *Mixed Integer-Linear Programming*
4 *Non-Linear Programming*

variáveis ou variáveis com expoente) e onde todas as variáveis podem assumir valores não inteiros reais;

- **Problemas envolvendo Programação Não-Linear Inteira Mista** (*MINLP*[5]): são problemas similares à Programação Não-Linear, mas que possuem variáveis de decisão que devem assumir valores inteiros e outras variáveis de decisão que podem assumir valores não inteiros.

Diversas técnicas para a solução de problemas envolvendo programação matemática são conhecidas. Cada modelo (*LP*, *IP*, *MILP*, *NLP* e *MINLP*) possui técnicas diferentes para encontrar a solução ótima do problema. A seguir são descritos dois métodos para solução de problemas de programação *MILP*: o método Simplex; e o método *Branch-and-Bound*.

O método Simplex

O método Simplex é uma metodologia que envolve uma sequencia de cálculos por meio dos quais é possível encontrar a solução ótima de um problema de programação linear (Hillier; Lieberman, 1974).

O método, segundo Hillier e Lieberman (1974), se baseia em três propriedades dos problemas lineares: 1. Se existe uma (ou mais) solução ótima, ela estará com certeza em um ponto extremo da região factível; 2. Uma solução ótima pode ser obtida por cálculos exaustivos; e 3. Se a solução encontrada em um ponto extremo da região factível é melhor que o resultado obtido por todos os pontos extremos adjacentes a ele, então esta é a solução ótima para o problema.

O método opera da seguinte forma (Hillier; Lieberman, 1974):

- **Início**: um ponto extremo da região factível é selecionado para se iniciar;
- **Iterações**: movimentar para um ponto extremo adjacente que obtenha uma solução melhor (este passo é repetido quantas vezes forem necessárias);

---

[5] *Mixed Integer Non-Linear Programming*

- **Critério de parada**: parar quando a solução obtida pelo ponto extremo adjacente for melhor que as soluções obtidas por todos os pontos extremos adjacentes.

O método Branch-and-Bound

O método *Branch-and-Bound* (B&B) substitui um problema original por um conjunto de subproblemas. Segundo Ignall e Schrage (1965) um problema original é escrito em uma árvore de soluções, na qual cada nó representa um limitante inferior da função objetivo.

Segundo Ríos-Mercado e Bard (1999) o método *Branch-and-Bound* (B&B) possui três características:

- Uma **regra de ramificação** (*branching*): que define como será realizada a divisão da árvore em nós (*i.e*. como será definido o conjunto de subproblemas em cada nó);
- Um **limitante inferior** (*bound*): que define um limitante inferior para cada subproblema;
- Uma **estratégia de busca**: que define quais nós serão ramificados, com o intuito de reduzir o tempo computacional.

No capítulo seguinte será apresentado de forma evolutiva o que se tem feito para solucionar o problema de sequenciamento da produção, baseado na literatura,.

3. O PROBLEMA DE SEQUENCIAMENTO EM AMBIENTE *FLOW SHOP* COM BLOQUEIO

No problema objeto desta pesquisa, o bloqueio de máquina deverá ocorrer sempre que a máquina do estágio subsequente não estiver disponível, ou seja, trata-se do caso conhecido como *buffer* zero. Esse tipo de sistema produtivo pode existir tanto em *Job Shops* quanto em *Flow Shops*. A maior parte dos trabalhos reportados na literatura considera os modelos de *flow shop* nos quais os tempos de *setup* são incluídos nos tempos de processamento das tarefas, além de uma única máquina em cada estágio de produção.

Dentre os trabalhos mais importantes e pioneiros que foram reportados na literatura, pode ser citado o trabalho de Reddi e Ramamoorthy (1972), onde os autores mostram que o problema de programação *flow shop* tradicional com somente duas máquinas (dois estágios de produção) e a restrição de bloqueio de máquinas pode ser transformado em um caso especial do problema do Caixeiro-Viajante (*TSP*[6]).

Papadimitriou e Kanellakis (1980) provaram que o problema com capacidade de armazenagem de somente uma tarefa (*buffer* limitado) entre operações sucessivas é *NP-hard*.

A intensificação de trabalhos sobre *flow shop* com bloqueio de máquinas ocorreu a partir do final dos anos 1980.

McCormick *et al.* (1989) apresentaram um método heurístico, denominado *Profile Fitting* (*PF*), o qual procura sequenciar as tarefas de forma a minimizar a soma dos tempos de máquina parada, assim como dos tempos de bloqueio das máquinas.

Leisten (1990) apresentou dois métodos heurísticos para problemas *flow shop* permutacionais e não-permutacionais com *buffer* limitado. A medida de desempenho era maximizar a utilização dos buffers e minimizar o tempo de bloqueio das máquinas. Uma experimentação computacional mostrou que uma adaptação do tradicional método *NEH* (Nawaz-Enscore-Ham) criado por Nawaz *et al*. (1983),

---

[6] *Traveling Salesman Problem*

originalmente desenvolvido para o problema com buffer ilimitado, teve um desempenho melhor que os métodos propostos.

Hall e Sriskandarajah (1996), com base no resultado obtido por Papadimitriou e Kanellakis (1980), mostraram que o problema *flow shop* tradicional com três estágios de produção e restrição de bloqueio de máquinas é um problema fortemente NP-completo. Nesse mesmo trabalho, os autores relatam os principais trabalhos já então reportados na literatura.

Nowicki (1999) desenvolveu uma meta-heurística de Busca Tabu (*TS*[7]) que utiliza uma generalização da noção de bloco de tarefas que foi sugerida por Grabowski *et al*. (1983).

Norman (1999) abordou o problema *flow shop* permutacional com estoque intermediário limitado e *setup* dependente da sequencia de fabricação e da máquina com o objetivo de minimizar o *makespan*. Foram apresentados dois métodos heurísticos construtivos adaptados: *NEH* (Nawaz *et al*., 1983) e *PF* (McCormick *et al.*, 1989), e uma meta-heurística de busca tabu para a solução do problema. Aos métodos heurísticos construtivos foi ainda adicionado um algoritmo iterativo *greedy* (“guloso”). Foram gerados 900 problemas para a avaliação dos métodos propostos, variando tempo de *setup*, tamanho dos estoques intermediários, número de tarefas e número de máquinas. Este foi o primeiro trabalho encontrado na literatura a abordar esse problema.

Abadi *et al.* (2000) propuseram um método heurístico para minimizar o tempo de ciclo em um *flow shop* com bloqueio de máquinas, utilizando a correlação entre este problema e o problema de programação *flow shop* com restrição sem espera na execução das tarefas.

Armentano e Ronconi (2000) apresentaram a minimização do tempo total de atraso utilizando a Busca Tabu, um dos primeiros trabalhos a apresentar solução para este tipo de problema.

Ronconi e Armentano (2001) apresentaram algoritmos *Branch-and-Bound* para a solução do problema. Foi proposto um novo limitante inferior (*lower bound*) para as datas de término das tarefas. A partir daí, foram obtidos limitantes inferiores para o atraso total e o *makespan*.

---

[7] *Tabu Search*

Caraffa *et al.* (2001) desenvolveram um Algoritmo Genético para problemas *flow shop* de grande porte com restrições de fluxo (na literatura conhecidos como *Restricted Slowdown Flow Shop Problems*), para os quais o problema com bloqueio de máquinas foi um caso especial.

Ronconi (2004) abordou o problema com capacidade zero de armazenagem de tarefas entre operações sucessivas (*buffer* zero), com o objetivo de minimizar a duração total da programação (*makespan*). Um método heurístico construtivo que utiliza características específicas do problema é apresentado. O novo método, combinado com outros existentes na literatura, é comparado com o conhecido método heurístico *NEH* (Nawaz *et al.*, 1983) originalmente desenvolvido para o problema *flow shop* com *buffer* ilimitado, apresentando desempenho superior.

Ronconi (2005) apresentou um algoritmo *Branch-and-Bound*, no qual foram utilizados novos limitantes inferiores, os quais utilizam de maneira vantajosa a natureza e estrutura do problema com bloqueio de máquinas. O desempenho do novo método foi melhor do que aqueles propostos em Ronconi e Armentano (2001).

Martinez *et al*. (2006) trataram do problema *flow shop* tradicional com *buffer* zero e com o objetivo de minimizar o *makespan*. Os autores apresentaram um novo tipo de restrição de bloqueio de máquinas, o qual utilizado com a restrição clássica proporcionou a solução de problemas específicos de quatro estágios de produção em tempo polinomial.

Bagchi *et al*. (2006) fizeram uma revisão dos métodos baseados no *TSP* (*Traveling Salesman Problems*) para solucionar problemas *flow shop*. Os autores demonstram que diversos problemas *flow shop*, mesmo que bastante complexos, podem ser modelados como problemas de *TSP* e então solucionados por algoritmos disponíveis para eles.

Companys e Mateo (2007) estudaram dois problemas, o *flow shop* permutacional e o *flow shop* permutacional com bloqueio, ambos com o objetivo de minimizar o *makespan*. Inicialmente, os autores sugerem a heurística denominada *NEH*+, que é a aplicação de uma busca local *NEDM-RCT* (*Non-Exhaustive Descent Method with Random Consideration of Ties*) juntamente com a heurística *NEH* (Nawaz *et al*., 1983). Quando testado por meio de uma base de dados criada pelos autores, a aplicação da busca local melhorou significativamente os resultados tanto para problemas *flow shop* permutacional como para problemas *flow shop* permutacional com bloqueio. Em seguida foi proposto o uso do algoritmo de duplo

Branch-and-Bound LOMPEN (*Lomnicki Pendular algorithm*) usando a heurística *NEH+* para se obter a solução inicial. Por fim foi proposto o uso de 10 diferentes heurísticas para compor a solução inicial para o *LOMPEN*. Para os testes computacionais foram utilizados a base de dados de Taillard (1993) e outra base de dados criada pelos autores.

Grabowski e Pempera (2007) desenvolveram meta-heurísticas de Busca Tabu para a minimização do *makespan*. Algumas propriedades relacionadas ao bloqueio das máquinas foram apresentadas, discutidas e utilizadas nos métodos propostos. Experimentalmente, os autores mostraram que os métodos propostos tiveram um desempenho relativo superior na busca de melhores soluções, com pequeno esforço computacional, quando comparados com métodos já reportados na literatura.

Ronconi e Henriques (2009) trataram do problema *flow shop* com bloqueio de máquinas (*buffer* zero), com o objetivo de minimizar o atraso total na execução das tarefas. Além de um método heurístico construtivo que utiliza características específicas do problema, foi apresentado um método *GRASP* (*Greedy Randomized Adaptive Search Procedure*). Foram efetuados testes computacionais envolvendo os métodos propostos, uma adaptação do clássico método *NEH* (Nawaz, *et al*., 1983) e um algoritmo *Branch-and-Bound*. Os resultados experimentais mostraram que os novos métodos são promissores.

Liu e Kozan (2009) trataram do problema *flow shop* com bloqueio combinado, com objetivo de minimizar o *makespan*. No problema tratado as condições de *buffer* entre cada uma das máquinas é diferente (*buffer* limitado, *buffer* zero e *buffer* ilimitado). Inicialmente foi proposta pelos autores uma heurística construtiva denominada Liu-Kozan para solucionar o problema. Posteriormente a heurística construtiva foi combinado ao método *NEH* (Nawaz *et al*., 1983) para melhorar os resultados. Os métodos propostos foram capazes de gerar resultados para o problema, mostrando-se bastante versátil, por conseguir tratar de problemas com diferentes tipos de *buffer*.

Wang *et al*. (2010a) propuseram um algoritmo evolutivo híbrido *HDDE* (*Hybrid Discrete Differential Evolution*), com o objetivo de minimização do *makespan*. Por meio de uma experimentação computacional, utilizando os problemas-teste de Taillard (1993), os autores mostraram que o algoritmo proposto não só obtém melhores soluções do que as meta-heurísticas de Busca Tabu e Busca Tabu com

multi movimentos (*TS+M*[8]) apresentadas por Grabowski e Pempera (2007), como também superam o algoritmo *HDE* (*Hybrid Differential Evolution*) desenvolvido por Qian *et al.* (2009) em termos de qualidade de solução, robustez e eficiência de busca. Além disso, 112 das 120 melhores soluções obtidas por Grabowski e Pempera (2007) e Ronconi (2005) são melhoradas pelo algoritmo híbrido *HDDE* proposto.

Wang *et al*. (2010b) apresentaram três algoritmos híbridos *HS* (*Harmony Search*) que foram denotados por *hHS* (*Hybrid Harmony Search*), *hgHS* (*Hybrid Globalbest Harmony Search*), e *hmgHS* (*Hybrid Modified Globalbest Harmony Search*), para o problema *flow shop* com bloqueio de máquinas e minimização do tempo total de fluxo (ou, equivalentemente, o tempo médio de fluxo). Originalmente, o algoritmo *HS* (Harmony Search) foi proposto por Geem *et al*. (2001), sendo considerado como a mais recente extensão dos Algoritmos Genéticos, apresentando simplicidade matemática e uma evolução (convergência para as melhores soluções) mais rápida. Os métodos propostos foram avaliados e comparados por meio de uma extensa experimentação computacional, utilizando problemas-teste reportados na literatura. Os resultados experimentais mostraram que os métodos propostos apresentam boas soluções, especialmente o algoritmo *hmgHS*.

Gong *et al.* (2010) trataram do problema *flow shop* com duas máquinas, onde a primeira máquina produz por lotes e a segunda produz individualmente, com bloqueio e *setup* compartilhado e não antecipatório. A função objetivo do problema foi minimizar a soma ponderada do *makespan* e do tempo de bloqueio. Inicialmente os autores apresentam um modelo em programação quadrática inteira mista (*QMIP*[9]) para a solução do problema e comprovam que se trata de um problema fortemente *NP*-Hard. Em seguida são apresentadas pelos autores duas heurísticas baseadas em regras para solucionar duas variantes do problema principal (um caso onde não há bloqueio e outro onde a sequência é predeterminada). Para os testes foram gerados aleatoriamente 2640 problemas, alterando número de tarefas, tamanho do lote, tempo de *setup* e o fator de peso. Os problemas de pequeno porte foram resolvidos com o *QMIP* e com as heurísticas, já os problemas maiores foram

---

[8] *Tabu Search with Multi-Moves*
[9] *Quadratic Mixed Integer Program*

solucionados apenas com as heurísticas. As heurísticas foram capazes de gerar bons resultados apresentando também eficiência computacional.

Wang *et al.* (2011) adaptaram o algoritmo *hmgHS* (*Hybrid Modified Globalbest Harmony Search*) mencionado no trabalho de Wang *et al*., (2010b), agora para minimização do *makespan* em um *flow shop* com bloqueio de máquinas. Por meio de simulação computacional e comparação com outros métodos existentes, os autores mostraram a superioridade do método proposto em termos de qualidade da solução.

Ribas *et al.* (2011) apresentaram um algoritmo iterativo *greedy* (“guloso”) para minimização do *makespan* em um *flow shop* com bloqueio de máquinas. Além do algoritmo *IG* (*Iterated Greedy*), os autores apresentaram um método heurístico de melhoria baseado no método *NEH* (Nawaz *et al.*, 1983), o qual é utilizado como procedimento de solução inicial para o algoritmo *IG*. Os desempenhos dos métodos apresentados foram avaliados utilizando problemas-teste de Taillard (1993). Os resultados experimentais mostraram a eficiência do método proposto quando comparado com os melhores métodos reportados na literatura. As soluções de alguns problemas-teste utilizados na experimentação computacional foram melhoradas.

Pan *et al*. (2011a) apresentaram um algoritmo híbrido *HDDE* (*Hybrid Discrete Differential Evolution*) associado às buscas locais de inserção e permutação para solucionar problemas *flow shop* com *buffer* intermediário, com objetivo de minimizar o *makespan*. Utilizando os problemas-teste de Taillard (1993) o algoritmo foi comparado ao *P_DDE* proposto por Pan *et al.* (2008), ao *W_DDE* apresentado por Wang *et al.* (2010a), ao *HGA* (*Hybrid Genetic Algorithm*) proposto por Wang *et al.* (2006) e ao *HPSO* (*Hybrid Particle Swarm Optimization*) proposto por Liu *et al.* (2008). O algoritmo obteve ótimos resultados, principalmente para problemas maiores, e com tempos computacionais próximos aos demais métodos comparados.

Ainda em 2011 Pan *et al*. (2011b) propuseram um algoritmo híbrido *HS* (*Harmony Search*) denominado *CHS* (*Chaotic Harmony Search*) para solucionar problemas *flow shop* permutacional com *buffer* limitado, com o objetivo de minimizar o *makespan*. O método proposto foi avaliado e comparado com os algoritmos *IHS* (*Improved Harmony Search*), *GHS* (*Globalbest Harmony Search*), *ACO* (*Ant Colony Optimization*), *HPSO* (*Hybrid Particle Swarm Optimization*) e o *HGA* (*Hybrid Genetic*

Algorithm). Os resultados experimentais mostraram que o método proposto apresenta boas soluções.

Pan e Wang (2012) trataram do problema tradicional de programação *flow shop* com bloqueio de máquinas, com o objetivo de minimizar o *makespan*. Inicialmente, os autores apresentaram dois métodos heurísticos construtivos simples, denominados *wPF* e *PW*, ambos baseados no procedimento *PF* (*Profile Fitting*) introduzido por McCormick *et al*. (1989), e explorando as características do problema de bloqueio. Em seguida foram propostos três métodos heurísticos construtivos de "melhoria" (II fase), denominados *PF-NEH*, *wPF-NEH*, e *PW-NEH*, combinando os métodos inicialmente mencionados com o procedimento de enumeração do clássico método *NEH* (Nawaz *et al.*, 1983). Finalmente, utilizando o método de busca local baseado em inserção de tarefas (*LS*[10]) são desenvolvidos três métodos heurísticos "compostos" que foram denotados por *PF-NEH*/*LS*, *wPF-NEH*/*LS* e *PW-NEH*/*LS*. Os novos métodos propostos foram avaliados e comparados com métodos existentes por meio de uma simulação computacional utilizando os problemas-teste de Taillard (1993). Os resultados experimentais mostraram que os métodos construtivos apresentados tiveram um desempenho bem melhor do que os métodos já reportados na literatura. Além disso, os métodos compostos conseguiram melhorar de maneira significativa a qualidade das soluções obtidas pelos métodos construtivos. A experimentação computacional forneceu melhores soluções para 17 problemas-teste de grande porte, do banco de dados utilizado.

Trabelsi *et al*. (2012) apresentaram uma heurística *TSS* para solucionar problemas *flow shop* com bloqueio misto, com o objetivo de minimizar o *makespan*. Os autores apresentaram também uma adaptação da heurística *NEH* para o problema. Ambos os métodos foram utilizados com a meta-heurística *GA* (*Genetic Algorithm*) para solucionar os 100 problemas gerados pelos autores. Os resultados obtidos com as heurísticas em conjunto com o *GA* foram comparados com as já conhecidas soluções ótimas dos problemas e obtiveram bons resultados com bons tempos computacionais.

Han *et al.* (2012a) propuseram um algoritmo *DABC* (*Discrete Artificial Bee Colony*) para minimizar o tempo total de fluxo em problema *flow shop* com bloqueio.

---

[10] *Local Search*

Para a construção da solução inicial os autores utilizaram o *MME* (Ronconi, 2004). Ao *DABC* foi aplicada na etapa das abelhas oportunistas uma estratégia de busca local de inserção e na etapa das abelhas exploradoras uma estratégia de destruição e reconstrução. O algoritmo foi testado e comparado com três algoritmos propostos por Wang *et al.* (2010b): *hHS* (*Hybrid Harmony Search*); *hgHS* (*Hybrid Global-Best Harmony Search*); *hmgHS* (*Hybrid Modified Global-Best Harmony Search*). Para a avaliação dos métodos foi utilizado os problemas-teste de Taillard (1993). O algoritmo obteve resultados melhores que os demais.

Guanlong *et al*. (2012) trataram do problema *flow shop* com bloqueio com o objetivo de minimizar o tempo total de fluxo. O algoritmo apresentado pelos autores foi uma modificação do *DABC* (*Discrete Artificial Bee Colony Algorithm*). O algoritmo foi testado e comparado com outros três algoritmos: *DABC* (*Discrete Artificial Bee Colony Algorithm*) proposto por Tasgetirem *et al*. (2011), *HDDE* (*Hybrid Discrete Differential Evolution*) e *IG* (*Iterated Greedy*), utilizando os problemas-teste de Taillard (1993). Os resultados mostraram que o *DABC* proposto por Guanlong *et al*. (2012) foi significativamente melhor que os demais métodos com os quais ele foi comparado.

Wang e Tang (2012) adotaram um controle de diversidade auto-adaptativo ao *DPSO* (*Discrete Particle Swarm Optimization*) para o problema *flow shop* com bloqueio com o objetivo de minimizar o *makespan*. Além disso, um método de busca local conhecido como *stochastic variable neighborhood search* foi utilizado para intensificar a busca. O algoritmo foi então testado e comparado ao algoritmo busca tabu proposto por Grabowski e Pempera (2007), o algoritmo genético de Caraffa *et al.* (2001) e o *Branch-and-Bound* de Ronconi (2005) utilizando os problemas-teste de Taillard (1993). Os resultados mostraram que o método proposto foi superior aos métodos comparados, obtendo 111 novos melhores limitantes superiores (*upper bounds*) dos 120 problemas.

Han *et al.* (2012b) desenvolveram um algoritmo *IABC* (*Improved Artificial Bee Colony*) para problemas *flow shop* com bloqueio e com o objetivo de minimizar o tempo de fluxo. O algoritmo foi comparado com o *DABC* (*Discrete Artificial Bee Colony Algorithm*) de Guanlong *et al.* (2012) e com o *HDDE* (*Hybrid Discrete Differential Evolution*) de Wang *et al.* (2010a) utilizando para avaliação os problemas-teste de Taillard (1993). A experimentação computacional demonstrou a

superioridade do método proposto na qualidade das soluções obtidas e no seu processamento computacional.

Bautista *et al.* (2012) adaptaram o uso do *BDP* (*Bounded Dynamic Programming*) para minimizar o *makespan* em problemas *flow shop* permutacional com bloqueio. O método proposto foi testado utilizando os problemas-teste de Taillard (1993) e foi capaz de melhorar 17 das melhores soluções conhecidas dentre os 120 problemas da base de dados. Dentre os problemas que tiveram os resultados melhorados estão os 10 problemas de 500 tarefas e 20 máquinas (os maiores da base de dados).

Wang *et al.* (2012) apresentaram um *TPA* (*Three-Phase Algorithm*) para o problema *flow shop* com bloqueio, com o objetivo de minimizar o *makespan*. Na primeira fase do algoritmo é aplicada uma regra de prioridade para organizar a solução inicial, em uma segunda fase o resultado é melhorado com a aplicação do *NEH* modificado, por fim é aplicado o algoritmo *SA* (*Simulated Annealing*) modificado. Os resultados obtidos com o método foram comparados com os obtidos pelo *HDDE* (*Hybrid Discrete Differential Evolution*) e pelos métodos apresentados por Ribas *et al.* (2011). O *TPA* superou significativamente os demais métodos e ainda foi capaz de definir novos e melhores limitantes superiores para 53 dos 60 problemas de grande porte dos problemas-teste de Taillard (1993).

Abyaneh e Zandieh (2012) abordaram o problema *flow shop* com máquinas múltiplas, bloqueio e *setup* dependente da sequência, com dois objetivos simultâneos (minimizar o *makespan* e a soma dos atrasos). Foram apresentados quatro métodos para solucionar o problema: *SPGA* II (*Sub-Population Genetic Algorithm II*) proposto por Chang e Chen (2009); *NSGA* II (*Non-Dominated Sorting Genetic Algorithm II*) proposto por Deb *et al.* (2002); *SPMA* (*Sub-Population Memetic Algorithm*); e *NSMA* (*Non-Dominated Sorting Memetic Algorithm*).

Maleki-Darounkolaei *et al.* (2012) desenvolveram um modelo *MILP* e um *SA* (*Simulated Annealing*) para o problema *flow shop* com máquinas múltiplas com três estações de trabalho, *setup* dependente da sequencia apenas na primeira estação de trabalho e bloqueio, com dois objetivos (minimização do *makespan* e do tempo de fluxo). No trabalho problemas com número de tarefas superior a nove não foram resolvidos utilizando o modelo *MILP*, pois o tempo computacional mostrou-se muito elevado.

Davendra e Bialic-Davendra (2013) apresentaram uma meta-heurística *DSOMA* (*Discrete Self-Organising Migrating Algorithm*) para minimizar o *makespan* em problemas *flow shop* permutacional com bloqueio. O algoritmo foi comparado com os até então três melhores métodos para o problema. Os testes foram realizados utilizando duas bases de dados diferentes, comprovando o melhor desempenho do método proposto. O método foi capaz ainda de identificar novos limitantes superiores para diversos problemas de Taillard (1993).

Ribas *et al.* (2013a) propuseram uma busca local *ILS* (*Iterated Local Search*) combinado a um mecanismo de perturbação *VNS* (*Variable Neighbourhood Search*) para solucionar problemas *flow shop* permutacionais com bloqueio e com o objetivo de minimizar o tempo total de atraso. Inicialmente os autores trabalharam a solução inicial, propondo dois híbridos do *NEH* (Nawaz-Enscore-Ham): um deles utilizando a regra *EDD* (*Earliest Due Date*) e o outro utilizando a regra *FPD* (*Fitting Processing Times And Due Dates*). Ambos resultaram em resultados similares. Em seguida foram testadas as buscas locais considerando apenas uma vizinhança (inserção ou permutação) e a busca local considerando as duas vizinhanças. A busca local considerando apenas a inserção obteve melhores resultados que a busca local considerando apenas a permutação, porém a busca local que utiliza as duas vizinhanças obteve resultados superiores aos dois. Finalmente o *ILS* foi comparado ao *GRASP* (*Greedy Randomized Adaptive Search Procedure*) de Ronconi e Henriques (2009) e o *IG* (*Iterated Greedy*) de Ribas *et al.* (2011). Os resultados comprovaram que o algoritmo apresentado obtém os melhores resultados, e ainda quanto maior e mais complexo é o problema, maior é a diferença entre os resultados obtidos pelo *ILS* combinado com o *VNS* em comparação com os demais métodos avaliados.

Lin e Ying (2013) propuseram um algoritmo *RAIS* (*Revised Artificial Immune System*) para problemas *flow shop* permutacional com bloqueio com objetivo de minimizar o *makespan*. O algoritmo foi comparado a seis outras meta-heurísticas, obtendo melhores resultados, identificando ainda novos melhores resultados para os problemas de Taillard (1993).

Moslehi e Khorasanian (2013) apresentaram dois modelos de programação inteira mista binária (*MIBP*[11]) para solucionar problemas *flow shop* permutacional com bloqueio, com o objetivo de minimizar o tempo total de fluxo. Um dos modelos baseia-se no tempo de término de cada tarefa em cada máquina e o outro se baseia no tempo de ociosidade e de bloqueio das tarefas. Foram propostos três limitantes inferiores para o problema, os quais obtiveram resultados melhores que o limitante inferior proposto em estudos anteriores. Os modelos *MIBP* apresentados tiveram dificuldades em encontrar a solução para os problemas de tamanho (*n*,*m*) iguais a (16,10), (18,7) e (18,10). O modelo apresentado foi capaz de encontrar a solução ótima de 30 problemas de Taillard (1993) com baixo tempo computacional.

Pan *et al.* (2013) apresentaram um algoritmo *MA* (*Memetic Algorithm*) de alta performance para o problema *flow shop* com bloqueio. Para solução inicial foi utilizado o *PF* (*Profile Fitting*) juntamente com o *NEH* (Nawaz-Enscore-Ham). O algoritmo foi comparado com as cinco melhores meta-heurísticas conhecidas para o problema *flow shop* com bloqueio: *HDDE* (*Hybrid Discrete Differential Evolution*); *IG* (*Iterated Greedy*); *HGA* (*Hybrid Genetic Algorithm*); *HPSO* (*Hybrid Particle Swarm Optimization*); *DPSO* (*Discrete Particle Swarm Optimization*). Foram ainda adaptados mais três outros algoritmos genéticos híbridos que eram utilizados para o problema clássico de *flow shop*. O algoritmo apresentado superou os demais e ainda melhorou, em 75 dos 120 problemas-teste de Taillard (1993), os limitantes superiores fornecidos por Ribas *et al.* (2011).

Han *et al.* (2013) investigaram o problema *flow shop* com bloqueio tendo como objetivo minimizar o tempo total de fluxo. Inicialmente foram propostas duas variações da heurística *MME* (Ronconi, 2004), chamadas de *MME-A* e *MME-B*, para compor o quadro de soluções iniciais. Em seguida foi apresentado um algoritmo híbrido *hDABC* (*Hybrid Discrete Artificial Bee Colony*) para solucionar o problema. Por fim uma busca local de inserção foi aplicada ao *hDABC* de três maneiras distintas: Na etapa das abelhas trabalhadoras (*hDABC*1); na etapa das abelhas oportunistas (*hDABC*2) e na fase das abelhas exploradoras (*hDABC*3). Os três algoritmos foram então comparados a três outros híbridos propostos por Wang *et al.* (2010a): *hHS* (*Hybrid Harmony Search*); *hgHS* (*Hybrid Global-Best Harmony*

---

[11] *Mixed Integer Binary Program*

Search); *hmgHS* (*Hybrid Modified Global-Best Harmony Search*). Os testes foram feitos utilizando os problemas-teste de Taillard (1993) e apresentaram a superioridade dos algoritmos propostos em relação aos demais, tanto em qualidade de solução como em eficiência e robustez.

Toumi *et al.* (2013a) estudaram a minimização do tempo total de fluxo e do tempo total ponderado de conclusão das tarefas em um problema *flow shop* permutacional com bloqueio com *buffer* zero. Um algoritmo *Branch-and-Bound* foi apresentado para ambos os problemas. O algoritmo foi testado em 600 instancias geradas aleatoriamente que variavam em número de tarefas e máquinas. Os resultados mostraram que o algoritmo foi eficiente até mesmo para problemas de grande porte quando o critério de avaliação é o tempo de fluxo, no entanto para os problemas de minimização do tempo total ponderado de conclusão das tarefas o algoritmo teve dificuldades em encontrar a solução ótima.

Toumi *et al.* (2013b) desenvolveram dois limitantes inferiores para o problema *flow shop* com bloqueio com *buffer* zero. Um limitante inferior para minimizar o tempo total de atraso e outro limitante inferior para minimizar o tempo total ponderado de atraso. Ambos os limitantes foram testados usando uma base de dados gerada aleatoriamente, que consiste em 540 problemas que variam em número de máquinas e de tarefas, e na faixa de valores do prazo de entrega. Os resultados mostram que os limitantes inferiores funciona bem para problemas com menos de 20 tarefas e 4 máquinas.

Ribas *et al.* (2013b) analisaram a performance de dois métodos *Variable Neighbourhood Search* (*VNS*) para o problema *flow shop* permutacional com bloqueio com *buffer* zero, com o objetivo de minimizar o *makespan*. Cada um dos métodos usa uma estratégia diferente para mudança de vizinhança na fase de melhoria: estratégia um, nomeada de versão paralela, escolhe aleatoriamente entre permutação e inserção para melhorar a solução; enquanto a estratégia dois, nomeada de versão serial, começa a busca utilizando uma das formas de mudança de vizinhança e continua a busca com a outra. Muitos dos métodos heurísticos apresentado em trabalhos anteriores foram usados para fornecer uma solução inicial para ambos os métodos. Os resultados sugerem que os métodos são bastante competitivos.

Toumi *et al.* (2013c) estudaram o problema *flow shop* permutacional com bloqueio com *buffer* zero, com o objetivo de minimizar o tempo total de atraso. Um

algoritmo *Branch-and-Bound* foi desenvolvido para o problema e testado usando uma base de dados gerada aleatoriamente com 132 problemas que variam em número de tarefas e de máquinas. Para cada problema quatro diferentes prazos de entrega foram gerados. Os resultados mostram que o algoritmo *Branch-and-Bound* proposto é capaz de resolver a maior parte dos problemas de pequeno porte, porém, problemas com mais de 20 tarefas e 4 máquinas não foram resolvidos dentro do tempo computacional máximo estipulado de 3600 segundos. Então uma comparação com o modelo *Branch-and-Bound* apresentado por Ronconi e Henriques (2009) foi realizada usando a base de dados de Taillard (1993). O modelo proposto obteve melhores resultados em 136 dos 440 problemas rodados.

Chen *et al.* (2014) estudaram o problema *flow shop* com dois estágios, máquinas de processamento em lotes, datas de liberação arbitrárias e bloqueio com *buffer* zero, com o objetivo de minimizar o *makespan*. Um modelo *MILP e um algoritmo HDDE* (*Hybrid Discrete Differential Evolution*) foram propostos para o problema. O algoritmo *HDDE* proposto foi comparado ao modelo *MILP*, e a um *Hybrid Simulated Annealing* (*HSA*) e a um *Hybrid Genetic ALgorithm* (*HGA*). O algoritmo *HDDE* superou os outros métodos tanto na qualidade das soluções como também em robustez e em tempo computacional.

Ribas *et al.* (2015) propuseram um algoritmo *Discrete Artificial Bee Colony* (chamado de *DABC_RCT*) para o problema *flow shop* permutacional com bloqueio com *buffer* zero, com o objetivo de minimizar o tempo total de fluxo. Quatro estratégias para a fase de fonte de comida e duas estratégias para cada uma das três fazes restantes foram apresentadas. Uma calibração do algoritmo foi realizada usando o *Design of experiments* (*DOE*). O algoritmo proposto foi comparado aos algoritmos *HS_WPT*, *DABC_DXG* e ao *IG_KM*, mostrando sua eficiência e superioridade.

Han *et al.* (2015) apresentaram um *novel discrete Artificial Bee Colony* (*ABC*) incorporado a um *Differential Evolution* (*DE*) para minimizar o *makespan* em um problema *flow shop* permutacional com bloqueio com *buffer* zero. O algoritmo foi testado utilizando a já conhecida base de dados de Taillard (1993), e comparado aos algoritmos *TS+M*, *HDDE*, *DABC*, *IABC*, $DPSO_{syns}$ e *EDEc*. Os resultados demonstram a superioridade do algoritmo proposto, chamado de *DE-ABC*, em relação ao desvio relativo médio do *makespan* quando comparado os outros algoritmos comparados.

Ribas e Companys (2015) estudaram a minimização do tempo total de fluxo em um problema *flow shop* permutacional com bloqueio com *buffer* zero. Duas heurística construtivas, baseadas na heurística *PF*, foram propostas, chamadas de *HPF*1 e *HPF*2. Cada uma das heurísticas propostas foram combinadas com o procedimento *NEH*, criando dois novos procedimentos chamados de *NHPF*1 e *NHPF*2. As estruturas dessas heurísticas foram usadas em um procedimento *Greedy Randomized Adaptive Search Procedures* (*GRASP*) com uma busca variada de vizinhança na fase de melhoria para gerar soluções aleatórias. Os procedimentos propostos foram comparados a um algoritmo *Harmony Search* (*HS*) e a um algoritmo *Descrete Artificial Bee Colony* (*DABC*). Os resultados mostram a superioridade dos procedimentos propostos, que melhoraram 68 das 120 melhores soluções conhecidas das instancias de Taillard (1993) para o problema.

Ding *et al.* (2015) apresentaram novas propriedades para o problema *flow shop* permutacional com bloqueio com *buffer* zero para minimizar o *makespan*. As propriedades apresentadas foram usadas como base para um procedimento *Pruning Procedure* que foi usada na fase de construção de um algoritmo *Iterated Greedy* (*IG*). As propriedade propostas foram comparadas às propriedades clássicas usando as instancias de Taillard (1993). O algoritmo *IG* proposto, nomeado de *B-IG*, for comparado aos algoritmos *HDDE*, *IG*, *TPA*, *RAIS* e *MA* também usando a base de dados fornecida por Taillard (1993). Os resultados mostram que as propriedades propostas podem eliminar mais candidatos à solução do que as propriedades clássicas. Também o algoritmo *B-IG* é bastante competitivo quando comparado a outras aproximações de alto desempenho.

Sadaqa e Moraga (2015) propuseram uma meta-heurística para minimizar o *makespan* em problemas *flow shop* permutacional com bloqueio e *buffer* zero. O método proposto, chamado de *Meta-Heurístic for Randomized Priority Search* (*Meta-RaPS*), usa uma heurística *NEH* para fornecer uma solução inicial, então, a solução construída é melhorada com uma técnica de busca de vizinhança baseada em uma comparação de números aleatórios. O método proposto foi comparado aos algoritmos *mNEH* e *DE-ABC* usando instancias da base de dados de Taillard (1993). O algoritmo *Meta-RaPS* superou o algoritmo *mNEH* somente para problemas pequenos e foi superado pelo algoritmo *DE-ABC*.

Han *et al.* (2016) apresentaram um algoritmo *Modified Fruit Fly Optimization* (*MFFO*) para minimizar o *makespan* em problemas *flow shop* permutacional com

bloqueio e *buffer* zero. O algoritmo proposto foi testado e comparado aos algoritmos *HDDE*, *DABC*, *IABC*, *DE-ABC* e *IG* usando a base de dados fornecida por Taillard (1993). Os resultados mostram que o algoritmo apresentado superou os outros algoritmos e também melhorou 94 dos 120 limitantes superiores obtidos por algoritmos anteriores.

A Tabela 3 apresenta, de forma resumida, os problemas e as contribuições de cada autor.

Tabela 3 – Revisão bibliográfica do problema *flow shop* com bloqueio

(continua)

Autor(es) (ano da publicação)	**Problema(s) considerado(s)**	**Método(s) desenvolvido(s) / Contribuição(ões)**
Reddi e Ramamoorthy (1972)	*F*2 / prmu, nwt / *C*max	*Branch-and-Bound*
Papadimitriou e Kanellakis (1980)	*F*2 / prmu, block / *C*max, *F*4 / prmu, block / *C*max	Provaram que os problemas são NP-Hard
McCormick *et al.* (1989)	*Fm* / prmu, block / *C*max	*Profile Fitting* (*PF*)
Leisten (1990)	*Fm* / prmu, block / *C*max, *Fm* / block / *C*max	Nawaz-Enscore-Ham (*NEH*)
Hall e Sriskandarajah (1996)	*F*3 / prmu, block / *Cmax*	Provaram que o problema é NP_Completo
Nowicki (1999)	*Fm* / prmu, block / *C*max	Busca Tabu (*TS*)
Norman (1999)	*Fm* / prmu, S_{ijk}, block / *C*max	Nawaz-Enscore-Ham (*NEH*), *Profile Fitting* (*PF*), Busca Tabu (*TS*)
Abadi *et al.* (2000)	*Fm* / prmu, block / Tempo de ciclo	*Sequential Slowing Down* (*SSD*) *Parallel Slowing down* (*PSD*)
Armentano e Ronconi (2000)	*Fm* / prmu, block / ΣT_j	Busca Tabu (*TS*)
Ronconi e Armentano (2001)	*Fm* / prmu, block / ΣT_j	*Branch-and-Bound*
Caraffa *et al.* (2001)	*Fm* / prmu, block / *C*max	Algoritmo Genético (*GA*)
Ronconi (2004)	*Fm* / prmu, block / *C*max	*MinMax* (*MM*), *MME*, *PFE*
Ronconi (2005)	*Fm* / prmu, block / *C*max	*Branch-and-Bound*
Martinez *et al*. (2006)	*Fm* / prmu, block / *C*max	Apresentaram um novo tipo de restrição de bloqueio de máquinas
Bagchi *et al.* (2006)	Diversos problemas de *flow shop* com objetivo de minimizar o *makespan*	Provaram que todos os problemas de *flow shop* podem ser transformados em problemas *TSP*

Tabela 3 – Revisão bibliográfica do problema *flow shop* com bloqueio

(continua)

Autor(es) (ano da publicação)	**Problema(s) considerado(s)**	**Método(s) desenvolvido(s) / Contribuição(ões)**
Grabowski e Pempera (2007)	*Fm* / prmu, block / *C*max	Busca Tabu (*TS*), Busca Tabu com multi movimentos (*TS+M*)
Ronconi e Henriques (2009)	*Fm* / prmu, block / ΣT_j	FPDNEH, *Greedy Randomized Adaptive Search Procedure* (*GRASP*)
Liu e Kozan (2009)	*Fm* / prmu, block / *C*max	Liu-Kozan Liu-Kozan-*BIH*
Wang *et al.* (2010a)	*Fm* / prmu, block / *C*max	*Hybrid Discrete Differential Evolution* (*HDDE*)
Wang *et al.* (2010b)	*Fm* / prmu, block / ΣC_j	*Hybrid Harmony Search* (*hHS*), *Hybrid Globalbest Harmony Search* (*hgHS*), *Hybrid Modified Globalbest Harmony Search* (*hmgHS*)
Gong *et al.* (2010)	*F*2 / prmu, batch(*b*) / $\lambda Cmax+(1-\lambda)w_j$, *F*2 / prmu / $\lambda Cmax+(1-\lambda)w_j$	Programação quadrática inteira mista (*QMIP*), Heurística *G*, Heurística *H*
Wang *et al.* (2011)	*Fm* / prmu, block / *C*max	*Hybrid Modified Globalbest Harmony Search* (*hmgHS*)
Ribas *et al.* (2011)	*Fm* / prmu, block / *C*max	*Iterated Greedy* (*IG*)
Pan *et al.* (2011a)	*Fm* / prmu, block / *C*max	*Hybrid Discrete Differential Evolution* (*HDDE*) + *Local Search* (*LS*)
Pan *et al.* (2011b)	*Fm* / prmu, block / *C*max	*Chaotic Harmony Search* (*CHS*)
Pan e Wang (2012)	*Fm* / prmu, block / *C*max	*wPF*, *PW*, *wPFE*, *PWE*, *PF-NEH*(*x*), *wPF-NEH*(*x*), *PW-NEH*(*x*), *Local Search* (*LS*)
Trabelsi *et al.* (2012)	*Fm* / prmu, block / *C*max	*TSS*, Nawaz-Enscore-Ham (*NEH*), Algoritmo Genético (*GA*)
Han *et al.* (2012a)	*Fm* / prmu, block / ΣC_j	*Discrete Artificial Bee Colony* (*DABC*)
Guanlong *et al.* (2012)	*Fm* / prmu, block / ΣC_j	*Discrete Artificial Bee Colony* (*DABC*) modificado
Wang e Tang (2012)	*Fm* / prmu, block / *C*max	*Discrete Particle Swarm Optimization* (*DPSO*) + *stochastic variable neighborhood search*
Han *et al.* (2012b)	*Fm* / prmu, block / ΣC_j	*Improved Artificial Bee Colony* (*IABC*)

Tabela 3 – Revisão bibliográfica do problema *flow shop* com bloqueio

(continua)

Autor(es) (ano da publicação)	**Problema(s) considerado(s)**	**Método(s) desenvolvido(s) / Contribuição(ões)**
Bautista *et al.* (2012)	*Fm* / prmu, block / *C*max	*Bounded Dynamic Programming* (*BDP*)
Wang *et al.* (2012)	*Fm* / prmu, block / *C*max	*Three-Phase Algorithm* (*TPA*)
Abyaneh e Zandieh (2012)	*FFc* / prmu, block, S_{ij} / *C*max, Σ*Tj*	*Sub-Population Genetic Algorithm* II (SPGA II), *Non-Dominated Sorting Genetic Algorithm* II (NSGA II), *Sub-Population Memetic Algorithm* (SPMA), *Non-Dominated Sorting Memetic Algorithm* (NSMA)
Maleki-Darounkolaei *et al.* (2012)	*FF*3 / prmu, block, S_{ij} / *C*max, Σ*Cj*	Programação linear inteira mista (*MILP*), *Simulated Annealing* (*SA*)
Davendra e Bialic-Davendra (2013)	*Fm* / prmu, block / *C*max	*Discrete Self-Organising Migrating Algorithm* (*DSOMA*)
Ribas *et al.* (2013)	*Fm* / prmu, block / Σ*Tj*	*Iterated Local Search* (*ILS*) + *Variable Neighbourhood Search* (*VNS*)
Lin e Ying (2013)	*Fm* / prmu, block / *C*max	*Revised Artificial Immune System* (*RAIS*)
Moslehi e Khorasanian (2013)	*Fm* / prmu, block / ΣC_j	Programação inteira mista binária (*MIBP*)
Pan *et al.* (2013)	*Fm* / prmu, block / *C*max	*Memetic Algorithm* (*MA*)
Han *et al.* (2013)	*Fm* / prmu, block / ΣC_j	*Hybrid Discrete Artificial Bee Colony* (*hDABC*) + *Local Search* (*LS*)
Toumi *et al.* (2013a)	*Fm / prmu, block / ΣCj*, *Fm / prmu, block / ΣwjCj*	*Branch-and-Bound*
Toumi *et al.* (2013b)	*Fm / prmu, block / ΣTj*, *Fm / prmu, block / ΣwjTj*	*Branch-and-Bound*
Ribas *et al.* (2013b)	*Fm / prmu, block / Cmax*	*Variable Neighbourhood Search (VNS)*
Toumi *et al*. (2013c)	*Fm / prmu, block / ΣTj*	*Branch-and-Bound*
Chen *et al.* (2014)	*F2 / prmu,batch(b), rj, block / Cmax*	*MILP* *Hybrid Discrete Differential Evolution (HDDE)*
Ribas *et al.* (2015)	*Fm / prmu, block / ΣCj*	*Discrete Artificial Bee Colony (DABC_RCT)*
Han *et al.* (2015)	*Fm / prmu, block / Cmax*	*Artificial Bee Colony (ABC) + Differential Evolution (DE)*
Ribas e Companys (2015)	*Fm / prmu, block / ΣCj*	*HPF1* *HPF2* *NHPF1* *NHPF2* *Greedy Randomized Adaptive Search Procedure (GRASP)*
Ding *et al.* (2015)	*Fm / prmu, block / Cmax*	*Iterated Greedy (IG)*

Tabela 3 – Revisão bibliográfica do problema *flow shop* com bloqueio

(conclusão)

Sadaqa e Moraga (2015)	*Fm / prmu, block / Cmax*	*Meta-heuristic for Randomized Priority Search (Meta- RaPS)*
Han *et al.* (2016)	*Fm / prmu, block / Cmax*	*Modified Fruit Fly Optmization (MFFO)*

Conforme a revisão bibliográfica apresentada no item, a literatura que trata do ambiente *flow shop* com bloqueio de máquinas é relativamente extensa. Entretanto, no exame efetuado na literatura para o problema *flow shop* com bloqueio e com *setup* separado e dependente da sequencia foram encontrados somente três trabalhos publicados por Norman (1999), Abyaneh e Zandieh (2012) e Maleki-Darounkolaei *et al.* (2012).

Porém, no trabalho de Norman (1999) o autor não considera o problema de *buffer* zero, variando o tamanho do estoque intermediário. Neste trabalho são utilizados apenas dois métodos heurísticos construtivos adaptados de problemas sem *setup* e a busca Tabu. No trabalho de Abyaneh e Zandieh (2012), o problema proposto consiste em um ambiente *flow shop* com máquinas múltiplas e tempo de *setup* independente da máquina. No trabalho de Maleki-Darounkolaei *et al.* (2012), o problema proposto consiste em um ambiente com três estações de trabalho, e com *setup* dependente da sequência apenas na primeira estação de trabalho.

O presente trabalho, por outro lado, considera o estoque intermediário igual a zero (*buffer* zero), *setup* dependente não apenas da sequência, mas também da máquina, e ainda propõe a adaptação de 28 heurísticas construtivas, quatro limitantes inferiores para um algoritmo *Branch-and-Bound* e dois programas *MILP* e a adaptação de mais dois modelos *MILP* para a solução do problema.

No próximo capítulo são apresentadas duas propriedades estruturais do problema: um limitante superior para o tempo de ociosidade; e um limitante inferior para o tempo de bloqueio.

4. PROPRIEDADES ESTRUTURAIS DO PROBLEMA *Fm / prmu, Sijk, block / Cmax*

Neste capítulo é apresentada a base para a obtenção do limitante superior (*upper bound*) para o tempo de ociosidade das máquinas (denominado *UBO*) e do limitante inferior (*lower bound*) para o tempo de bloqueio das máquinas (denominado *LBB*) para o problema *Fm* / prmu, *Sijk*, block / *C*max. O limitante inferior será utilizado para a obtenção dos limitantes inferiores para o *Branch-and-Bound*. As notações utilizadas são:

- n: Número de tarefas ;
- m: Número de máquinas;
- $j_1, j_2, ..., j_n$: Conjunto de tarefas que devem ser alocadas nas máquinas;
- $k_1, k_2, ..., k_m$: Conjunto de máquinas que realizarão as tarefas;
- P_{jk}: Tempo de processamento da tarefa j na máquina k;
- S_{ijk}: Tempo de *setup* da tarefa j, precedida diretamente pela tarefa i, na máquina k, sendo S_{0jk} o tempo de *setup* da primeira tarefa da sequência;
- C_{jk}: Data de término da tarefa j na máquina k;
- R_{jk}: Data de término do *setup* da máquina k para execução da tarefa j;
- B_{jk}: Tempo de bloqueio da tarefa j na máquina k, *i.e.* o intervalo de tempo entre o término da tarefa j na máquina k e o início da operação da mesma tarefa na máquina (k+1). Ou seja:

$$B_{jk} = \left(C_{j,k+1} - P_{j,k+1} - B_{j,k+1}\right) - C_{jk} \qquad \forall k = 1,..., m-1 \quad (7)$$

e,

$$B_{jm} = 0 \quad (8)$$

- O_{jk}: Tempo de ociosidade da máquina k, *i.e*. Intervalo de tempo entre o término do *setup* para execução da tarefa j e o início da operação da tarefa j na máquina k. Ou seja:

$$O_{j1} = 0 \quad (9)$$

e,

$$O_{jk} = \left(C_{jk} - P_{jk}\right) - R_{jk} - B_{jk} \tag{10}$$

As propriedades B_{jk} e O_{jk} são representadas na Figura 4.

4.1. Limitante Superior

O limitante superior da ociosidade foi obtido por meio do estudo de algumas propriedades estruturais do problema. Como não havia estudos a respeito deste problema em específico, as propriedades são demonstradas a seguir.

As propriedades estruturas apresentadas a seguir foram desenvolvidas adaptando as já conhecidas e demonstradas propriedades estruturais para problemas de programação de operações *flow shop* permutacional, propriedades A, B e C (Bellman *et al.*, 1982; French, 1982) e propriedades I e II (Moccellin, 1992).

PROPRIEDADE A

Considerando uma sequencia qualquer de tarefas e a tarefa *j*, tem-se que:

$$C_{j+1,k+1} = \max\left(C_{j+1,k} + P_{j+1,k+1} + B_{j+1,k+1}, R_{j+1,k+1} + P_{j+1,k+1} + B_{j+1,k+1}\right) \tag{11}$$

para $k = 1, 2, ..., m-1$

A propriedade A é ilustrada na Figura 4.

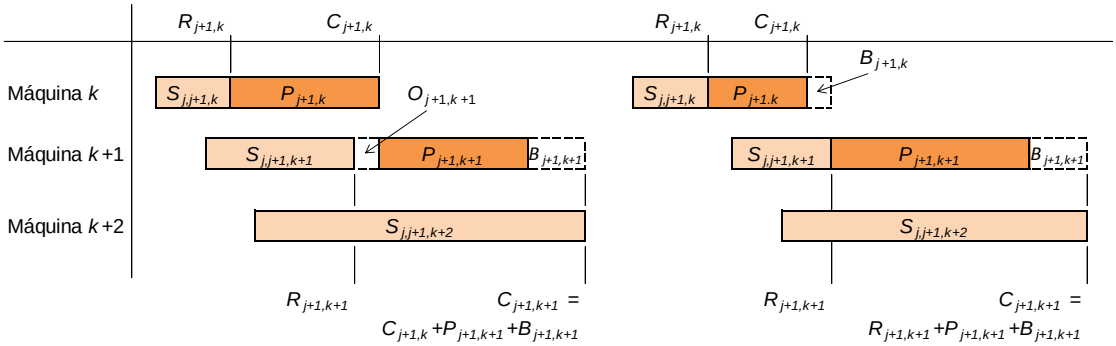

Figura 4 – Ilustração da propriedade A e das propriedades B_{jk} e O_{jk}

PROPRIEDADE B

Considerando uma sequencia qualquer de tarefas e as tarefas *j* e (*j*+1), tem-se:

B.1 Se $O_{j+1,k+1} > 0$ então $B_{j+1,k} = 0$;

B.2 Se $B_{j+1,k} > 0$ então $O_{j+1,k+1} = 0$.

Ou seja,

$O_{j+1,k+1}$ e $B_{j+1,k}$ não podem ser simultaneamente positivos ($k = 1, 2, ..., m-1$).

A verificação da propriedade B decorre diretamente da propriedade A e das definições de $O_{j+1,k+1}$ e $B_{j+1,k}$, conforme segue:

Sendo

$$\begin{cases} C_{j+1,k+1} = \max\left(C_{j+1,k}; R_{j+1,k+1}\right) + P_{j+1,k+1} + B_{j+1,k+1} \\ O_{j+1,k+1} = \left(C_{j+1,k+1} - P_{j+1,k+1}\right) - R_{j+1,k+1} - B_{j+1,k+1} \\ B_{j+1,k} = \left(C_{j+1,k+1} - P_{j+1,k+1} - B_{j+1,k+1}\right) - C_{j+1,k} \end{cases}$$

Tem-se:

a) Se $B_{j+1,k} > 0$ então $C_{j+1,k+1} > C_{j+1,k} + P_{j+1,k+1} + B_{j+1,k+1}$

e, portanto $C_{j+1,k+1} = R_{j+1,k+1} + P_{j+1,k+1} + B_{j+1,k+1}$, ou seja:

$$O_{j+1,k+1} = C_{j+1,k+1} - P_{j+1,k+1} - R_{j+1,k+1} - B_{j+1,k+1} = 0 \qquad (12)$$

b) Se $O_{j+1,k+1} > 0$ então $C_{j+1,k+1} > R_{j+1,k+1} + P_{j+1,k+1} + B_{j+1,k+1}$

e, portanto $C_{j+1,k+1} = C_{j+1,k} + P_{j+1,k+1} + B_{j+1,k+1}$, ou seja:

$$B_{j+1,k} = C_{j+1,k+1} - C_{j+1,k} - P_{j+1,k+1} - B_{j+1,k+1} = 0 \quad (13)$$

PROPRIEDADE C

Considerando uma sequencia qualquer de tarefas e as tarefas *j* e (*j*+1), tem-se:

$$S_{j,j+1,k} + O_{j+1,k} + P_{j+1,k} + B_{j+1,k} = P_{j,k+1} + B_{j,k+1} + S_{j,j+1,k+1} + O_{j+1,k+1} \quad (14)$$

para $k = 1, 2, ..., m-1$.

A propriedade C é ilustrada na Figura 5.

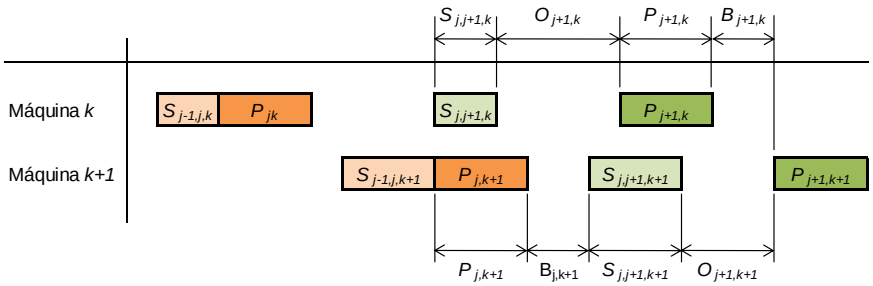

Figura 5 – Ilustração da propriedade C

TEOREMA 1

Para facilitar a visualização do problema, considera-se o caso especial com somente duas tarefas (*n* = 2), quantidade de máquinas qualquer (*m* ≥ 2) e tempo de *setup* da primeira tarefa igual a zero ($S_{0,1,k} = 0$), com $k = 1, 2, ..., m$:

a) Se $\left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) > 0$ então

$$O_{2,k+1} = \left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) \quad (15)$$

b) Se $\left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) \leq 0$ então

$$O_{2,k+1} = 0 \quad (16)$$

O teorema 1 é representado na Figura 6.

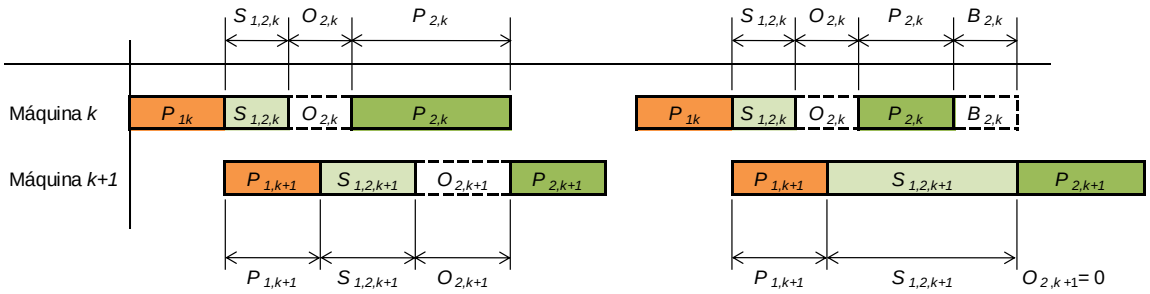

Figura 6 – Representação do teorema 1

PROVA

A partir da equação 14 da propriedade C, pode-se escrever, para *j*=1:

$$S_{1,2,k} + O_{2k} + P_{2k} + B_{2k} = P_{1,k+1} + B_{1,k+1} + + S_{1,2,k+1} + O_{2,k+1} \tag{17}$$

Como $S_{0,1,k}$=0 para $k = 1, 2, ..., m$, então não há tempo de bloqueio das máquinas para a primeira tarefa, ou seja,

$$B_{1k} = 0 \tag{18}$$

para $k = 1, 2, ..., m$-1,

logo, tem-se:

$$S_{1,2,k} + O_{2k} + P_{2k} + B_{2k} = P_{1,k+1} + S_{1,2,k+1} + O_{2,k+1} \tag{19}$$

ou

$$O_{2,k+1} = \left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) + B_{2k} \tag{20}$$

que leva a

$$O_{2,k+1} - B_{2k} = \left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) \quad (21)$$

Por meio da propriedade B pode-se estabelecer:

- Se $O_{2,k+1} > 0 \quad \Rightarrow \quad B_{2,k} = 0$;
- Se $B_{2,k} > 0 \quad \Rightarrow \quad O_{2,k+1} = 0$.

a) Se $\left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) > 0$, então $O_{2,k+1} - B_{2k} > 0$, ou seja,

$$O_{2,k+1} > B_{2k} \quad (22)$$

Como, por definição, $B_{2k} \geq 0$, tem-se que $O_{2,k+1} > 0$, logo $B_{2k} = 0$, substituindo na equação 21 tem-se que:

$$O_{2,k+1} = \left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) \quad (23)$$

b) Se $\left(O_{2k} + S_{1,2,k} + P_{2k}\right) - \left(P_{1,k+1} + S_{1,2,k+1}\right) \leq 0$, então

$$O_{2,k+1} - B_{2k} \leq 0 \quad (24)$$

Uma vez que, por definição, $O_{2,k+1} \geq 0$, e supondo que $O_{2,k+1} > 0$, onde então $B_{2,k} = 0$, ter-se-ia $O_{2,k+1} \leq 0$, ou seja, um absurdo, logo,

$$O_{2,k+1} = 0 \quad (25)$$

PROPRIEDADE I

No caso particular de somente duas tarefas (n = 2), quantidade de máquinas qualquer ($m \geq 2$) e tempo de *setup* da primeira tarefa igual a zero ($S_{0,1,k} = 0$), o tempo de ociosidade da máquina (k+1), com $k = 1, 2, ..., m-1$, é dado por:

$$O_{2,k+1} = \max\left(0,\left(S_{1,2,k} + O_{2k} + P_{2k}\right)-\left(P_{1,k+1} + S_{1,2,k+1}\right)\right) \quad (26)$$

com $O_{2,1} = 0$

A verificação da Propriedade I decorre diretamente do Teorema 1 e da característica do problema de que na primeira máquina (k = 1) não há tempo de ociosidade, ou seja, $O_{2,1} = 0$.

A equação 26, da Propriedade I, é uma expressão que pode ser utilizada para calcular o tempo de ociosidade de todas as máquinas, inclusive da última (onde $k = m$-1), entre o *setup* da segunda tarefa e a operação também da segunda tarefa.

Definição: Seja $UBO_{j+1}^{k} \geq 0$ um LIMITANTE SUPERIOR de $O_{j+1,k}$, ou seja, tal que $O_{j+1,k} \leq UBO_{j+1}^{k}$ para todo $k = 1, 2, ..., m$ e $j = 1, 2, ..., n$-1.

Uma vez que $O_{j+1,k} \leq UBO_{j+1}^{k}$, adicionando-se $\left(S_{j,j+1,k} + P_{j+1,k} - P_{j,k+1} - S_{j,j+1,k+1}\right)$ em ambos os termos da desigualdade, tem-se:

$$O_{j+1,k} + \left(S_{j,j+1,k} + P_{j+1,k} - P_{j,k+1} - S_{j,j+1,k+1}\right) \leq UBO_{j+1}^{k} + \left(S_{j,j+1,k} + P_{j+1,k} - P_{j,k+1} - S_{j,j+1,k+1}\right) \quad (27)$$

ou seja,

$$\left(O_{j+1,k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \leq \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \quad (28)$$

Da equação 14 da propriedade C:

$$S_{j,j+1,k} + O_{j+1,k} + P_{j+1,k} + B_{j+1,k} = P_{j,k+1} + B_{j,k+1} + S_{j,j+1,k+1} + O_{j+1,k+1}$$

tem-se:

$$O_{j+1,k+1} - B_{j+1,k} = \left(O_{j+1,k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) - B_{j,k+1} \quad (29)$$

Como $B_{j,k+1} \geq 0$, tem-se, então:

$$O_{j+1,k+1} - B_{j+1,k} \leq \left(O_{j+1,k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \quad (30)$$

Utilizando-se as equações 28 e 30 tem-se que:

$$O_{j+1,k+1} - B_{j+1,k} \leq \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \quad (31)$$

TEOREMA 2

Considerando o caso geral com *n*, *m* e $S_{j,j+1,k}$ quaisquer, ou seja, $n \geq 2$, $m \geq 2$ e $S_{j,j+1,k} \geq 0$, com $k = 1, 2, ..., m$;

a) Se $\left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1.k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) > 0$, então

$$O_{j+1,k+1} \leq \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \quad (32)$$

b) Se $\left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1.k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \leq 0$, então

$$O_{j+1,k+1} = 0 \quad (33)$$

PROVA

a) Uma vez que, por definição $O_{j+1,k+1} \geq 0$,

- Se $O_{j+1,k+1} = 0$, obviamente:

$$O_{j+1,k+1} \leq \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right)$$

- Se, por outro lado, $O_{j+1,k+1} > 0$, então pela propriedade B, $B_{j+1,k} = 0$, logo por meio da equação 31 ter-se-á que:

$$O_{j+1,k+1} \leq \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right)$$

b) Se $\left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1.k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \leq 0$, então, pela equação 31, ter-se-á:

$$O_{j+1,k+1} - B_{j+1,k} \leq 0$$

Supondo que $O_{j+1,k+1} > 0$, então ter-se-ia, pela propriedade B, $B_{j+1,k} = 0$, o que levaria a $O_{j+1,k+1} \leq 0$, ou seja, um absurdo. Logo:

$$O_{j+1,k+1} = 0$$

PROPRIEDADE II

Para o caso geral com *n*, *m* e $S_{j,j+1,k}$ quaisquer, ou seja, $n \geq 2$, $m \geq 2$ e $S_{j,j+1,k} \geq 0$, com $k = 1, 2, ..., m$, e sendo $O_{j+1,k+1}$ o tempo de ociosidade da máquina (*k*+1), entre o *setup* da tarefa (*j*+1) e a operação da tarefa (*j*+1), então UBO_{j+1}^{k+1} é um LIMITANTE SUPERIOR de $O_{j+1,k+1}$, dado por:

$$UBO_{j+1}^{k+1} = \max\left(0, \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right)\right) \qquad (34)$$

$$\text{com } UBO_{j+1}^{1} = 0.$$

VERIFICAÇÃO DA PROPRIEDADE II

Por meio do Teorema 2 e considerando $UBO_{j+1}^{k+1} \geq O_{j+1,k+1}$, limitante superior de $O_{j+1,k+1}$, tem-se:

a) Se $\left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1.k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) > 0$ então

$$O_{j+1,k+1} \leq \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right)$$

o que permite escrever

$$UBO_{j+1}^{k+1} = \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \quad (35)$$

b) Se $\left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1.k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right) \leq 0$ então

$$O_{j+1,k+1} = 0$$

o que permite estabelecer que

$$UBO_{j+1}^{k+1} = 0 \quad (36)$$

c) $UBO_{j+1}^{1} = 0$, para $j = 1, 2, ..., n-1$, uma vez que na primeira máquina não há tempo de espera entre as operações sucessivas das tarefas.

A propriedade II mostra então que, de maneira recorrente, a partir da equação 34 e fazendo-se *k* = *m*-1, pode-se calcular um LIMITANTE SUPERIOR para o tempo de ociosidade da última máquina, entre o *setup* da tarefa (*j*+1) e a operação da tarefa (*j*+1).

4.2. Limitante Inferior

Tendo como fundamento o estudo efetuado no item 4.1 deste trabalho, pode-se rescrever a equação 31:

$$O_{j+1,k+1} - B_{j+1,k} \leq \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) - \left(P_{j,k+1} + S_{j,j+1,k+1}\right)$$

como:

$$B_{j+1,k} \geq \left(P_{j,k+1} + S_{j,j+1,k+1}\right) - \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) + O_{j+1,k+1} \quad (37)$$

Uma vez que, por definição, $O_{j+1,k+1} \geq 0$, pode-se concluir que:

$$B_{j+1,k} \geq \left(P_{j,k+1} + S_{j,j+1,k+1}\right) - \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right) \quad (38)$$

A partir da expressão acima, pode-se estabelecer a propriedade LBB. No problema de programação de operações *flow shop* com bloqueio e com tempo de *setup* dependente da sequencia com $n \geq 2$ e $m \geq 2$, seja σ uma sequencia qualquer das *n* tarefas. Sendo $B_{j+1,k}$ o tempo de bloqueio entre as operações sucessivas da tarefa (*j*+1) nas máquinas *k* e (*k*+1), então, para qualquer *j*, LBB_{j+1}^{k} é um **LIMITANTE INFERIOR** de $B_{j+1,k}$ dado por:

$$LBB_{j+1}^{k} = \max\left(0, \left(P_{j,k+1} + S_{j,j+1,k+1}\right) - \left(UBO_{j+1}^{k} + S_{j,j+1,k} + P_{j+1,k}\right)\right) \quad (39)$$

$$\text{com } LBB_{j+1}^{m} = 0. \quad (40)$$

Onde UBO_{j+1}^{k} é um limitante superior do Intervalo de tempo entre o término do *setup* para execução da tarefa *j* e o início da operação da tarefa *j* na máquina *k*, dado por:

$$UBO_{j+1}^{k} = \max\left(0, \left(UBO_{j+1}^{k-1} + S_{j,j+1,k-1} + P_{j+1,k-1}\right) - \left(P_{j,k} + S_{j,j+1,k}\right)\right)$$

$$\text{com } UBO_{j+1}^{1} = 0.$$

As propriedades UBO e LBB são ilustradas na Figura 7.

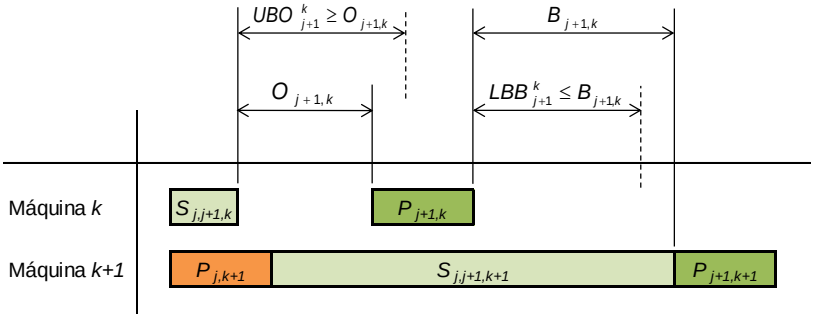

Figura 7 – Ilustração das Propriedades UBO_{j+1}^{k} e LBB_{j+1}^{k}

Para demonstrar os procedimentos para os cálculos do limitante superior da ociosidade (UBO) e do limitante inferior do tempo de bloqueio (LBB), consideremos o mesmo exemplo apresentado no capítulo 2.1, com $n=4$ e $m=3$. Pela propriedade II, tem-se que, para qualquer tarefa $j=[1,2,3,4]$, precedida diretamente pela tarefa $i=[1,2,3,4]$ na sequência na máquina $k1$

$$UBO_{ij}^{1}=0.$$

Utilizando a equação 34 pode-se calcular o limitante superior para a ociosidade das máquinas $k2$ e $k3$ para cada tarefa j diretamente precedida da tarefa i. Para exemplificar o cálculo consideremos a sequência $\sigma=[3,1,4,2]$ e os dados de tempos de processamento e de *setup* das tabelas 1 e 2. Nesse caso, UBO para a transição da tarefa que ocupa a posição $\sigma 1$ para a tarefa que ocupa a posição $\sigma 2$ na máquina $k2$ é

$$UBO_{12}^{2}=\max\left(0,\left(UBO_{2}^{1}+S_{3,1,1}+P_{1,1}\right)-\left(P_{3,2}+S_{3,1,2}\right)\right)=\max\left(0,\left(0+3+5\right)-\left(4+9\right)\right)=0,$$

e na máquina $k3$ é

$$UBO_{12}^{3}=\max\left(0,\left(UBO_{2}^{2}+S_{3,1,2}+P_{1,2}\right)-\left(P_{3,3}+S_{3,1,3}\right)\right)=\max\left(0,\left(0+9+5\right)-\left(5+7\right)\right)=2.$$

Para qualquer tarefa $j=[1,2,3,4]$, precedida diretamente pela tarefa $i=[1,2,3,4]$ na sequência na máquina $k3$, LBB é

$$LBB_{ij}^{3} = 0$$

Para os cálculos dos limitantes inferiores do bloqueio (LBB), pode utilizar a equação 39. Portanto para a transição da tarefa que ocupa a posição $\sigma 1$ para a tarefa que ocupa a posição $\sigma 2$ na máquina $k1$, LBB é

$$LBB_{12}^{1} = \max\left(0,\left(4+9\right)-\left(0+3+5\right)\right) = 5,$$

e na máquina $k2$ é

$$LBB_{12}^{2} = \max\left(0,\left(5+7\right)-\left(0+9+5\right)\right) = 0.$$

De forma análoga são calculados os limitantes superiores e inferiores para a transição para $\sigma 3$ e para $\sigma 4$. Inicialmente os cálculos para UBO para a transição da tarefa que ocupa a posição $\sigma 2$ para a tarefa que ocupa a posição $\sigma 3$ são, na máquina $k2$

$$UBO_{23}^{2} = \max\left(0,\left(0+13+4\right)\text{-}\left(5+7\right)\right) = 5,$$

e na máquina $k3$

$$UBO_{23}^{3} = \max\left(0,\left(5+7+3\right)\text{-}\left(3+15\right)\right) = 0.$$

Para a transição da tarefa que ocupa a posição $\sigma 2$ para a tarefa que ocupa a posição $\sigma 3$ na máquina $k1$, LBB é

$$LBB_{23}^{1} = \max\left(0,\left(5+7\right)-\left(0+13+4\right)\right) = 0,$$

e na máquina $k2$

$$LBB_{23}^{2} = \max\left(0,\left(3+15\right)-\left(5+7+3\right)\right)=3\,.$$

Para a transição da tarefa que ocupa a posição $\sigma 3$ para a tarefa que ocupa a posição $\sigma 4$ na máquina $k2$, UBO é

$$UBO_{34}^{2} = \max\left(0,\left(0+10+3\right)\text{-}\left(3+4\right)\right)=6\,,$$

e na máquina $k3$

$$UBO_{34}^{3} = \max\left(0,\left(6+4+4\right)\text{-}\left(3+11\right)\right)=0\,.$$

Para a transição da tarefa que ocupa a posição $\sigma 3$ para a tarefa que ocupa a posição $\sigma 4$ na máquina $k1$, LBB é

$$LBB_{34}^{1} = \max\left(0,\left(3+4\right)-\left(0+10+3\right)\right)=0\,,$$

e na máquina $k2$

$$LBB_{34}^{2} = \max\left(0,\left(3+11\right)-\left(6+4+4\right)\right)=0\,.$$

Na Figura 8 são apresentados, no gráfico de GANTT, os limitantes inferiores e superiores calculados.

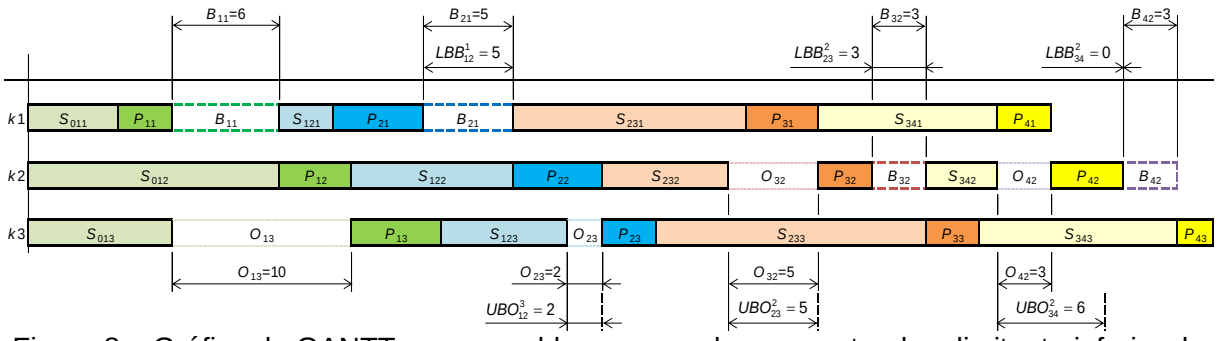

Figura 8 – Gráfico de GANTT para o problema exemplo apresentando o limitante inferior do bloqueio e o limitante superior da ociosidade

No próximo capítulo é apresentado o método de pesquisa utilizado neste trabalho, como foram realizados os testes computacionais e as avaliações dos métodos de solução.

5. MÉTODO DE PESQUISA

Tendo em vista sua natureza Bibliográfica e de Laboratório, a pesquisa proposta é desenvolvida por meio das seguintes etapas:

I. Revisão da bibliografia com consultas periódicas às fontes bibliográficas pertinentes;

O exame da literatura apresentado neste projeto indica que o problema que está sendo investigado ainda não foi intensivamente reportado na literatura. Desta forma, o acompanhamento contínuo do Estado da Arte torna-se uma etapa relevante e que pode eventualmente influenciar as etapas seguintes.

II. Concepção de métodos para solução do problema;

As propriedades estruturais do problema, incluindo os limitantes inferior e superior do problema, são utilizadas para o desenvolvimento de quatro limitantes inferiores para o makespan, que são utilizados em um algoritmo *Branch-and-Bound*, e dois modelos *MILP*. Também serão adaptados para o problema mais dois modelos *MILP* e 28 métodos heurísticos construtivos.

III. Avaliação dos métodos propostos;

Esta etapa é composta de quatro fases:

a) Implementação computacional;

Os métodos propostos são avaliados quanto ao seu desempenho na obtenção de soluções para o problema abordado e neste tipo de pesquisa a avaliação é feita por meio de uma simulação computacional. Desta forma, os métodos são implementados computacionalmente utilizando uma linguagem adequada de programação.

Todos os limitantes inferiores e o algoritmo *Branch-and-Bound*, bem como todos os métodos heurísticos adaptados foram programados em Matlab. Já os modelos *MILP* foram programados no *software* GAMS e solucionados utilizando o CPLEX 12.

b) Delineamento da experimentação;

O delineamento consiste na parametrização do conjunto de dados para os problemas a serem solucionados na simulação computacional, tais como: número de tarefas (n), número de máquinas (m), tempos de processamento das tarefas em cada máquina (P_{jk}) e tempos de *setup* das máquinas para cada par de tarefas (S_{ijk}).

Cada método de solução é mais indicado para diferentes classes de problemas, devido às suas características de execução. Portanto, para cada método de solução apresentado (modelos *MILP*, método *Branch-and-Bound*, e métodos heurísticos construtivos) foram utilizados problemas diferentes, alterando quantidade de máquinas e de tarefas. Na Tabela 4 são apresentados os dados utilizados para cada método de solução.

Tabela 4 – Conjunto de dados utilizados nas experimentações de cada método de solução

Método de solução	n	m	P_{jk}	S_{ijk}
MILP	5, 10, 15, 20	3, 7, 10	U[1,99]	U[1,99]
Branch-and-Bound	10, 12, 14	2, 3, 4, 5, 7, 10	U[1,99]	U[1,99]
Comparação *MILP* e *Branch-and-Bound*	5, 10, 15, 20	3, 7, 10	U[1,99]	U[1,99]
Heurísticas	20, 50, 100, 200, 500	5, 10, 20	U[1,99]	U[1,25], U[1,50], U[1,99], U[1,125]

c) Execução da experimentação computacional;

Tendo os programas computacionais dos métodos de solução a serem avaliados e os dados dos problemas, efetua-se a simulação computacional sobre o conjunto de problemas-teste, o qual foi definido no delineamento da experimentação.

Todos os experimentos foram realizados em um Intel® core i7 3610QM with 2.3 GHz, 8 Gb DDR3 RAM e sistema operacional Windows 7

d) Análise dos resultados;

Os resultados experimentais obtidos são analisados e os desempenhos dos métodos propostos são avaliados tendo em vista um adequado equilíbrio entre qualidade da solução e eficiência computacional.

Quanto à qualidade da solução, os resultados obtidos na simulação computacional são avaliados por meio da: tempo computacional, porcentagem de sucesso e desvio relativo médio, para todos os métodos de solução comparados.

A porcentagem de sucesso é calculada pelo número de vezes que o método forneceu a melhor solução (empatando ou não), dividido pelo número de problemas da classe.

O desvio relativo é o percentual da variação correspondente à melhor solução obtida pelos métodos. Quando o método fornecer a melhor solução para um determinado problema, o valor de seu desvio relativo será zero. Desta forma, o melhor método é aquele que apresenta o menor valor de desvio relativo médio (a média aritmética dos desvios relativos) para uma determinada classe de problemas.

O desvio relativo (DR_h) de um método *h* para um determinado problema é assim calculado:

$$DR_h = \frac{D_h - D^*}{D^*} \tag{41}$$

onde D_h é o *makespan* fornecido pelo método *h* e D^* é o melhor *makespan* fornecido pelos métodos.

Para a análise da eficiência computacional é utilizada a média de tempo de solução de cada método para cada classe de problemas.

IV. Redação de artigos para divulgação dos resultados obtidos.

Os resultados obtidos foram objeto de artigos que foram apresentados em eventos internacionais, bem como submetidos para publicação em periódicos qualificados da área de conhecimento.

5.1. Recursos de Infraestrutura Necessários

Os materiais e serviços necessários para a realização da pesquisa, ou seja, acervo bibliográfico, *software* e equipamentos computacionais, são disponíveis no sistema integrado de bibliotecas da Universidade de São Paulo e nos Laboratórios de Apoio Computacional e Processamento Científico da Escola de Engenharia de São Carlos. Os *softwares* utilizados foram o Matlab e o GAMS e o computador utilizado é de uso pessoal do aluno.

Nos capítulos seis, sete e oito são apresentados os modelos de solução desenvolvidos e adaptados para o problema estudado.

6. MODELOS MILP

Neste trabalho dois modelos *MILP* são propostos, bem como a adaptação de dois outros modelos propostos por (Ronconi; Birgin, 2012). O objetivo é comparar, em relação a custos computacionais, os quatro modelos para solucionar problemas de sequenciamento da produção em ambientes *flow shop* permutacional com *buffer* zero e *setup* separado e dependente da sequência e da máquina, com a finalidade de minimizar o *makespan*.

6.1. Modelos Propostos

A restrição de *bufffer* zero determina que não haja estoque intermediário entre duas máquinas. Isso significa que uma máquina fica bloqueada caso ela termine o processamento de uma tarefa e a máquina seguinte ainda não esteja preparada para receber essa tarefa. Inicialmente são apresentados dois modelos *MILP* propostos para o problema (TNZBS1 e TNZBS2). Em seguida serão apresentados dois outros modelos *MILP* adaptados de Ronconi e Birgin (2012). As notações utilizadas para todos os modelos são as seguintes:

n número de tarefas;

m número de máquinas;

P_{jk} Tempo de processamento da tarefa *j* na máquina *k*;

S_{ijk} Tempo de *setup* da máquina *k* entre o término da tarefa *i* e o início da tarefa *j*;

$R_{\sigma k}$ Data de término do *setup* da máquina *k* para a tarefa na posição σ da sequência;

$C_{\sigma k}$ Data de término do processamento da tarefa na posição σ da sequência na máquina *k*;

$D_{\sigma k}$ Data em que uma tarefa libera a máquina após o término do processamento. $D_{\sigma k} \geq C_{\sigma k}$, se não houver bloqueio $D_{\sigma k} = C_{\sigma k}$, caso contrário $D_{\sigma k} > C_{\sigma k}$;

$x_{j\sigma}$ $\begin{cases}1\\0\end{cases}$

Se a tarefa j pertence à posição σ da sequência;
Caso contrário;

$y_{ij\sigma}$ $\begin{cases}1\\0\end{cases}$

Se a tarefa *i* precede diretamente a tarefa *j*, que pertence à posição σ da sequência;
Caso contrário;

$e_{\sigma k}$ Data de início da tarefa na posição σ da sequência na máquina *k*;

$empty_{\sigma k}$ Intervalo de tempo entre o término do *setup* na máquina *k* para a execução da tarefa na posição σ da sequência e o início do seu processamento na máquina, ou seja, o tempo de ociosidade da máquina *k*;

$block_{\sigma k}$ O intervalo de tempo entre o término da tarefa na posição σ da sequência na máquina *k* e o início da operação da mesma tarefa na máquina (*k*+1).

6.1.1. Modelo TNZBS1

$$\text{Minimizar: } D_{max} = D_{nm} \quad (42)$$

$$\text{Restrito a: } \sum_{j=1}^{n} x_{j\sigma} = 1 \qquad \forall \sigma = 1, \dots, n \quad (43)$$

$$\sum_{\sigma=1}^{n} x_{j\sigma} = 1 \qquad \forall j = 1, \dots, n \quad (44)$$

$$y_{ij\sigma} \geq x_{j\sigma} + x_{i,\sigma-1} - 1 \qquad \forall i = 1, \dots, n; j = 1, \dots, n; \sigma = 2, \dots, n; i \neq j \quad (45)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij\sigma} = 1 \qquad \forall \sigma = 2, \dots, n \quad (46)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij1} = 0 \quad (47)$$

$$R_{1k} = \sum_{i=1}^{n} \sum_{j=1} S_{ijk} * x_{j1} \qquad \forall k = 1, \dots, m \quad (48)$$

$$R_{\sigma k} = D_{\sigma-1,k} + \sum_{i=1}^{n} \sum_{j \neq i} S_{ijk} * y_{ij\sigma} \qquad \forall \sigma = 2, \dots, n; k = 1, \dots, m \quad (49)$$

$$D_{\sigma k} \geq R_{\sigma,k+1} \qquad \forall \sigma = 1, \dots, n; k = 1, \dots, m-1 \quad (50)$$

$$D_{\sigma 1} \geq R_{\sigma 1} + \sum_{j=1}^{n} P_{j1} * x_{j\sigma} \qquad \forall \sigma = 1, \dots, n \quad (51)$$

$$D_{\sigma k} \geq D_{\sigma,k-1} + \sum_{j=1}^{n} P_{jk} * x_{j\sigma} \qquad \forall \sigma = 1, \dots, n; k = 2, \dots, m \quad (52)$$

$$D_{\sigma k}, R_{\sigma k} \geq 0 \text{ e inteiro} \qquad \forall \sigma = 1, \dots, n; k = 1, \dots, m \quad (53)$$

$$x_{j\sigma}, y_{ij\sigma} \in \{0,1\} \qquad \forall \mathrm{i} = 1, \dots, \mathrm{n}; \mathrm{j} = 1, \dots, \mathrm{n}; \sigma = 1, \dots, n \quad (54)$$

As restrições (43) e (44) garantem que cada tarefa será alocada a apenas uma posição da sequência e que cada posição da sequência só pode estar relacionada a uma tarefa. A restrição (46) em conjunto com a restrição (45) garante que cada tarefa terá apenas uma tarefa que a antecede na sequência. Pela restrição (45), se a tarefa *j* não for a σ-ézima tarefa da sequência e a tarefa *i* não for a (σ-1)-ézima tarefa da sequência, então $y_{ij\sigma}$ será maior ou igual a -1. Se a tarefa *j* é a σ-ézima tarefa na sequência e a tarefa *i* não precede diretamente a tarefa *j* na sequência, $y_{ij\sigma}$ será maior ou igual a zero. Finalmente, se a tarefa *j* é a σ-ézima tarefa na sequência e a tarefa *i* precede diretamente a tarefa *j* na sequência, $y_{ij\sigma}$ será maior ou igual a um. A restrição (46) garante que, para cada posição na sequência (σ), apenas um $y_{ij\sigma}$ será igual a um. Já a restrição (47) garante que nenhuma tarefa irá preceder a primeira tarefa da sequência. As restrições (48) e (49) são utilizadas para calcular a data de término do *setup* das máquinas. A restrição (48) é aplicada apenas à primeira tarefa da sequência, cuja data de término do *setup* depende exclusivamente da data de liberação da tarefa (representada por S_{jjk}). Por outro lado a restrição (49) é a fórmula geral da data de término do *setup* das máquinas, ou seja, a soma da data de término do processamento da (σ-1)-ézima tarefa da sequência e do tempo de *setup* da máquina *k* para a transição da (σ-1)-ézima para a σ-ézima tarefa da sequência. As restrições (50–52) são utilizadas para calcular a data de término do processamento das tarefas nas máquinas, considerando a possibilidade de existência de bloqueio. A restrição (50), ilustrada na Figura 9a, é aplicada a todas as tarefas em todas as máquinas exceto na última, ela é utilizada para verificar a existência ou não do bloqueio na máquina. Se $C_{\sigma k} > R_{\sigma,k+1}$, então não houve bloqueio na máquina e as restrições (51) ou (52)

determinarão o valor de $C_{\sigma k}$. se $C_{\sigma k} = R_{\sigma,k+1}$, então o valor do bloqueio é maior ou igual a zero. A restrição (51), ilustrada na Figura 9b, é aplicada a todas as tarefas apenas na primeira máquina, onde, por não haver ociosidade na máquina, a data de término do processamento das tarefas é igual à data de término do *setup* da máquina somado ao tempo de processamento da tarefa, caso não ocorra bloqueio. Para as demais máquinas é aplicada a restrição (52), que define a data de término do processamento das tarefas igual à data de término do processamento das tarefas na máquina anterior somado ao tempo de processamento da tarefa na máquina, caso não ocorra bloqueio, como mostrado na Figura 9c.

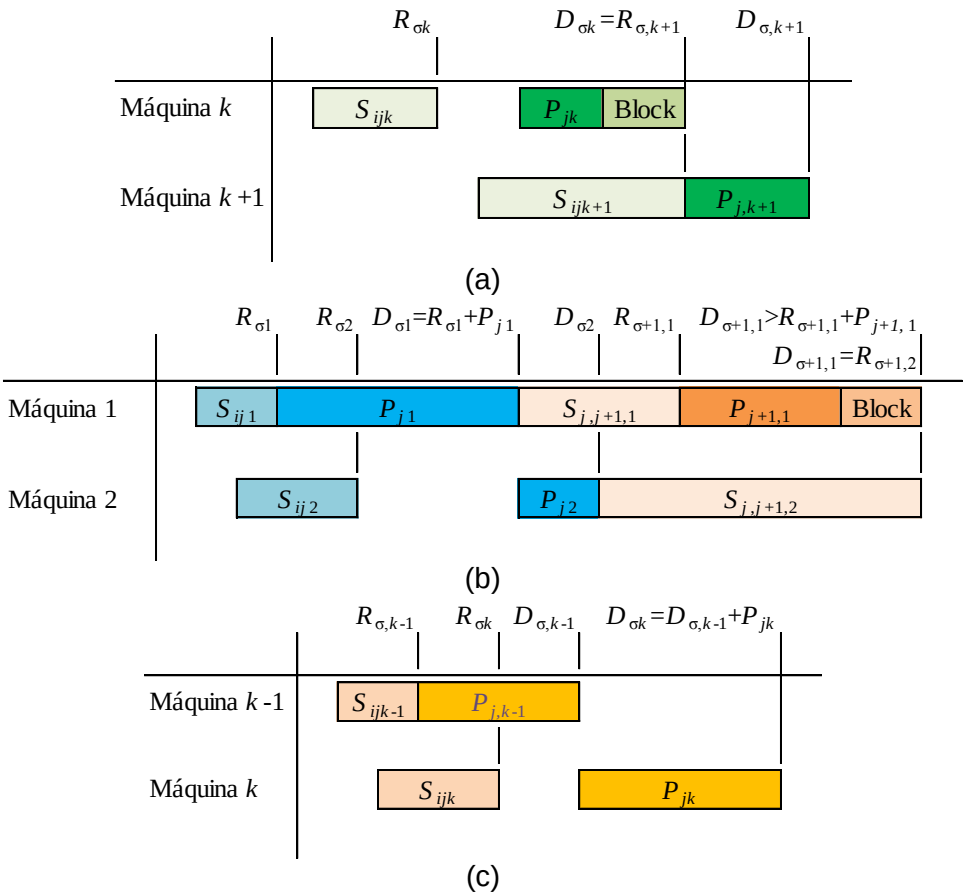

Figura 9 – Representação gráfico da (a) restrição 50; (b) restrição 51; e (c) restrição 52

6.1.2. Modelo TNZBS2

$$\text{Minimizar: } D_{max} = D_{nm} \qquad (55)$$

$$\text{Restrito a: } \sum_{j=1}^{n} x_{j\sigma} = 1 \qquad \forall \sigma = 1, \dots, n \quad (56)$$

$$\sum_{\sigma=1}^{n} x_{j\sigma} = 1 \qquad \forall j = 1, \dots, n \quad (57)$$

$$y_{ij\sigma} \geq x_{j\sigma} + x_{i,\sigma-1} - 1 \qquad \forall i = 1, \dots, n; j = 1, \dots, n; \sigma = 2, \dots, n; i \neq j \quad (58)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij\sigma} = 1 \qquad \forall \sigma = 2, \dots, n \quad (59)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij1} = 0 \quad (60)$$

$$R_{1k} = \sum_{i=1}^{n} \sum_{j=1} S_{ijk} * x_{j1} \qquad \forall k = 1, \dots, m \quad (61)$$

$$R_{\sigma k} = D_{\sigma-1,k} + \sum_{i=1}^{n} \sum_{j \neq i} S_{ijk} * y_{ij\sigma} \qquad \forall \sigma = 2, \dots, n; k = 1, \dots, m \quad (62)$$

$$e_{\sigma 1} = R_{\sigma 1} \qquad \forall \sigma = 1, \dots, n \quad (63)$$

$$e_{\sigma k} \geq D_{\sigma,k-1} \qquad \forall \sigma = 1, \dots, n; k = 2, \dots, m-1 \quad (64)$$

$$D_{\sigma,k-1} \geq R_{\sigma k} \qquad \forall \sigma = 1, \dots, n; k = 2, \dots, m \quad (65)$$

$$D_{\sigma k} = C_{\sigma k} + block_{\sigma k} \qquad \forall \sigma = 1, \dots, n; k = 1, \dots, m \quad (66)$$

$$block_{1k} \geq R_{1,k+1} - C_{1k} \qquad \forall k = 1, \dots, m-1 \quad (67)$$

$$block_{\sigma m} = 0 \qquad \forall \sigma = 1, \dots, n \quad (68)$$

$$C_{\sigma k} = e_{\sigma k} + \sum_{j=1}^{n} P_{jk} * x_{j\sigma} \qquad \forall \sigma = 1, \dots, n; k = 1, \dots, m \quad (69)$$

$$C_{\sigma k}, R_{\sigma k}, e_{\sigma 1}, D_{\sigma k}, block_{\sigma k} \geq 0 \text{ e inteiro} \qquad \forall \sigma = 1, \dots, n; k = 1, \dots, m \quad (70)$$

$$x_{j\sigma}, y_{ij\sigma} \in \{0,1\} \qquad \forall \mathrm{i} = 1, \dots, \mathrm{n}; \mathrm{j} = 1, \dots, \mathrm{n}; \sigma = 1, \dots, n \quad (71)$$

As restrições (63) e (64) definem a data de início do processamento das tarefas nas máquinas. A restrição (63) é aplicada somente à primeira máquina, onde não há ociosidade da máquina entre o término do *setup* e o início do processamento, portanto a data de início do processamento das tarefas é igual à data de término do *setup* da máquina. A restrição (64) é a fórmula geral para a data de início do processamento das tarefas, que é maior ou igual à data de término do processamento da tarefa na máquina anterior. Se $e_{\sigma k} = C_{\sigma,k-1}$ então não houve bloqueio na máquina *k*-1 nem ociosidade na máquina *k*, se $e_{\sigma k} > C_{\sigma,k-1}$, então houve bloqueio na máquina *k*-1, ou ocorreu ociosidade na máquina *k*, ou ocorreram os dois

casos (bloqueio na máquina *k*-1 e ociosidade na máquina *k*). As restrições (65) e (66) determinam a data em que as tarefas liberam as máquinas. A restrição (65) é aplicada a todas as máquinas exceto à primeira máquina e garante que nenhuma tarefa irá liberar a máquina até que a máquina seguinte já esteja preparada para recebê-la. Já a restrição (66), aplicada a todas as máquinas, define a data da liberação das tarefas igual à data de término do processamento da tarefa mais o tempo de bloqueio da máquina. As restrições (67 e 68) são utilizadas para calcular o tempo de bloqueio das máquinas. A restrição (68) garante que não ocorrerá bloqueio na última máquina, e a restrição (67) calcula o tempo de bloqueio nas demais máquinas para a primeira tarefa da sequência. A restrição (69) é utilizada para calcular a data de término do processamento das tarefas nas máquinas, sendo esta igual à data de início do processamento das tarefas mais o tempo de processamento das tarefas nas máquinas.

6.2. Modelos Adaptados

Ronconi e Birgin (2012) propuseram dois modelos para o problema *flow shop* permutacional com bloqueio e com o objetivo de minimizar o *earlyness* e o *tardiness*. O primeiro e o segundo modelos adaptados para o problema *flow shop* permutacional com bloqueio e *setup* dependente e com o objetivo de minimizar o *makespan* (RBZBS1 e RBZBS2) são apresentados a seguir.

6.2.1. Modelo RBZBS1

$$\text{Minimizar: } D_{max} = D_{nm} \tag{72}$$

$$\text{Restrito a: } \sum_{j=1}^{n} x_{j\sigma} = 1 \qquad \forall \sigma = 1, \dots, n \tag{73}$$

$$\sum_{\sigma=1}^{n} x_{j\sigma} = 1 \qquad \forall j = 1, \dots, n \tag{74}$$

$$y_{ij\sigma} \geq x_{j\sigma} + x_{i,\sigma-1} - 1 \qquad \forall i = 1, \dots, n; j = 1, \dots, n; \sigma = 2, \dots, n; i \neq j \tag{75}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij\sigma} = 1 \qquad \forall \sigma = 2, \dots, n \tag{76}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij1} = 0 \tag{77}$$

$$empty_{\sigma 1} = 0 \qquad \forall \sigma = 1, \dots, n \quad (78)$$

$$block_{\sigma m} = 0 \qquad \forall \sigma = 1, \dots, n \quad (79)$$

$$\sum_{i=1}^{n} \sum_{j=i} S_{ijk} * x_{j1} + empty_{1k} + \sum_{j=1}^{n} P_{jk} * x_{j1} + block_{1k} = \sum_{i=1}^{n} \sum_{j=i} S_{i,j,k+1} * x_{j1} + empty_{1,k+1} \qquad \forall k = 1, \dots, m-1 \quad (80)$$

$$\sum_{i=1}^{n} \sum_{j \neq i} S_{ijk} * y_{i,j,\sigma+1} + empty_{\sigma+1,k} + \sum_{j=1}^{n} P_{jk} * x_{j,\sigma+1} + block_{\sigma+1,k} = \sum_{j=1}^{n} P_{j,k+1} * x_{j,\sigma} + block_{\sigma,k+1} + \sum_{i=1}^{n} \sum_{j \neq i} S_{i,j,k+1} * y_{i,j,\sigma+1} + empty_{\sigma+1,k+1}$$

$$\forall \sigma = 1, \dots, \mathrm{n}-1; k = 1, \dots, m-1 \quad (81)$$

$$D_{1m} \geq \sum_{i=1}^{n} \sum_{j=i} S_{i,j,1} * x_{j1} + \sum_{k=1}^{m} \left(block_{1k} + \sum_{j=1}^{n} \left(P_{jk} * x_{j1} \right) \right) \quad (82)$$

$$D_{\sigma m} \geq C_{\sigma-1,m} + \sum_{i=1}^{n} \sum_{j \neq i} S_{ijm} * y_{ij\sigma} + empty_{\sigma m} + \sum_{j=1}^{n} P_{jk} * x_{j\sigma}$$

$$\forall \sigma = 2, \dots, \mathrm{n} \quad (83)$$

$$D_{\sigma k}, empty_{\sigma k}, block_{\sigma k} \geq 0 \text{ e inteiro}$$

$$\forall \sigma = 1, \dots, n; k = 1, \dots, m \quad (84)$$

$$x_{j\sigma}, y_{ij\sigma} \in \{0,1\}$$

$$\forall \mathrm{i} = 1, \dots, \mathrm{n}; \mathrm{j} = 1, \dots, \mathrm{n}; \sigma = 1, \dots, n \quad (85)$$

As restrições (78) e (79) garantem que não haja ociosidade na primeira máquina nem bloqueio na última máquina respectivamente. As restrições (80) e (81) estabelecem a relação existente entre a ociosidade, o bloqueio, o tempo de processamento e o tempo de *setup*. A restrição (80) estabelece esta relação para a primeira tarefa da sequência e é ilustrada na Figura 10. A restrição (81) estabelece a relação para as demais tarefas exceto para a última tarefa da sequência e é ilustrada na Figura 11. As restrições (82) e (83) são utilizadas para calcular a data de término do processamento das tarefas na última máquina. A restrição (82) é utilizada para o cálculo da data de término do processamento da primeira tarefa da sequência na última máquina, que é igual à soma do tempo de *setup* na primeira máquina, dos tempos de bloqueio de todas as máquinas e o tempo de processamento em todas as máquinas. A restrição (83) é a fórmula geral do cálculo da data de término do processamento das tarefas na última máquina, que é a soma da data de término do processamento da tarefa anterior na máquina, do tempo de *setup* da última máquina, o tempo de ociosidade e o tempo de processamento da tarefa.

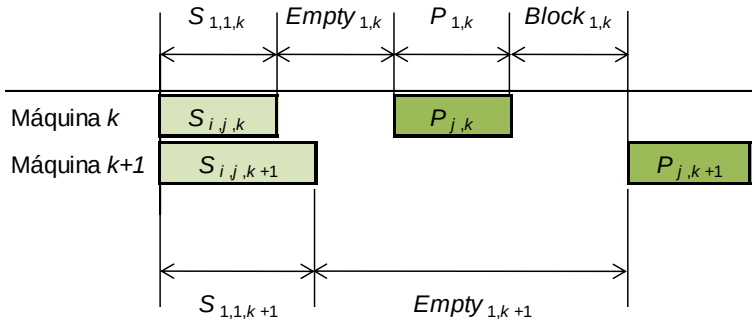

Figura 10 – Representação gráfica da restrição (80) que expressa a relação existente entre o tempo de *setup*, o tempo de processamento, o tempo de ociosidade e o tempo de bloqueio para a primeira tarefa da sequência

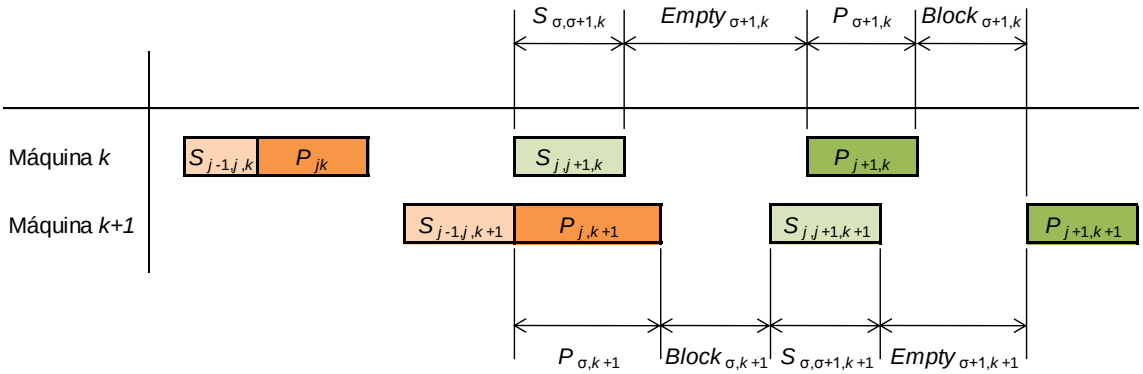

Figura 11 – Representação gráfica da restrição (81) que expressa a relação existente entre o tempo de *setup*, o tempo de processamento, o tempo de ociosidade e o tempo de bloqueio para as demais tarefas

6.2.2. Modelo RBZBS2

$$\text{Minimizar: } D_{max} = D_{nm} \quad (86)$$

$$\text{Restrito a: } \sum_{j=1}^{n} x_{j\sigma} = 1 \qquad \forall \sigma = 1, \dots, n \quad (87)$$

$$\sum_{\sigma=1}^{n} x_{j\sigma} = 1 \qquad \forall j = 1, \dots, n \quad (88)$$

$$y_{ij\sigma} \geq x_{j\sigma} + x_{i,\sigma-1} - 1 \qquad \forall i = 1, \dots, n; j = 1, \dots, n; \sigma = 2, \dots, n; i \neq j \quad (89)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij\sigma} = 1 \qquad \forall \sigma = 2, \dots, n \quad (90)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij1} = 0 \quad (91)$$

$$D_{11} \geq \sum_{i=1}^{n} \sum_{j=1} S_{i,j,1} * x_{j1} + \sum_{j=1}^{n} P_{j1} * x_{j1} \quad (92)$$

$$D_{1k} \geq D_{1,k-1} + \sum_{j=1}^{n} P_{jk} * x_{j1} \qquad \forall k = 2, \dots, m \quad (93)$$

$$D_{1k} \geq \sum_{i=1}^{n} \sum_{j=1} S_{i,j,k+1} * x_{j1} \qquad \forall k = 1, \dots, m-1 \quad (94)$$

$$D_{\sigma k} \geq D_{\sigma-1,k} + \sum_{i=1}^{n} \sum_{j \neq i} S_{ijk} * y_{ij\sigma} + \sum_{j=1}^{n} P_{jk} * x_{j\sigma}$$

$$\forall \sigma = 2, \dots, n; k = 1, \dots, m \quad (95)$$

$$D_{\sigma k} \geq D_{\sigma,k-1} + \sum_{j=1}^{n} P_{jk} * x_{j\sigma}$$

$$\forall \sigma = 2, \dots, n; k = 2, \dots, m \quad (96)$$

$$D_{\sigma k} \geq D_{\sigma-1,k+1} + \sum_{i=1}^{n} \sum_{j \neq i} S_{i,j,k+1} * y_{ij\sigma}$$

$$\forall \sigma = 2, \dots, n; k = 1, \dots, m-1 \quad (97)$$

$$D_{\sigma k} \geq 0 \text{ e inteiro}$$

$$\forall \sigma = 1, \dots, n; k = 1, \dots, m \quad (98)$$

$$x_{j\sigma}, y_{ij\sigma} \in \{0,1\}$$

$$\forall \mathrm{i} = 1, \dots, \mathrm{n}; \mathrm{j} = 1, \dots, \mathrm{n}; \sigma = 1, \dots, n \quad (99)$$

As restrições (92–97) são utilizadas para calcular a data de término do processamento das tarefas nas máquinas, considerando a possibilidade de bloqueio nas máquinas. A restrição (92) é aplicada apenas à primeira tarefa da sequência na primeira máquina, definindo que a data de término do processamento dessa tarefa nessa máquina é maior ou igual ao tempo de *setup* da primeira máquina somado ao tempo de processamento dessa tarefa nessa máquina. A restrição (93) é aplicada à primeira tarefa da sequência da segunda à última máquina, definindo a data de término do processamento da tarefa nas máquinas maior ou igual à data de término do processamento da tarefa na máquina anterior somada ao tempo de processamento da tarefa na máquina. A restrição (94) é aplicada ainda à primeira tarefa da sequência nas demais máquinas, exceto à última, e define a data de término do processamento da tarefa nas máquinas maior ou igual ao tempo de *setup* da máquina seguinte. Em conjunto as equações (92–94) definem as datas de término do processamento da primeira tarefa da sequência nas máquinas, caso a restrição (92) para a primeira máquina ou a restrição (93) para as demais máquinas limitem os valores das datas de término, não haverá bloqueio nas máquinas. Quando a restrição (94) é a restrição limitante para o valor da data de término do processamento, então pode ocorrer bloqueio. A restrição (95) garante que a data de término do processamento das demais tarefas da sequência seja maior ou igual à data de término do processamento da tarefa anterior somada ao tempo de *setup* da máquina e ao tempo de processamento da tarefa. A restrição (96) garante que a data de término do processamento das demais tarefas da segunda à última máquina seja maior ou igual à data de término do processamento da tarefa na máquina

anterior somada ao tempo de processamento da tarefa na máquina. Finalmente a restrição (97) garante que a data de término do processamento das demais tarefas nas máquinas, exceto na última, seja maior ou igual à data de término do processamento da tarefa anterior na sequência na máquina posterior somada ao tempo de *setup* da tarefa na máquina seguinte. Quando as restrições (95) ou (96) limitam a data de término do processamento da tarefa na máquina significa que não ocorreu bloqueio. Por outro lado quando a restrição (97) limita a data de término do processamento da tarefa na máquina, então pode ocorrer bloqueio.

As características dos quatro modelos apresentados para o problema de *flow shop* com *buffer* zero e *setup* dependente da sequência e da máquina, com objetivo de minimizar o *makespan* são apresentadas na Tabela 5. As características apresentadas são relacionadas ao tamanho de cada modelo, expressado pelo número de restrições e de variáveis binárias e contínuas.

Tabela 5 – Número de variáveis e de restrições dos modelos apresentados

Modelo	Variáveis Binárias	Variáveis Contínuas	Restrições
TNZBS1	$n^3 + n^2$	$2nm + 1$	$n^3 - 2n^2 + 3n + 3nm$
TNZBS2	$n^3 + n^2$	$5nm + 1$	$n^3 - 2n^2 + 3n + 5nm + m - 1$
RBZBS1	$n^3 + n^2$	$2nm + n + 1$	$n^3 - 2n^2 + 6n + nm$
RBZBS2	$n^3 + n^2$	$nm + 1$	$n^3 - 2n^2 + 2n + 3nm - m + 1$

7. MODELOS BRANCH-AND-BOUND

O algoritmo do *B&B* utilizado foi proposto por Kim (1995), onde cada nó representa uma sequência parcial da resposta final. Essa sequência é denominada |*PS*|, e o conjunto de tarefas que não fazem parte desta sequência é denominado |*NPS*|. Quando um nó é ramificado uma ou mais sequências parciais (nós) são criadas adicionando-se uma nova tarefa de |*NPS*| à sequência parcial associada ao nó ramificado. Para cada nó criado é calculado um limitante inferior do *makespan* para a sequência parcial. A seleção do nó a ser ramificado foi usada por Ronconi (2005) sendo realizada pela regra *depth first*, onde o nó com o maior número de tarefas na sequência parcial é selecionado, em caso de empate, o nó com o menor valor de limitante inferior do *makespan* é selecionado. Este método de seleção de nó foi utilizado devido ao sucesso de sua aplicação no problema de *flow shop* permutacional com bloqueio por Ronconi (2005).

Na Figura 12 é apresentado um exemplo da execução do *B&B* com três tarefas. Inicialmente todos os nós da primeira camada são criados, neles cada sequência parcial é constituída de apenas uma única tarefa. Em seguida é calculado um limitante inferior (*LB*) para cada nó. Pela regra *depth first*, o nó com o maior número de tarefas na sequência parcial é ramificado, como todos os nós possuem a mesma quantidade de tarefas na sequência parcial, aquela que obtiver o menor resultado de *LB* é então ramificado, gerando, nesse caso, mais dois nós na segunda e última camada. Calcula-se, então, o limitante inferior para cada um dos nós da segunda camada. Novamente é utilizada a regra de seleção de nó para ser ramificado. Por se tratar da última camada da árvore, não há como ramificar nenhum dos nós, portanto são analisados os nós com o maior número de tarefas na sequência parcial que ainda podem ser ramificados (nesse caso, as tarefas da primeira camada). Como há, nesse exemplo, um nó cujo valor de *LB* é menor que o menor valor de *LB* obtido na última camada, ele também será ramificado, gerando, novamente, mais dois nós na segunda e última camada. Mais uma vez são calculados os limitantes inferiores para todos os nós gerados. Realizando a análise dos nós da camada anterior, pode-se verificar que não há nenhum valor de *LB* menor que o menor valor de *LB* obtido na última camada. Portanto, pode-se concluir

com certeza que, nesse caso, a sequência de tarefas {1,2,3} é a que resultará no menor valor do *makespan*.

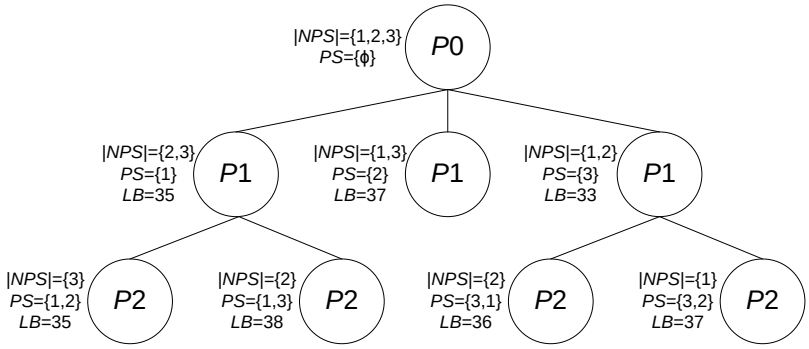

Figura 12 – Exemplo de uma árvore *Branch-and-Bound*

7.1. Limitante inferior para o makespan

Considerando uma sequência parcial *PS*, o limitante inferior para o *makespan* é o valor mínimo para a data de término da última tarefa dentre as tarefas ainda não sequenciadas |*NPS*| na última máquina (*m*). Quatro limitantes inferiores para o *makespan* foram desenvolvidos para o problema *Fm* | *prmu,Sijk*, *block* levando em consideração a utilização do limitante inferior do bloqueio (LBB_{ij}^{k}). Seja *PS* a última tarefa na sequência parcial e |*NPS*| o conjunto de tarefas que não são parte da sequência parcial. Seja também $S\_S_{i,j,k}$ a soma dos tempos mínimos de *setup* das tarefas que não são parte da sequência parcial na máquina *k*, e $S\_LBB_{i,j,k}$ a soma dos valores mínimos dos limitantes inferiores do bloqueio das tarefas que não são parte da sequência parcial na máquina *k*. $S\_S_{i,j,k}$ e $S\_LBB_{i,j,k}$ são obtidos pelas equações 100 e 101.

$$S\_S_{i,j,k} = \sum_{i \in |NPS| \cup |PS|} \left(min\left(S_{i,|NPS|,k}\right)\right) - max\left(min\left(S_{|NPS|,|NPS|,k}\right)\right) \quad (100)$$

$$S\_LBB_{i,j,k} = \sum_{i \in |NPS| \cup |PS|} \left(min\left(LBB_{i,|NPS|}^{k}\right)\right) - max\left(min\left(LBB_{|NPS|,|NPS|}^{k}\right)\right) \quad (101)$$

Seja LW o tempo de processamento somado ao valor do limitante inferior do tempo de bloqueio da última tarefa da sequência. A última tarefa da sequência é aquela com o menor valor da soma dos tempos de processamento e dos limitantes inferiores do bloqueio. Ou seja:

$$LW(k) = \min\left(\sum_{q=k+1}^{m}\left(\mathrm{P}_{|\mathrm{NPS}|,\mathrm{q}}\right) + min\left(\sum_{q=k+1}^{m}\left(\mathrm{LBB}^{\mathrm{q}}_{|\mathrm{NPS}|,|\mathrm{NPS}|}\right)\right)\right) \quad (102)$$

A última tarefa da sequência é removida do conjunto |*NPS*|, a este novo conjunto de tarefas é dado o nome de |*NPSn*|. Com isto é possível calcular $S\_Sn_{i,j,k}$, que é a soma dos tempos mínimos de *setup* das tarefas que não fazem parte da sequência parcial nem da última tarefa da sequência na máquina *k*, e $S\_LBBn_{i,j,k}$, que é a soma dos valores mínimos dos limitantes inferiores do bloqueio das tarefas que não são parte da sequência parcial nem da última tarefa da sequência na máquina *k*, utilizando as seguintes equações:

$$S\_Sn_{i,j,k} = \sum_{i\in|NPSn|\cup|PS|}\left(min\left(S_{i,|NPS|,k}\right)\right) - max\left(min\left(S_{|NPSn|,|NPS|,k}\right)\right) \quad (103)$$

$$S\_LBBn_{i,j,k} = \sum_{i\in|NPSn|\cup|PS|}\left(min\left(LBB^{k}_{i,|NPS|}\right)\right) - max\left(min\left(LBB^{k}_{|NPSn|,|NPS|}\right)\right) \quad (104)$$

Os limitantes inferiores para o *makespan* foram nomeados de LB_{TN1}, LB_{TN2}, LB_{TN3} e LB_{TN4} e são obtidos pelas seguintes equações:

$$LB_{TN1}(k) = C_{|PS|k} + \mathrm{S\_S}_{i,j,k} + \sum_{h\in|NPS|}(P_{hk}) + \mathrm{S\_LBB}^{k}_{i,j} + min\left(\sum_{q=k+1}^{m}\left(P_{|NPS|,q} + min\left(LBB^{q}_{|NPS|,|NPS|}\right)\right)\right) \quad (105)$$

$$LB_{TN2}(k) = C_{|PS|k} + S\_S_{i,j,k} + \sum_{h\in|NPS|}(P_{hk}) + \mathrm{S\_LBB}^{k}_{i,j} + min\left(\sum_{q=k+1}^{m}\left(P_{|NPS|,q}\right) + min\left(\sum_{q=k+1}^{m}\left(LBB^{q}_{|NPS|,|NPS|}\right)\right)\right) \quad (106)$$

$$LB_{TN3}(k) = C_{|PS|k} + S\_Sn_{i,j,k} + \sum_{h\in|NPS|}(P_{hk}) + \mathrm{S\_LBBn}^{k}_{i,j} + LW(k) \quad (107)$$

$$LB_{TN4}(k) = C_{|PS|k} + S\_S_{i,j,k} + \sum_{h\in|NPS|}(P_{hk}) + \mathrm{S\_LBB}^{k}_{i,j} + \sum_{q=k+1}^{m}\left(min\left(P_{|NPS|,q}\right) + min\left(LBB^{q}_{|NPS|,|NPS|}\right)\right) \quad (108)$$

$$LB_{TN1} = max\left(LB_{TN1}(k)\right) \quad (109)$$

$$LB_{TN2} = max\left(LB_{TN2}(k)\right) \quad (110)$$

$$LB_{TN3} = max\left(LB_{TN3}(k)\right) \quad (111)$$

$$LB_{TN4} = max\left(LB_{TN4}(k)\right) \quad (112)$$

Todos os limitantes inferiores são obtidos em duas etapas, inicialmente são calculados os limitantes inferiores de cada nó para cada máquina, em seguida, dentre os limitantes inferiores das máquinas, aquele que obtiver o maior valor é definido como sendo o limitante inferior do nó.

Todos os limitantes inferiores são similares, a diferença entre eles é a forma como é tentado definir qual é a última tarefa da sequência. Nos limitantes inferiores LB_{TN1}, LB_{TN2} e LB_{TN4} calcula-se inicialmente a data de término da última tarefa na sequência parcial na máquina *k* ($C_{|PS|k}$), a esse valor é somado o valor do tempo de processamento e também os tempos de *setup* ($\mathrm{S\_}S_{i,j,k}$) e do limitante inferior do bloqueio ($\mathrm{S\_LBB}_{i,j}^{k}$) de todas as tarefas que ainda não foram sequenciadas. Ao resultado dessa soma é adicionado os valores dos tempos de processamento e dos limitantes inferior do bloqueio da última tarefa nas máquinas subsequentes.

Já no limitante inferior LB_{TN3} calcula-se inicialmente a soma dos tempos de processamento e dos valores dos limitantes inferior do bloqueio da última tarefa nas máquinas subsequentes ($LW(k)$). A tarefa que obtiver o menor valor de $LW(k)$ é definida como sendo a última tarefa da sequência, portanto ela é removida dos cálculos da soma do tempo de *setup* e do limitante inferior do bloqueio. Desta forma, evita-se que seja utilizado o limitante inferior do bloqueio novamente nos cálculos e, também, que se utilize o tempo de *setup* da transição da última tarefa da sequência para alguma outra tarefa. Na Figura 13 é apresentado um exemplo de como é realizado o cálculo do limitante inferior para uma determinada máquina *k*. Onde $\mathrm{S\_}S_{i,j,k} = S_{|PS|jk} + S_{j,j+1,k}$ e $\mathrm{S\_LBB}_{i,j}^{k}$ é igual à soma dos limitantes inferiores dos tempos de bloqueio que ocorrem na máquina *k*.

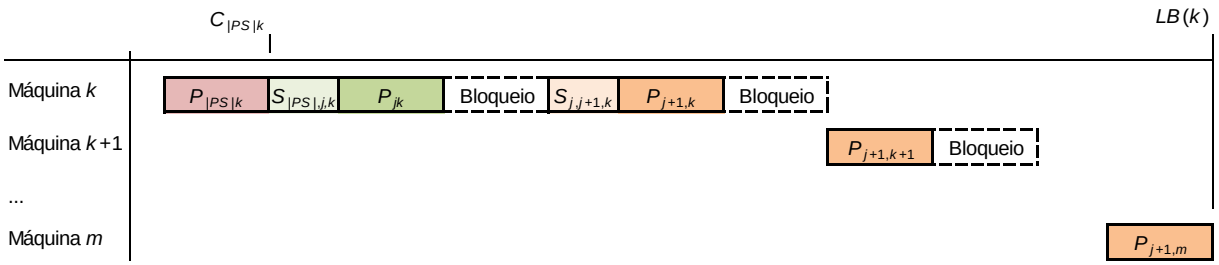

Figura 13 – Exemplo do cálculo do limitante inferior do *Branch-and-Bound*

8. MÉTODOS HEURÍSTICOS

Neste capítulo são apresentados os métodos heurísticos adaptados para o problema de bloqueio com *setup* dependente da sequência e da máquina. Os métodos heurísticos foram classificados de acordo com o método de classificação apresentado por Framinan *et al.* (2004). Na Tabela 6 são apresentados todos os métodos heurísticos adaptados de acordo com sua classificação.

Tabela 6 – Métodos heurísticos adaptados

Método heurístico adaptado	Fase	Método heurístico original	Autor do método original
MM	Fase I	*MM*	Ronconi (2014)
*MM*1	Fase I		
PF	Fase I	*PF*	McCormick *et al.* (1989)
*PF*1	Fase I		
wPF	Fase I	*wPF*	Pan and Wang (2012)
*wPF*1	Fase I		
PW	Fase I	*PW*	
*PW*1	Fase I		
MME	Fase II	*MME*	Ronconi (2014)
*MM*1*E*	Fase II		
PFE	Fase II	*PFE*	
*PF*1*E*	Fase II		
wPFE	Fase II	*wPFE*	Pan and Wang (2012)
*wPF*1*E*	Fase II		
PWE	Fase II	*PWE*	
*PW*1*E*	Fase II		
PF-NEH(x)	Fase II	*PF-NEH*(x)	Pan and Wang (2012)
*PF*1-*NEH*(x)	Fase II		
wPF-NEH(x)	Fase II	*wPF-NEH*(x)	
*wPF*1-*NEH*(x)	Fase II		
PW-NEH(x)	Fase II	*PW-NEH*(x)	
*PW*1-*NEH*(x)	Fase II		
PF-NEH$_{ls}$(x)	Fase III	*PF-NEH*$_{ls}$(x)	
*PF*1-*NEH*$_{ls}$(x)	Fase III		
wPF-NEH$_{ls}$(x)	Fase III	*wPF-NEH*$_{ls}$(x)	
*wPF*1-*NEH*$_{ls}$(x)	Fase III		
PW-NEH$_{ls}$(x)	Fase III	*PW-NEH*$_{ls}$(x)	
*PW*1-*NEH*$_{ls}$(x)	Fase III		

8.1. Heurísticas de I Fase

Os modelos heurísticos de I fase utilizados neste trabalho são: *MM*, proposto por Ronconi (2004); *PF*, proposto por McCormick *et al.* (1989); *wPF* e *PW*, propostos por Pan e Wang (2012).

8.1.1. Heurística *MM*

O método MinMax(*MM*) proposto por Ronconi (2004) sequencia as tarefas baseando-se nas propriedades do *makespan* do problema, propostas por Ronconi e Armentano (2001). Inicialmente, a heurística *MM* define a primeira e a última tarefa da sequência de acordo com a regra de Johnson (Johnson, 1954), ou seja, a primeira tarefa da sequência é a tarefa que possui o menor tempo de processamento na primeira máquina, e a última tarefa da sequência é a tarefa que possui o menor tempo computacional na última máquina. A próxima tarefa da sequência, começando pela segunda posição da sequência, é a tarefa com o menor valor de caminho crítico. Este caminho crítico é composto pela soma dos valores máximos entre o tempo de processamento da máquina consecutiva e pela soma dos *m* tempos de processamento. Ou seja, a tarefa seguinte à tarefa já sequenciada (*i*) é a tarefa (*j*) que obtiver o menor valor na seguinte expressão:

$$\alpha \sum_{l=1}^{m-1} \left| P_{jl} - P_{i,l+1} \right| + (1-\alpha) \sum_{k=1}^{m} P_{jk} \qquad (113)$$

onde α é um parâmetro que pondera os dois termos da expressão.

O mesmo método é aplicado ao problema de *flow shop* permutacional com bloqueio e *setup* dependente da sequência e da máquina. A primeira e a última tarefa da sequência são as tarefas com o menor tempo de processamento na primeira e na última máquina, respectivamente. Então a equação 113 é usada para definir a posição das demais tarefas na sequência. O parâmetro α foi definido em 0,6, visto que o método apresenta seu melhor desempenho para este valor de α (Ronconi, 2004).

Ainda outra adaptação do método é proposta. Como o tempo de *setup* também é uma variante do problema, ele também foi considerado nos cálculos. Este método adaptado foi nomeado como *MM*1. Nele, a primeira e a última tarefa da sequência são as tarefas que possuem a menor soma do tempo de processamento e do tempo de *setup* na primeira máquina e na última máquina, respectivamente. A segunda tarefa da sequência é a tarefa (*c*) que obtiver o menor valor na seguinte expressão:

$$\alpha \sum_{l=1}^{m-1} \left| \left(P_{jl} + S_{ijl}\right) - \left(P_{i,l+1} + S_{0il}\right) \right| + (1-\alpha) \sum_{k=1}^{m} \left(P_{jk} + S_{ijk}\right) \qquad (114)$$

onde S_{0il} é o tempo de *setup* da primeira tarefa da sequência na máquina *i*.

A próxima tarefa na sequência, começando pela terceira, é a tarefa que obtiver o menor valor na seguinte expressão:

$$\alpha \sum_{l=1}^{m-1} \left| \left(P_{jl} + S_{ijl}\right) - \left(P_{i,l+1} + S_{hil}\right) \right| + (1-\alpha) \sum_{k=1}^{m} \left(P_{jk} + S_{ijk}\right) \qquad (115)$$

onde *h* é a tarefa que precede a tarefa *i* na sequência.

Como cada adaptação fornece resultados diferentes uma da outra, elas foram nomeadas de *MM* e *MM*1. Ambos os métodos adaptados são comparados no capítulo 9.

8.1.2. Heurística *PF*

O método *Profile Fitting* (*PF*) proposto por McCormick *et al.* (1989) tenta encontrar a sequência que minimiza o tempo de ociosidade e de bloqueio nas máquinas. A primeira tarefa na sequência é a tarefa que obtiver a menor soma dos tempos de processamento nas máquinas. A próxima posição na sequência (*c*) pertence à tarefa (*j*) que obtiver o menor valor de δ_j, que é obtido pela seguinte expressão:

$$\delta_j = \sum_{k=1}^{m} \left(C_{[c]k} - C_{[i]k} - P_{jk}\right) \qquad (116)$$

A equação 116 calcula a soma do tempo de ociosidade e de bloqueio das máquinas. Na Figura 14 é apresentada uma comparação entre o problema sem tempo de *setup* (Figura 14a) e o problema com tempo de *setup* (Figura 14b). Como pode ser visto na Figura 14b, o tempo de *setup* também deve ser subtraído do resultado para calcular a soma dos tempos de ociosidade e de bloqueio. Portanto, para adaptar o método para o problema estudado neste trabalho, δ_j é calculado pela seguinte expressão:

$$\delta_j = \sum_{k=1}^{m}\left(C_{[c]k} - C_{[i]k} - P_{jk} - S_{ijk}\right) \quad (117)$$

onde S_{ijk} é o tempo de *setup* entre a última tarefa da sequência parcial e a tareja (*j*).

A última tarefa da sequência é a tarefa com a menor soma dos tempos de processamento nas máquinas.

Outra forma de se adaptar o modelo é considerando o tempo de *setup* nos cálculos. Como o tempo de *setup* depende da sequência, a primeira tarefa na sequência é aquela com a menor soma dos tempos de processamento e dos tempos de *setup* em todas as máquinas. A tarefa seguinte na sequência, começando da segunda posição, é aquela que obtiver a menor soma dos tempos de ociosidade, bloqueio e de *setup*. Ou seja, a próxima tarefa da sequência é aquela que obtiver o menor valor de δ_j, que é calculada pela equação 118. Note que matematicamente a equação 118 é igual à equação 116, porém a interpretação de ambas é diferente.

$$\delta_j = \sum_{k=1}^{m}\left(C_{[c]k} - C_{[i]k} - P_{jk}\right) \quad (118)$$

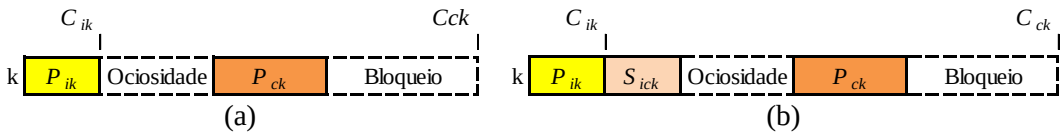

Figura 14 – Comparação entre os tempos de ociosidade e de bloquei a) sem *setup*; e b) com *setup*

Como cada adaptação fornece resultados diferentes uma da outra, elas foram nomeadas de *PF* e *PF*1. Ambos os métodos adaptados são comparados no capítulo 9.

8.1.3. Heurística *wPF*

A heurística *wPF* proposta por Pan e Wang (2012) funciona de forma semelhante à heurística *PF*. No entanto, é aplicado um peso diferente para cada máquina e diferentes posições na sequência. Isso porque o tempo de ociosidade ou de bloqueio das primeiras máquinas pode levar a maiores atrasos no processo, e o tempo de ociosidade ou de bloqueio das primeiras tarefas da sequência pode ter um efeito mais significativo para o *makespan* do que o tempo de ociosidade ou de bloqueio das últimas tarefas da sequência.

A primeira tarefa da sequência é a tarefa com a menor soma do tempo de processamento nas máquinas, a próxima posição na sequência (*c*) pertence à tarefa (*j*) com a menor soma ponderada dos tempos de ociosidade e de bloqueio nas máquinas, ou seja, a tarefa com o menor valor de δ_j, que é dado por:

$$\delta_j = \sum_{k=1}^{m} w_k \left(C_{[c]k} - C_{[i]k} - P_{jk}\right) \tag{119}$$

onde w_k é o peso da máquina e da posição da sequência, e é dado por:

$$w_k = m/\left(k + \left((c-1)(m-k)\right)/(n-1)\right) \tag{120}$$

onde *c* é a posição da sequência na qual a tarefa esta sendo posicionada.

Para adaptar o método para o problema, δ_j foi alterado para que a soma ponderada dos tempos de ociosidade e de bloqueio em todas as máquinas fosse calculada para o problema com *setup*. Então:

$$\delta_j = \sum_{k=1}^{m} w_k \left(C_{[c]k} - C_{[i]k} - P_{jk} - S_{ijk}\right) \tag{121}$$

A primeira tarefa na sequência é a tarefa com a menor soma dos tempos de processamento nas máquinas.

Outra forma de se adaptar o método é considerando o tempo de *setup* nos cálculos, já que ele é uma variável que depende da sequência e da máquina. Então, a primeira tarefa da sequência é a tarefa que obtiver a menor soma dos tempos de

processamento e dos tempos de *setup* em todas as máquinas. A próxima posição da sequência (*c*) pertence à tarefa (*j*) com a menor soma ponderada dos tempos de *setup*, ociosidade e bloqueio, ou seja, a tarefa que obtiver o menor valor de δ_j, que é dado por:

$$\delta_j = \sum_{k=1}^{m} w_k \left(C_{[c]k} - C_{[i]k} - P_{jk} \right) \quad (122)$$

Novamente a equação 122 é matematicamente igual à equação 119, porém a interpretação de ambas é diferente.

Como cada adaptação fornece resultados diferentes uma da outra, elas foram nomeadas de *wPF* e *wPF*1. Ambos os métodos adaptados são comparados no capítulo 9.

8.1.4. Heurística PW

A heurística *PW* proposta por Pan e Wang (2012) também tenta fornecer a sequência que minimize a soma dos tempos de ociosidade e de bloqueio. No entanto, no *PW* a tarefa que será atribuída a uma posição é selecionada por uma função que tem duas partes: 1. Calcular a soma ponderada dos tempos de ociosidade e de bloqueio da tarefa que está sendo testada; e 2. Calcular os efeitos das tarefas restantes (aquelas que ainda não foram sequenciadas) nos tempos de ociosidade e de bloqueio. A primeira parte do método é similar à heurística *wPF*. Para a segunda parte uma tarefa artificial (*v*) é criada, essa tarefa artificial é então atribuída à posição (*c*+1) da sequência. O tempo de processamento da tarefa *v* é a média dos tempos de processamento de todas as tarefas que ainda não foram sequenciadas (*NS*), ou seja:

$$P_{vk} = \sum_{\substack{q \in NS \\ q \neq j}} P_{qk} / (n - c) \quad (123)$$

A segunda parte do método utiliza P_{vk} para calcular $C_{[c+1],k}$. A soma dos tempos de ociosidade e de bloqueio causados pelas tarefas que ainda não foram sequenciadas (X_j) é então calculada por:

$$X_j = \sum_{k=1}^{m} w_k \left(C_{[c+1]k} - C_{[c]k} - P_{vk} \right) \quad (124)$$

Combinando a soma dos tempos de ociosidade e de bloqueio causados pela inserção da tarefa *j* na posição (*c*) da sequência e a soma dos tempos de ociosidade e de bloqueio causados pelas tarefas que ainda não foram sequenciadas (*v*), é calculado f_j:

$$f_j = (n - c - 1)\delta_j + X_j \quad (125)$$

onde $(n - c - 1)$ é uma ponderação usada para balancear os efeitos causados pelos tempos de ociosidade e de bloqueio nas últimas tarefas.

À medida que mais tarefas são inseridas na sequência parcial, mais preciso será o valor de X_j, então, para as primeiras tarefas, X_j tem um efeito menor no cálculo de f_j e seu efeito vai gradualmente aumentando para as últimas tarefas. Então, a tarefa (*j*) que obtiver o menor valor de f_j é inserida na posição (*c*) da sequência.

Esse índice f_j também é usado para determinar a primeira tarefa da sequência, no entanto, δ_j da primeira tarefa da sequência é dado por:

$$\delta_j = \sum_{k=1}^{m} w_k \left(C_{[c]k} - P_{jk} \right) \quad (126)$$

A primeira adaptação do método assume que δ_j e X_j são as somas ponderadas dos tempos de ociosidade e de bloquei das tarefas *j* e *v*, respectivamente. Então, δ_j foi alterado para a seguinte expressão:

$$\delta_j = \sum_{k=1}^{m} w_k \left(C_{[c]k} - C_{[i]k} - P_{jk} - S_{ijk} \right) \quad (127)$$

E X_j foi adaptado para a seguinte expressão:

$$X_j = \sum_{k=1}^{m} w_k \left(C_{[c+1]k} - C_{[c]k} - P_{vk} - S_{jvk} \right) \quad (128)$$

Sendo que,

$$S_{jvk} = \sum_{\substack{q \in NS \\ q \neq j}} S_{jqk}/(n-c) \quad (129)$$

A segunda adaptação do método usada foi considerar o tempo de *setup* nos cálculos tanto de δ_j como no de X_j. Então, δ_j é calculado da seguinte forma:

$$\delta_j = \sum_{k=1}^{m} w_k \left(C_{[c]k} - C_{[i]k} - P_{jk}\right) \quad (130)$$

E X_j é calculado pela seguinte expressão:

$$X_j = \sum_{k=1}^{m} w_k \left(C_{[c+1]k} - C_{[c]k} - P_{vk}\right) \quad (131)$$

Note novamente as equações 130 e 131 são matematicamente iguais às equações 119 e 124, porém a interpretação de ambas é diferente.

Como cada adaptação fornece resultados diferentes uma da outra, elas foram nomeadas de *PW* e *PW*1. Ambos os métodos adaptados são comparados no capítulo 9.

8.2. Heurísticas de II Fase

Neste trabalho os modelos heurísticos de II fase adaptados são: *MME* e *PFE* propostos por Ronconi (2004); *wPFE*, *PWE*, *PF-NEH*(*x*), *wPF-NEH*(*x*), e *PW-NEH*(*x*), propostos por Pan e Wang (2012).

8.2.1. MME, PFE, wPFE e PWE

As heurísticas *MME*, *PFE*, *wPFE* e *PWE* são modificações da heurística *NEH* (Nawaz *et al.*, 1983). Enquanto a heurística *NEH* utiliza a regra de prioridade *LPT* para fornecer uma solução inicial, a heurística *MME* utiliza o *MM*, a heurística

PFE utiliza o *PF*, a heurística *wPFE* utiliza o *wPF* e a heurística *PWE* utiliza o *PW* para fornecer uma solução inicial.

As heurísticas inicialmente obtêm uma solução inicial usando uma das heurísticas de I fase. Em seguida define que a primeira tarefa da solução inicial obtida será a primeira tarefa da sequência π. As demais tarefas são sequenciadas por uma heurística de inserção, *i.e.* a *k*-ézima tarefa da solução inicial é testada em todas as *k* possíveis posições da sequência π, e é inserida na posição que resultar no menor valor da função objetivo do problema. Esse processo é realizado até que todas as tarefas estejam sequenciadas na sequência π. A seguir é descrito a heurística *NEH*.

Heurística *NEH*

Gerar uma sequência inicial $\beta = (\beta_1, \beta_2, \dots, \beta_n)$ usando a regra de prioridade *LPT*.
$\pi := \beta_1$.
for $k := 2$ **to** n **do**
Pegar a tarefa β_k de β e testar em todas as k possíveis posições de π.
Inserir a tarefa β_k na sequência π na posição que resulta no menor valor da função objetivo do problema.
endfor
return π

Todas as heurísticas de I fase adaptados foram utilizados para fornecer uma solução inicial para essas heurísticas de II fase. A função objetivo da heurística *NEH* para o problema é o *makespan*, calculado com as equações 1 a 6. As heurísticas adaptadas foram nomeadas de *MME*, *MM*1*E*, *PFE*, *PF*1*E*, *wPFE*, *wPF*1*E*, *PWE* e *PW*1*E*.

8.2.2. PF-NEH(x), wPF-NEH(x) e PW-NEH(x)

Semelhantemente ao *PFE*, *wPFE* e o *PWE*, essas heurísticas obtêm uma solução inicial e então melhora o resultado obtido. No entanto, a solução inicial é inicialmente obtida pela regra de prioridade *LPT*. Em seguida a primeira tarefa da solução inicial é definida como sendo a primeira tarefa da sequência β, e as demais

tarefas são sequenciadas utilizando as heurísticas *PF*, *wPF* ou *PW*. A sequência β é então usada como uma solução inicial para uma versão modificada da heurística *NEH*. A heurística NEH aplica a heurística de inserção apenas às última λ tarefas da sequência, gerando a sequência π^1.

Esse procedimento é repetido *x* vezes, cada vez definindo a primeira tarefa da sequência β como sendo a *x*-ézima tarefa da solução inicial fornecida pela regra de prioridade *LPT*. No fim são geradas x diferentes sequências π (π^1, π^2, π^3, …, π^x). A sequência π que resultar o menor valor da função objetivo é então o resultado final da heurística. A seguir é descrito a heurística *PF-NEH*(*x*).

Heurística *PF-NEH*(*x*)

Gerar uma sequência inicial $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ usando a regra de prioridade *LPT*.

for *l*:=1 **to** *x* **do** %(gerar *x* sequências)

Definir α_l como a primeira tarefa e gerar a sequência $\beta = (\beta_1, \beta_2, \dots, \beta_n)$ usando a heurística *PF*.

$\pi^l := (\beta_1, \beta_2, \dots, \beta_{n-\lambda})$.

for $k := n - \lambda + 1$ **to** n **do** %(a heurística *NEH*)

Pegar a tarefa β_k de β e testar em todas as k possíveis posições de π^l.

Inserir a tarefa β_k na sequência π^l na posição que resulta no menor valor da função objetivo do problema.

endfor

endfor

return a sequência $\pi \epsilon \{\pi^1, \pi^2, \dots, \pi^x\}$ com o menor valor do *makespan*.

Os demais métodos, *wPF-NEH*(*x*) e *PW-NEH*(*x*), funcionam de forma similar, porém a sequência β é obtida usando os métodos *wPF* e *PW*, respectivamente. Todos os métodos heurísticos de I fase foram utilizados para providenciar uma solução inicial para essas heurísticas de II fase. O objetivo da heurística *NEH* é a minimização do *makespan*, calculado pelas equações 1 a 6. As heurísticas adaptadas foram nomeadas de *PF-NEH*(*x*), *PF*1-*NEH*(*x*), *wPF-NEH*(*x*), *wPF*1-*NEH*(*x*), *PW-NEH*(*x*) e *PW*1-*NEH*(*x*).

8.3. Heurísticas de III Fase

Neste trabalho os modelos heurísticos de III fase adaptados são: *PF-NEH$_{ls}$*(*x*), *wPF-NEH$_{ls}$*(*x*), e *PW-NEH$_{ls}$*(*x*) propostos por Pan e Wang (2012).

As heurísticas de II fase *PF-NEH*(*x*), *wPF-NEH*(*x*), e *PW-NEH*(*x*) são utilizadas para prover uma solução inicial para as heurísticas *PF-NEH$_{ls}$*(*x*), *wPF-NEH$_{ls}$*(*x*), e *PW-NEH$_{ls}$*(*x*), respectivamente. Então uma busca local é usada como um procedimento de melhoria. A busca local utilizada é a busca local referenciada (*RLS*[12]) apresentada por Pan *et al.* (2008) e adaptada por Pan e Wang (2012). O procedimento *RLS* utiliza uma solução inicial gerada por uma das heurísticas de II fase e então busca iterativamente as soluções vizinhas até que um ótimo local é encontrado. O procedimento *RLS* é apresentado a seguir.

Procedimento *RLS* $(\boldsymbol{\pi}, \boldsymbol{\pi^{ref}})$
  do
    for i:=1 **to** n **do**
      $\pi' \coloneqq \pi$
      Encontrar a tarefa π_i^{ref} em π' e removê-la.
      Testar π_i^{ref} em todas as posições possíveis de π'.
      Inserir π_i^{ref} na posição de π' que resultar no menor valor do *makespan*.
      if $f(\pi') < f(\pi)$ **then** $\pi \coloneqq \pi'$ %($f(\pi)$ representa o valor do makespan de π)
      endfor
  while π é melhorado
return π

Todos os métodos heurísticos de II fase foram utilizados para providenciar uma solução inicial para essas heurísticas de III fase. As heurísticas adaptadas foram nomeadas de *PF-NEH$_{ls}$*(*x*), *PF*1-*NEH$_{ls}$*(*x*), *wPF-NEH$_{ls}$*(*x*), *wPF*1-*NEH$_{ls}$*(*x*), *PW-NEH$_{ls}$*(*x*) e *PW*1-*NEH$_{ls}$*(*x*).

---

[12] *Referenced Local Search*

No capítulo seguinte são apresentados os resultados obtidos pelos testes computacionais realizados, bem como as comparações dos métodos de solução.

9. RESULTADOS COMPUTACIONAIS

Cada método de solução é mais indicado para diferentes classes de problemas, devido às suas características de execução. Portanto, para cada método de solução apresentado (modelos *MILP*, método *Branch-and-Bound*, e métodos heurísticos construtivos) foram utilizados problemas diferentes, alterando quantidade de máquinas e de tarefas. Os resultados obtidos para cada método são apresentados a seguir.

9.1. Modelos MILP

Devido ao elevado tempo computacional, os modelos *MILP* são recomendados para problemas de pequeno e médio porte. Portanto, ao invés de utilizar a conhecida base de dados fornecida por Taillard (1993), que é composta de problemas relativamente grandes, uma nova base de dados foi gerada para a comparação dos modelos *MILP* apresentados, composta de 80 problemas, separados em oito classes que variam na quantidade de tarefas (*n*) e na quantidade de máquinas (*m*). As classes de problemas utilizados são:

$$(n, m) = \{(5,3); (10,3); (10,7); (10,10); (15,3); (15,7); (15,10); (20,3)\}$$

Cada classe de problemas possui 10 problemas. Os valores dos tempos de processamento foram uniformemente distribuídos entre 1 e 99 (como proposto por Taillard, 1993). Os valores dos tempos de *setup* também foram uniformemente distribuídos entre 1 e 99. Desta forma é possível que os tempos de *setup* sejam menores, iguais ou maiores que o tempo de processamento, sem uma discrepância muito grande entre seus valores. Valores de tempo de *setup* muito menores que o tempo de processamento podem gerar problemas onde não ocorram bloqueios, e valores de tempo de *setup* muito maiores que o tempo de processamento podem impor a ocorrência de muitos bloqueios no problema. Os modelos foram programados no *software* GAMS e solucionados utilizando o CPLEX 12. Os experimentos foram realizados com um processador Intel® core i7 3610QM com 2.3

GHz, 8 Gb DDR3 RAM e sistema operacional Windows 7. Foi estipulado um limite de tempo computacional de 3600 segundos para a solução de cada problema com cada modelo.

Para as comparações foram calculados as médias do tempo computacional (tempo CPU), do número de iterações no Simplex (*Simplex it*) e do número de nós explorados no *Branch-and-Bound* (*B&B nós*). Na Tabela 7 é apresentada a comparação do tempo computacional dos modelos *MILP*, na Tabela 8 é apresentada a comparação do número de iterações *Simplex* utilizadas por cada método para solucionar os problemas, e na Tabela 9 são comparados os números de nós explorados no *Branch-and-Bound* de cada modelo *MILP*. Cada célula das tabelas 7, 8 e 9 representa a média de um conjunto de 10 problemas. Nas tabelas foram calculadas as médias totais de cada modelo para cada um dos parâmetros comparados e também as médias considerando apenas os modelos onde "tempo CPU" < 3600 segundos.

Tabela 7 – Comparação do tempo computacional dos modelos *MILP*

Tamanho		Tempo CPU (segundos)			
n	*m*	TNZBS1	TNZBS2	RBZBS1	RBZBS2
5	3	0,146	0,169	**0,145**	0,151
10	3	213,1	221,0	**196,1**	209,5
10	7	477,7	553,3	**452,7**	548,0
10	10	453,3	421,1	**379,5**	477,7
15	3	**3600**	**3600**	**3600**	**3600**
15	7	**3600**	**3600**	**3600**	**3600**
15	10	**3600**	**3600**	**3600**	**3600**
20	3	**3600**	**3600**	**3600**	**3600**
Média total		1943,03	1949,45	**1928,56**	1954,42
Média ("tempo CPU" < 3600)		286,07	298,90	**257,11**	308,83

Tabela 8 – Comparação do número de iterações Simplex dos modelos *MILP*

Tamanho		Simplex it			
n	*m*	TNZBS1	TNZBS2	RBZBS1	RBZBS2
5	3	**1245,9**	1448,1	1112,0	1373,3
10	3	**4085221,8**	4136571,1	4161631,2	4089335,0
10	7	**7799378,5**	8893589,7	8252105,5	8692875,5
10	10	6691378,6	**5874142,0**	6218593,4	6722366,7
15	3	25935828,6	**25923635,6**	27008938,3	26541075,1
15	7	19538952,4	**18918844,4**	21884126,6	19362250,3
15	10	17142515,0	16296682,0	18918547,7	**15973099,9**
20	3	9093888,7	**8782921,6**	9040474,9	8992677,9
Média total		11286051,2	**11103479,3**	11935691,2	11296881,7
Média ("tempo CPU" < 3600)		**4644306,2**	4726437,7	4658360,5	4876487,6

Tabela 9 – Comparação do número de nós explorados no *Branch-and-Bound* dos modelos *MILP*

Tamanho		B&B nós			
n	*m*	TNZBS1	TNZBS2	RBZBS1	RBZBS2
5	3	55,6	62,4	**48,8**	56,1
10	3	129204,8	128825,0	120004,7	**110815,9**
10	7	167662,4	182453,1	172612,9	**167288,6**
10	10	116056,4	**97921,6**	104894,9	103742,2
15	3	279524,1	290851,1	294175,4	**160068,1**
15	7	135841,1	132454,4	146117,8	**61937,9**
15	10	102530,2	94324,1	102600,2	**52910,7**
20	3	47750,3	36449,4	35882,1	**17565,5**
Média total		122328,1	120417,6	122042,1	**84298,1**
Média (“tempo CPU” < 3600)		103244,8	102315,5	99390,3	**95475,7**

Como o tempo computacional foi limitado, alguns modelos não foram capazes de obter o resultado ótimo para alguns dos problemas. Por isso, foi calculado também o desvio relativo médio (*DRM*) dos resultados obtidos (*makespan*) mostrados na Tabela 10. O desvio relativo médio do *makespan* indica a qualidade dos resultados obtidos, quanto menor o valor melhor, e é calculado para cada classe de problemas pela média dos valores obtidos de *DR* (que é obtido individualmente para cada problema pela equação 41) e também foram calculadas as médias considerando apenas os modelos onde “tempo CPU” >= 3600 segundos.

Tabela 10 – Comparação do desvio relativo médio do *makespan* dos modelos *MILP*

Tamanho		Desvio Relativo Médio do *makespan* (%)			
n	*m*	TNZBS1	TNZBS2	RBZBS1	RBZBS2
5	3	**0%**	**0%**	**0%**	**0%**
10	3	**0%**	**0%**	**0%**	**0%**
10	7	**0%**	**0%**	**0%**	**0%**
10	10	**0%**	**0%**	**0%**	**0%**
15	3	0,9370%	**0,8897%**	1,3542%	1,2041%
15	7	**0,5831%**	1,1842%	1,4246%	1,8271%
15	10	0,8086%	**0,6929%**	1,3649%	1,2882%
20	3	**1,3060%**	2,2139%	1,4629%	2,2029%
Média total		**0,4543%**	0,6226%	0,7008%	0,8153%
Média (“tempo CPU” >= 3600)		**0,9087%**	1,2452%	1,4016%	1,6306%

Na Tabela 11 são apresentados os resultados do *makespan* para os problema com 15 tarefas e 3 máquinas e problemas com 15 tarefas e 10 máquinas. Em ambos os casos o menor *DRM* do *makespan* foi obtido pelo modelo *TNZBS*2, no entanto na maioria dos problemas nessas classes de problemas outros modelos obtiveram melhores resultados do *makespan*.

Tabela 11 – Comparação do valo do *makespan* dos modelos *MILP* para as classes de problemas 15x3 e 15x10

Tamanho		Problema	*Makespan*			
n	m	n°	TNZBS1	TNZBS2	RBZBS1	RBZBS2
15	3	1	1528	1524	1576	**1523**
		2	1576	1594	**1542**	1601
		3	**1531**	1535	1552	1543
		4	1602	**1591**	1622	1630
		5	**1568**	1587	1587	1608
		6	1458	**1429**	1432	1461
		7	1543	1553	1549	**1531**
		8	1629	1630	1661	**1622**
		9	**1586**	1591	**1586**	1589
		10	1419	1403	1403	**1379**
15	10	1	2264	2250	2320	**2234**
		2	2285	**2225**	2247	2274
		3	2259	2253	**2245**	2277
		4	2234	**2180**	2212	2230
		5	**2259**	2275	2286	2308
		6	**2168**	2188	2207	2180
		7	**2212**	2230	2266	2279
		8	**2297**	2309	2298	2311
		9	**2168**	2216	2184	2181
		10	2242	2236	2246	**2221**

Nas tabelas é possível observar que os problemas com 15 tarefas ou mais não foram possíveis de serem resolvidos dentro do limite de tempo computacional estipulado. O número de variáveis binárias de todos os modelos é igual, havendo apenas variações no número de variáveis contínuas e no número de restrições. Pela Tabela 7 é possível ainda verificar certa similaridade nos modelos, sendo que o modelo RBZBS1 obteve os resultados um pouco mais rápido que os demais nos problemas menores. Porém, nos problemas maiores, onde o tempo computacional excedeu os 3600 segundos, é possível analisar pela Tabela 10 que o modelo TNZBS1 conseguiu alcançar em média resultados um pouco melhores do que os demais modelos, observado pelo desvio relativo médio do makespan das classes de problema onde o tempo computacional para resolver o problema foi maior que 3600 segundos (última linha da Tabela 10).

Nos problema com 15 tarefas e 3 máquinas o modelo que obteve os melhores valores de *makespan* na maioria das vezes foi o modelo *RBZBS*2; e nos problemas com 15 tarefas e 10 máquinas o modelo que obteve os melhores valores de *makespan* na maioria dos casos foi o modelo *TNZBS*1. No entanto em ambos os casos o modelo *TNZBS*2 obteve o menor valor do desvio relativo médio do *makespan*. Isso se deve ao fato do valor do *makespan* obtido pelo modelo *TNZBS*2 nunca ter sido muito maior do que o valor obtido pelo melhor modelo em cada um dos problemas.

Todos os modelos têm o mesmo número de variáveis binárias, porém o modelo RBZBS2 tem o menor número de variáveis contínuas (em média 80% menor que o modelo TNZBS2, que é o modelo com maior número de variáveis contínuas, para as classes de problemas propostos). O modelo RBZBS1 tem o menor número de restrições (em média 17% menor que o modelo TNZBS2, que é o modelo com maior número de restrições, para as classes de problema propostos). No entanto, o número de variáveis e o número de restrições não influenciou o tempo computacional dos modelos, visto que o modelo com maior número de restrições e de variáveis (TNZBS2) obteve resultados melhores do que o modelo com o menor número de variáveis e o segundo menor número de restrições (RBZBS2). Pelas tabelas 7, 8 e 9 pode ser notado que o número de iterações Simplex e o número de nós explorados no *Branch-and-Bound* para resolver os problemas também parecem não ter influenciado no tempo computacional, visto que o modelo com o maior número de iterações Simplex e o segundo maior número de nós explorados no *Branch-and-Bound* (RBZBS1) é o modelo que obteve os resultados no menor tempo computacional.

Pelos resultados pode-se observar ainda que o modelo adaptado RBZBS1 obteve os menores tempos computacionais para os problemas menores, porém, para os problemas maiores, o modelo proposto TNZBS1 obteve os melhores resultados (observado pelo desvio relativo médio do *makespan* na Tabela 10) dentro do limite de tempo computacional. O número médio de iterações simplex é menor no modelo TNZBS1 que no modelo RBZBS1 para todos os problemas. Ainda, o número médio de nós explorados na árvore de *Branch-and-Bound* é menor no modelo TNZBS1 que no modelo RBZBS1 para os problemas onde o tempo computacional excede os 3600 segundos. Tudo isso combinado pode sugerir que o modelo TNZBS1 consegue convergir em resultados melhores mais rapidamente para problemas maiores.

9.2. Método Branch-and-Bound

O algoritmo *B&B* foi aplicado a 540 problemas. Foram utilizados os dados de tempo de processamento propostos por RONCONI (2005). Os problemas variam em quantidade de tarefas e de máquinas e são divididos em 27 classes, cada uma com

20 problemas. Nesta base de dados o tempo de processamento variam uniformemente entre 1 e 99. A base de dados fornecida por RONCONI (2005) não inclui tempo de *setup*, portanto os tempos de *setup* para cada par de tarefas em cada máquina foram gerados usando o mesmo método, ou seja, novamente os valores dos tempos de *setup* foram uniformemente distribuídos entre 1 e 99. Todos os limitantes inferiores e o algoritmo *Branch-and-Bound* foram programados em Matlab. Os experimentos foram realizados com um processador Intel® core i7 3610QM com 2.3 GHz, 8 Gb DDR3 RAM e sistema operacional Windows 7. A solução inicial para os problemas foi calculada utilizando o algoritmo *NEH* (NAWAZ *et al.*, 1983) adaptado para o problema. O tempo computacional para cada problema foi limitado para 3600 segundos.

Para comparação foram calculados, para cada classe de problemas, a média do tempo computacional em segundos e a média do número de nós ramificados. Na Tabela 12 são apresentados os resultados obtidos, apenas dos problemas que foram resolvidos dentro do tempo computacional estabelecido, utilizando os quatro limitantes inferiores.

Tabela 12 – Desempenho dos algoritmos *Branch-and-Bound*.

Tamanho		$B\&B_{TN1}$		$B\&B_{TN2}$		$B\&B_{TN3}$		$B\&B_{TN4}$	
n	m	Tempo CPU (seg)	Número de nós	Tempo CPU (seg)	Número de nós	Tempo CPU (seg)	Número de nós	Tempo CPU (seg)	Número de nós
10	2	6,48	**41633,85**	**5,91**	**41633,85**	17,25	50223,30	6,49	**41633,85**
10	3	22,51	**104918,90**	**20,00**	**104918,90**	76,23	127231,35	23,47	109484,15
10	4	52,24	194619,30	**45,29**	**193334,70**	202,77	235896,20	58,66	218670,85
10	5	64,81	202104,35	**54,93**	**199743,05**	265,87	249201,45	74,72	232867,15
10	7	127,87	302099,05	**105,80**	**295858,65**	574,30	387785,70	161,24	380684,20
10	10	164,93	284526,60	**129,95**	**270314,60**	691,52	330139,70	198,64	344718,40
12	2	91,21	**572712,55**	**83,32**	**572712,55**	229,24	659327,55	91,11	**572712,55**
12	3	423,18	**1903380,06**	**376,59**	**1903380,06**	1439,92	2345772,67	456,63	2056967,72
12	4	727,54	2648836,14	**629,66**	**2618464,14**	2800,96	3205695,29	898,19	3248940,57
12	5	785,60	2389311,00	**660,51**	**2337803,00**	2793,43	2568676,33	870,45	2651519,67
14	2	594,16	**3890057,80**	**542,23**	**3890057,80**	1559,40	4683547,90	594,78	**3890057,80**
Média		278,23	1139472,69	**241,29**	**1129838,30**	968,26	1349408,86	312,22	1249841,54

* Média do tempo computacional e do número de nós dos problemas que foram resolvidos dentro do tempo computacional de 3600 segundos utilizando todos os limitantes inferiores

Nas demais classes de problemas um ou mais limitantes inferiores não foram capazes de resolver nenhuma tarefa dentro do tempo computacional de 3600 segundos, portanto não foram calculadas as médias de tempo computacional e de número de nós.

Após 3600 segundos o programa era interrompido e o resultado obtido até o momento era computado. Portanto, para comparação dos métodos, foram

computados também o desvio relativo médio do *makespan* e o número de problemas não resolvidos para cada classe de problemas. Na Tabela 13 são apresentados os desvios relativos médios do *makespan* e o número de problemas não resolvidos obtidos para cada classe de problemas utilizando cada um dos quatro limitantes inferiores para o problema.

Tabela 13 – Desvio relativo do *makespan* e número de problemas não resolvidos pelos algoritmos *Branch-and-Bound*.

Tamanho		$B\&B_{TN1}$		$B\&B_{TN2}$		$B\&B_{TN3}$		$B\&B_{TN4}$	
n	m	Desvio relativo médio do *makespan*	Número de problemas não resolvidos	Desvio relativo médio do *makespan*	Número de problemas não resolvidos	Desvio relativo médio do *makespan*	Número de problemas não resolvidos	Desvio relativo médio do *makespan*	Número de problemas não resolvidos
10	2	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**
10	3	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**
10	4	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**
10	5	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**
10	7	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**
10	10	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**
12	2	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	**0**
12	3	**0,000%**	**0**	**0,000%**	**0**	**0,000%**	2	**0,000%**	**0**
12	4	**0,000%**	**0**	**0,000%**	**0**	0,192%	13	**0,000%**	1
12	5	**0,000%**	4	**0,000%**	**1**	0,192%	17	**0,000%**	5
12	7	**0,000%**	8	**0,000%**	**4**	0,958%	20	**0,000%**	14
12	10	**0,003%**	17	**0,003%**	**15**	0,875%	20	0,395%	17
14	2	**0,032%**	5	**0,032%**	**4**	0,160%	10	**0,032%**	5
14	3	**0,004%**	18	**0,004%**	**16**	0,551%	20	**0,004%**	18
14	4	0,415%	**20**	**0,242%**	**20**	0,978%	**20**	0,444%	**20**
14	5	0,859%	**20**	**0,735%**	**20**	1,628%	**20**	0,855%	**20**
14	7	0,255%	**20**	**0,191%**	**20**	0,998%	**20**	1,009%	**20**
14	10	0,790%	**20**	**0,542%**	**20**	1,376%	**20**	1,276%	**20**
16	2	0,382%	**16**	**0,220%**	**16**	0,760%	20	0,382%	**16**
16	3	0,777%	**20**	**0,773%**	**20**	1,111%	20	0,921%	**20**
16	4	0,815%	**20**	**0,729%**	**20**	1,761%	20	1,470%	**20**
18	2	**0,721%**	**20**	**0,721%**	**20**	1,036%	20	**0,721%**	**20**
18	3	0,484%	**20**	0,484%	**20**	1,023%	20	**0,272%**	**20**
18	4	1,153%	**20**	1,090%	**20**	1,372%	20	**1,061%**	**20**
20	2	**0,378%**	**20**	**0,378%**	**20**	1,196%	20	**0,378%**	**20**
20	3	1,194%	**20**	1,167%	**20**	1,325%	20	**0,805%**	**20**
20	4	1,113%	**20**	**1,095%**	**20**	1,284%	20	1,361%	**20**
Total		9,375%	308	**8,406%**	**296**	18,776%	362	11,386%	316
Média		0,347%	11,407	**0,311%**	**10,963**	0,695%	13,407	0,422%	11,704

Cada classe de problemas possui 20 problemas diferentes, totalizando 540 problemas. 20 problemas não resolvidos em uma classe de problemas significa que o algoritmo de *Branch-and-Bound* não foi capaz de encontrar a solução dentro do tempo computacional estipulado para nenhum problema dentro daquela classe de problemas.

Os experimentos numéricos indicam que o melhor limitante inferior foi o LB_{TN2}, seguido pelo LB_{TN1}, LB_{TN4} e por último pelo LB_{TN3}. Em 100% dos problemas estudados o limitante inferior LB_{TN2} possibilitou que o *Branch-and-Bound* chegasse ao resultado ótimo para o problema com um número menor de ramificações que os demais, resultando também em um tempo computacional médio 75,08% menor

quando comparado ao limitante inferior menos efetivo (LB_{TN3}). O limitante inferior LB_{TN2} foi também o que conseguiu resolver mais problemas dentro do limite de tempo computacional proposto dentre os limitantes inferiores propostos (18,23% menos problemas não resolvidos que o limitante inferior LB_{TN3}), e nos problemas cujo resultado ótimo não foi alcançado dentro do tempo computacional de 3600 segundos, foi o limitante que obteve os melhores resultados (observados pela média do desvio relativo).

9.3. Comparação Modelo MILP e *Branch-and-Bound*

O algoritmo B&B utilizando o limitante inferior LB_{TN2}, que obteve os melhores resultados dentre todos os limitantes inferiores, foi comparado ao modelo TNZBS1, que obteve os melhores resultados dentre os modelos *MILP*. Para a comparação foi utilizada a mesma base de dados utilizada para os testes computacionais dos modelos *MILP*. Ou seja, foram utilizados 80 diferentes problemas, divididos em oito diferentes classes de problemas. Cada classe variando em número de tarefas e de máquinas, com 10 problemas diferentes cada uma.

O algoritmo *MILP* foi programado no software GAMS e resolvido utilizando o CPLEX 12. O algoritmo *Branch-and-Bound* bem como o limitante inferior LB_{TN2} foram programados em Matlab. Os experimentos foram realizados com um processador Intel® core i7 3610QM com 2.3 GHz, 8 Gb DDR3 RAM e sistema operacional Windows 7. Foi estipulado também um limite para o tempo computacional de 3600 segundos para resolver cada problema com cada um dos modelos.

Para comparação foi calculado o tempo computacional médio (tempo CPU) em segundos. Como o tempo computacional foi limitado para 3600 segundos, alguns modelos não foram capazes de encontrar a solução ótima para alguns dos problemas. Portanto, foi também calculado o desvio relativo médio dos resultados obtidos (*makespan*) e o número de problemas não resolvidos para cada classe de problemas. Na Tabela 14 são apresentados os resultados obtidos para as classes de problemas.

Tabela 14 – Tempo computacional e desvio relativo médio do *makespan* e número de problemas não resolvidos pelo algoritmo *Branch-and-Bound* e pelo modelo *MILP*.

Tamanho		Tempo CPU (segundos)		Número de problemas não resolvidos		Desvio relativo médio do *makespan*	
n	*m*	B&B	*MILP*	B&B	*MILP*	B&B	*MILP*
5	3	**0,054609**	0,1455	**0**	**0**	**0,0000%**	**0,0000%**
10	3	**61,31284**	213,107	**0**	**0**	**0,0000%**	**0,0000%**
10	7	**249,5021**	477,674	**0**	**0**	**0,0000%**	**0,0000%**
10	10	**271,4882**	453,3423	**0**	**0**	**0,0000%**	**0,0000%**
15	3	**3600**	**3600**	**10**	**10**	**0,3514%**	0,8025%
15	7	**3600**	**3600**	**10**	**10**	0,9361%	**0,6603%**
15	10	**3600**	**3600**	**10**	**10**	1,5787%	**0,3962%**
20	3	**3600**	**3600**	**10**	**10**	**0,0000%**	2,9399%
Total		**14982,37**	15544,41	**40**	**40**	**2,8661%**	4,7989%
Média		**1872,796**	1943,052	**5**	**5**	**0,3583%**	0,5999%

Cada classe de problemas possui 10 problemas diferentes, em um total de 80 problemas. 10 problemas não resolvidos em uma classe de problemas significa que o método não foi capaz de encontrar a solução ótima dentro do limite de tempo computacional estipulado para nenhum dos problemas daquela classe.

O número de problemas não resolvidos foi o mesmo para ambos os métodos, no entanto, o tempo computacional médio necessário para o algoritmo *Branch-and-Bound* resolver os problemas foi 3,62% menor (ou 50,89% menor se apenas os problemas nos quais o tempo computacional foi menor que 3600 segundos) que o tempo computacional médio necessário para o modelo *MILP* resolver os problemas. Também, o algoritmo *Branch-and-Bound* obteve melhores resultados nos problemas que não foram resolvidos dentro do limite estipulado do tempo computacional (3600 segundos) (em média 40,28% menor que os resultados obtidos pelo modelo *MILP*). Esses resultados mostram a consistência do limitante inferior proposto para o algoritmo *Branch-and-Bound*.

Também é importante notar pelos dados da Tabela 14 que o desempenho do modelo *MILP* em relação ao algoritmo *Branch-and-Bound* tende a melhorar à medida que o número de máquinas aumenta. Por outro lado o algoritmo Branch-and-Bound tende a melhorar seu desempenho em relação ao modelo *MILP* à medida que o número de tarefas aumenta.

9.4. Métodos Heurísticos

Todas as 28 heurísticas foram usadas para resolver os 120 problemas propostos por Taillard (1993). Esses problemas variam em número de tarefas e máquinas, com 20, 50, 100, 200 e 500 tarefas e 5, 10 e 20 máquinas. Os tempos de *setup* foram gerados distribuindo uniformemente os valores entre 1 e γ, onde γ varia entre 10%, 50%, 100% e 125% do valor máximo do tempo de processamento (*i.e.* os tempos de *setup* foram uniformemente distribuídos entre 1 e 10 para a primeira base de dados dos tempos de *setup*, entre 1 e 50 para a segunda base de dados dos tempos de *setup*, entre 1 e 99 para a terceira base de dados dos tempos de *setup* e entre 1 e 125 para a quarta base de dados dos tempos de *setup*) Portanto cada um dos 120 problemas foi resolvido quatro vezes, uma vez para cada base de dados dos tempos de *setup*, totalizando 480 problemas resolvidos usando cada um das 28 heurísticas.

Todos os métodos heurísticos foram programados em Matlab. Os experimentos foram realizados em um Intel® core i7 3610QM with 2.3 GHz, 8 Gb DDR3 RAM e sistema operacional Windows 7.

O valor de *x* das heurísticas *PF-NEH*(*x*), *wPF-NEH*(*x*), *PW-NEH*(*x*), *PF*1-*NEH*(*x*), *wPF*1-*NEH*(*x*), *PW*1-*NEH*(*x*), *PF-NEH*$_{ls}$(*x*), *wPF-NEH*$_{ls}$(*x*), *PW-NEH*$_{ls}$(*x*), *PF*1-*NEH*$_{ls}$(*x*), *wPF*1-*NEH*$_{ls}$(*x*), and *PW*1-*NEH*$_{ls}$(*x*) foi definido como sendo igual a 5, pois este é o valor que, segundo Pan e Wang (2012), fornece os melhores resultados para as heurísticas.

Para comparação foram calculados, para cada classe de problemas, a média do tempo computacional em segundos (tempo *CPU*) e o desvio relativo médio do *makespan* (*DRM makespan*).

Quanto menor o valor do *DRM* do *makespan* melhores foram os resultados obtidos pela heurística. Quando o *DRM* do *makespan* for igual a zero por cento, significa que aquela heurística superou todas as demais heurísticas em todos os 120 problemas para aquele tempo de *setup*.

Como o número de resultados obtidos é muito grande, os problemas foram classificados de acordo com o tempo de *setup* (10%, 50%, 100% e 125%) e apenas os valores das médias dos resultados de todos os 120 problemas para cada tempo de *setup* é apresentado. Os resultados são então apresentados divididos em três

classes, de acordo com o método de classificação apresentado por Framinan *et al.* (2004). Primeiro são apresentados os resultados para as heurísticas de I fase (Tabela 15), então os resultados para as heurísticas de II fase (Tabela 16) e, por fim, os resultados para as heurísticas de III fase (Tabela 17).

Tabela 15 – Performance das heurísticas de I fase.

	Tempo de *setup*	10%	50%	100%	125%	Média
MM	Tempo *CPU* (segundos)	0,085926	0,102239	0,084343	0,085347	0,089464
	DRM Makespan	6,84%	10,96%	17,93%	20,53%	14,06%
*MM*1	Tempo *CPU* (segundos)	0,261275	0,083035	0,263884	0,263608	0,217951
	DRM Makespan	6,65%	8,30%	11,10%	12,29%	9,59%
PF	Tempo *CPU* (segundos)	**0,031913**	0,040141	**0,032391**	**0,032739**	0,034296
	DRM Makespan	2,63%	4,01%	6,82%	8,37%	5,46%
*PF*1	Tempo *CPU* (segundos)	0,034416	0,014790	0,034861	0,034478	**0,029637**
	DRM Makespan	2,55%	1,48%	**1,00%**	**0,94%**	1,49%
wPF	Tempo *CPU* (segundos)	0,036179	0,039845	0,036542	0,036393	0,037240
	DRM Makespan	2,16%	3,98%	7,12%	8,12%	5,35%
*wPF*1	Tempo *CPU* (segundos)	0,038678	0,018807	0,037945	0,038334	0,033441
	DRM Makespan	2,14%	1,88%	1,99%	2,21%	2,06%
PW	Tempo *CPU* (segundos)	0,230951	0,029480	0,232365	0,231828	0,181156
	DRM Makespan	**0,90%**	2,95%	6,09%	7,39%	4,33%
PW1	Tempo *CPU* (segundos)	0,234252	**0,008914**	0,247537	0,247926	0,184657
	DRM Makespan	0,94%	**0,89%**	1,37%	1,75%	**1,24%**

Tabela 16 – Performance das heurísticas de II fase.

	Tempo de *setup*	10%	50%	100%	125%	Média
MME	Tempo *CPU* (segundos)	30,5200	30,5332	30,5791	30,3651	30,4993
	DRM Makespan	2,89%	2,77%	3,98%	4,63%	3,57%
*MM*1*E*	Tempo *CPU* (segundos)	30,6779	30,6076	30,4606	30,4445	30,5477
	DRM Makespan	2,70%	2,71%	3,67%	4,28%	3,34%
PFE	Tempo *CPU* (segundos)	29,2710	29,2024	29,2053	29,2563	29,2338
	DRM Makespan	3,06%	2,92%	3,83%	4,30%	3,53%
*PF*1*E*	Tempo *CPU* (segundos)	29,3250	29,3219	29,3437	29,3382	29,3322
	DRM Makespan	3,07%	2,72%	3,25%	3,46%	3,13%
wPFE	Tempo *CPU* (segundos)	28,7882	28,8043	28,9820	29,0061	28,8951
	DRM Makespan	3,16%	3,09%	3,85%	4,39%	3,62%
*wPF*1*E*	Tempo *CPU* (segundos)	30,0133	30,0524	30,0751	30,0937	30,0586
	DRM Makespan	3,33%	2,88%	3,42%	3,85%	3,37%
PWE	Tempo *CPU* (segundos)	29,1242	29,1717	29,0980	29,2106	29,1511
	DRM Makespan	3,20%	3,02%	3,88%	4,30%	3,60%
*PW*1*E*	Tempo *CPU* (segundos)	29,2552	29,2912	29,3188	29,3345	29,2999
	DRM Makespan	3,01%	2,66%	3,21%	3,75%	3,16%
PF-NEH (*x*)	Tempo *CPU* (segundos)	**23,9406**	**23,9242**	**23,9518**	**23,9126**	**23,9323**
	DRM Makespan	0,98%	2,01%	3,78%	4,51%	2,82%
*PF*1*-NEH* (*x*)	Tempo *CPU* (segundos)	24,9239	24,8806	24,9687	24,9623	24,9339
	DRM Makespan	0,91%	**0,64%**	**0,57%**	**0,42%**	**0,64%**
wPF-NEH (*x*)	Tempo *CPU* (segundos)	24,7401	24,7103	24,7489	24,7294	24,7322
	DRM Makespan	0,72%	1,95%	3,68%	4,58%	2,74%
*wPF*1*-NEH* (*x*)	Tempo *CPU* (segundos)	24,8432	24,8028	24,7778	24,7815	24,8013
	DRM Makespan	0,78%	0,75%	0,93%	1,01%	0,87%
PW-NEH (*x*)	Tempo *CPU* (segundos)	25,7594	25,8008	25,7432	25,7574	25,7652
	DRM Makespan	0,72%	1,89%	3,68%	4,45%	2,68%
*PW*1*-NEH* (*x*)	Tempo *CPU* (segundos)	25,9460	25,8599	25,8398	25,8390	25,8712
	DRM Makespan	**0,68%**	0,69%	0,83%	1,08%	0,82%

Tabela 17 – Performance das heurísticas de III fase.

Tempo de *setup*		10%	50%	100%	125%	Média
PF-NEH$_{ls}$ (*x*)	Tempo *CPU* (segundos)	2295,53358	2746,729	3012,349	3336,181	2847,698
	DRM Makespan	0,44%	1,19%	2,46%	2,90%	1,75%
*PF*1-*NEH*$_{ls}$ (*x*)	Tempo *CPU* (segundos)	2206,90025	**2073,815**	**1705,959**	**1586,225**	**1893,225**
	DRM Makespan	**0,38%**	**0,36%**	**0,42%**	**0,35%**	**0,38%**
wPF-NEH$_{ls}$ (*x*)	Tempo *CPU* (segundos)	**2112,2732**	2520,396	3144,436	3373,013	2787,529
	DRM Makespan	0,57%	1,41%	2,43%	3,01%	1,85%
*wPF*1-*NEH*$_{ls}$ (*x*)	Tempo *CPU* (segundos)	2147,7254	2133,1	2073,578	2163,711	2129,528
	DRM Makespan	0,52%	0,64%	0,72%	1,02%	0,73%
PW-NEH$_{ls}$ (*x*)	Tempo *CPU* (segundos)	2239,56082	2619,275	3406,103	3419,177	2921,029
	DRM Makespan	0,51%	1,36%	2,52%	2,91%	1,83%
*PW*1-*NEH*$_{ls}$ (*x*)	Tempo *CPU* (segundos)	2280,20831	2159,175	2096,921	1989,555	2131,465
	DRM Makespan	0,58%	0,63%	0,73%	1,00%	0,73%

É possível verificar pelas tabelas 15, 16 e 17 que todos os métodos adaptados que consideraram o tempo de *setup* (os métodos cujos nomes são seguidos pelo número 1) superaram os métodos que não levaram em conta o tempo de *setup* nos cálculos. Esses resultados indicam que considerar o tempo de *setup*, e não apenas os tempos de ociosidade e de bloqueio, nos cálculos é relevante, pois ele é também uma variante do problema.

Como pode ser visto na Tabela 15, a heurística *PW*1 obteve o melhor desvio relativo médio do *makespan* entre todas as heurísticas de I fase, com uma média total do *DRM* do *makespan* igual a 1,24% contra 1,49% obtido pela segunda melhor heurística de I fase, o *PF*1. No entanto a heurística *PF*1 obteve a melhor média do tempo computacional, com uma média total de 0,029637 segundos contra 0,184657 segundos obtida pela heurística *PW*1. A heurística *PW*1 obteve uma média de 0,25% melhores resultados no *DRM* do *makespan*; porém a heurística *PF*1 foi em média 83,851% mais rápido para encontrar uma solução para os problemas.

*PF*1-*NEH* (*x*) e *PF-NEH*(*x*) obtiveram os melhores *DRM* do *makespan* e o menor tempo computacional entre todas as heurísticas de II fase, respectivamente (observados pela Tabela 16). *PF*1-*NEH*$_{ls}$(*x*) obteve o melhor *DRM* do *makespan* e o menor tempo computacional entre todas as heurísticas de III fase, observado pela Tabela 17.

Os resultados apresentados por Pan e Wand (2012) mostram que, dentre as heurísticas de I fase, para o problema *flow shop* permutacional com bloqueio com *buffer* zero, a heurística *PW* obteve os melhores resultados em relação ao *DRM* do *makespan*, e a heurística *MM* obteve os menores tempos computacionais. O melhor *DRM* do *makespan* dentre as heurísticas de II fase foi obtido pela heurística *PW-NEH*(5), e dentre as heurísticas de III fase foi obtido pela heurística *PF-NEH*$_{ls}$(5). E o

menor tempo computacional dentre as heurísticas de II e III fase foi obtido pelas heurísticas *MME* e *wPF-NEH*$_{ls}$(5), respectivamente. Pelos resultados apresentados por Pan e Wang (2012) pode-se notar que a heurística *PW* fornece resultados um pouco melhores para o problema *flow shop* permutacional com bloqueio com *buffer* zero. No entanto, pelos resultados obtidos neste trabalho, *PF*1, combinado ou não com outros métodos (*NEH* ou *RLS*) pode ser considerado a heurística com os melhores resultados para o problema. Possivelmente a variação do tempo de *setup* dependente da sequência em cada máquina torna difícil prever os efeitos das tarefas restantes nos tempos de ociosidade e de bloqueio. Isso pode ser verificado pelo aumento na diferença entre os resultados das heurísticas baseadas na heurística *PF*1 e *PW*1 quando a faixa de valores do tempo de *setup* aumenta.

Para comparar as heurísticas, na Tabela 18 é apresentada uma comparação entre as heurísticas que obtiveram os melhores resultados em cada classe (I, II e III fases).

Tabela 18 – Comparação entre as melhores heurísticas de I, II e III fase.

Tempo de *setup*		10%	50%	100%	125%	Média
*PF*1	Tempo *CPU* (segundos)	**0,0344**	**0,0353**	**0,0349**	**0,0345**	**0,0348**
	DRM Makespan	6,7509%	5,1572%	4,0114%	4,0696%	4,9973%
*PW*1	Tempo *CPU* (segundos)	0,2342	0,2437	0,2475	0,2479	0,2434
	DRM Makespan	5,0597%	4,5275%	4,3845%	4,8923%	4,7160%
*PF*1-*NEH* (*x*)	Tempo *CPU* (segundos)	24,9238	24,8806	24,9687	24,9623	24,9339
	DRM Makespan	1,8190%	1,4506%	1,2360%	1,1658%	1,4178%
*PF*1-*NEH*$_{ls}$ (*x*)	Tempo *CPU* (segundos)	2206,9002	2073,8152	1705,9592	1586,2246	1893,2250
	DRM Makespan	**0,0012%**	**0,0000%**	**0,0000%**	**0,0000%**	**0,0003%**

Como esperado a heurística *PF*1 obteve os menores tempos computacionais, por ser uma heurística de I fase, e *PF*1-*NEH*$_{ls}$(*x*) obteve os melhores *DRM* do *makespan*, por ser uma heurística de III fase. A heurística *PF*1-*NEH*$_{ls}$(*x*) obteve *DRM*s do *makespan* que foram em média 4,9970%, 4,7157% e 1,4175% melhores do que os obtidos pelas heurísticas *PF*1, *PW*1 e *PF*1-*NEH*(*x*), respectivamente. A heurística *PF*1 resolveu os problema em média 85,7179%, 99,8606% e 99,9982% mais rápido que as heurísticas *PW*1, *PF*1-*NEH*(*x*) e *PF*1-*NEH*$_{ls}$(*x*), respectivamente.

A heurística *PF*1-*NEH*(*x*) apresentou uma boa relação de tempo de desempenho, apesar de a heurística *PF*1-*NEH*$_{ls}$(*x*) obteve um DRM do makespan em média 1,4175% melhor que a heurística *PF*1-*NEH*(*x*), esta obteve os resultados em média 98,6830% mais rápido que a heurística *PF*1-*NEH*$_{ls}$(*x*).

10. CONCLUSÕES

Este trabalho considerou um problema *flow shop* permutacional com bloqueio e *setup* dependente da sequência e da máquina. Muitos trabalhos consideram o problema com estoque intermediário ilimitado (Ignall; Schrage, 1965; Nawaz *et al.*, 1983) ou considerando o *setup* embutido ao tempo de processamento da tarefa (Hall; Sriskandarajah, 1996; Leisten, 1990; McCormick *et al*., 1989; Pan; Wang, 2012; Papadimitriou; Kanellakis, 1980; Ronconi, 2004; Ronconi; Armentano, 2001; Ronconi, 2005; Ronconi; Birgin, 2012). O tempo de *setup* da máquina separado do tempo de processamento permite uma flexibilidade maior para o sequenciamento da produção, podendo resultar em um maior aproveitamento de tempo, consequentemente numa redução do *makespan*.

Apenas dois trabalhos consideraram o bloqueio e o tempo de *setup* dependente da sequência (Norman, 1999; Maleki-Darounkolaei *et al.*, 2012), no entanto nenhum deles considera o tempo de *setup* dependente da sequência e da máquina e com a restrição de bloqueio com *buffer* zero, portanto não existem outros métodos para resolver esse problema.

Como muitos métodos de solução foram propostos neste trabalho, a seguir são descritas as conclusões de cada método separadamente.

10.1. Modelos MILP

Os resultados obtidos pelos modelos *MILP* permitem que seja observado que, apesar de ser um dos fatores que influencia na dificuldade em se solucionar um problema, o número de variáveis e de restrições por si só não determinam esse fator. Isso pode ser observado pelos resultados obtidos, o modelo RBZBS2, por exemplo, possui o menor número de variáveis e o segundo menor número de restrições e obteve os maiores tempos computacionais nos testes realizados.

Também o número de iterações simplex e o número de nós explorados na árvore do Branch-and-Bound não influenciou decisivamente a dificuldade para solucionar o problema. Isso pode ser observado pelo fato do modelo com o menor número de nós explorados na árvore de Branch-and-Bound e o terceiro menor

número de iterações simplex foi o modelo que obteve os maiores tempos computacionais para resolver os problemas.

Pelos resultados pode-se observar ainda que o modelo adaptado RBZBS1 obteve os melhores resultados para os problemas menores e o modelo TNZBS1 obteve os melhores desvios relativos médios do *makespan* para os problemas maiores que não foram resolvidos dentro do tempo computacional estipulado.

10.2. Método Branch-and-Bound

Para o algoritmo *Branch-and-Bound* foram propostos quatro limitantes inferiores para o *makespan* que exploram uma propriedade estrutural do problema, a qual foi demonstrada neste trabalho. Um algoritmo *Branch-and-Bound* foi desenvolvido utilizando esses limitantes inferiores e testes computacionais foram realizados. Os experimentos numéricos indicam que o melhor limitante inferior foi o LB_{TN2}, seguido pelo LB_{TN1}, LB_{TN4} e por último pelo LB_{TN3}. O limitante inferior LB_{TN2} foi melhor que os demais tanto no tempo computacional e no número de nós explorados como também no número de problemas não resolvidos e no desvio relativo médio do *makespan*.

10.3. Comparação Modelo MILP e *Branch-and-Bound*

O melhor limitante inferior para o *Branch-and-Bound* (LB_{TN2}) foi comparado o melhor modelo *MILP* apresentado (TNZBS1). A base de dados utilizada para a comparação foi a mesma utilizada para os testes do modelo *MILP*. O algoritmo *Branch-and-Bound* obteve resultados com tempos computacionais menores que o modelo *MILP*, e, dentre os problemas que não foram resolvidos, também obteve os menores valores de desvio relativo do *makespan*.

Para resultados mais precisos é necessária uma base de dados maior, considerando mais máquinas e mais tarefas, visto que o desempenho do *Branch-and-Bound* tende a melhorar em relação ao *MILP* conforme aumenta-se o número de tarefas e o desempenho do *MILP* tende a melhorar em relação ao *Branch-and-Bound* conforme aumenta-se o número de máquinas.

10.4. Métodos Heurísticos

Neste trabalho as 14 melhores heurísticas construtivas conhecidas para o problema *flow shop* permutacional com bloqueio foram adaptados para o problema estudado de duas formas diferentes, cada uma fornecendo resultados diferentes. As heurísticas adaptadas foram comparadas entre si por meio de uma base de dados composta de 480 problemas (baseados nas instâncias de Taillard, 1993), que variam em número de tarefas e máquinas, e no valor do tempo de *setup*. Todas as heurísticas forneceram resultados únicos que foram testados em relação às suas factibilidades.

Pelos resultados apresentados neste trabalho, para o problema *flow shop* permutacional com *buffer* zero e com tempo de *setup* dependente da sequência e da máquina, a heurística *PF* fornece, de forma geral, resultados um pouco melhores do que os demais métodos em todas as fases.

10.5. Trabalhos Futuros

A seguir são descritos algumas sugestões de melhorias e trabalho futuros.

Em relação aos modelos MILP pode-se tentar desenvolver um modelo que utilize as propriedades estruturais do problema que foram apresentados neste trabalho. Pode ser realizado um estudo mais completo com variações na faixa de valores dos tempos de *setup*, variando entre 10%, 50%, 100% e 125% (tal como foi realizado para os testes dos métodos heurísticos).

Como trabalhos futuros sugere-se ainda o desenvolvimento e uso de uma regra de dominância para reduzir o número de ramificações no algoritmo *Branch-and-Bound*. Neste trabalho foi utilizado a heurística *NEH* para fornecer uma solução inicial (limitante superior) para o algoritmo *Branch-and-Bound*, porém muitas outras heurísticas podem ser utilizadas com esse propósito. As heurísticas *PF*1 e *PW*1 apresentadas nesse trabalho podem ser utilizadas, por exemplo, para a obtenção da solução inicial para o algoritmo *Branch-and-Bound*, por possuírem baixo tempo computacional.

As propriedades estruturais apresentadas neste trabalho podem ainda ser utilizadas para o desenvolvimento de uma heurística construtiva de I, II ou III fase especificamente para o problema *flow shop* permutacional com bloqueio com *buffer* zero e com tempo de *setup* dependente da sequência e da máquina.

É possível se estudar o desenvolvimento de uma meta-heurística utilizando uma das heurísticas construtivas apresentadas neste trabalho para fornecer uma solução inicial. Uma comparação entre os métodos heurísticos construtivos apresentados e um método exato (*e.g.* um modelo *MILP* ou um algoritmo *Branch-and-Bound*) pode ser realizada para se poder estimar a eficiência das heurísticas adaptadas, uma vez que não existem outros resultados para o problema estudado para se realizar uma comparação com os resultados obtidos pelas heurísticas propostas.

REFERÊNCIAS

ABADI, I. N. K.; HALL, N. G.; SRISKANDARAJAH, C. Minimizing cycle time in a blocking flowshop. **Operations Research**, v. 48, n. 1, p. 177-180, 2000.

ABYANEH, S. H.; ZANDIEH, M. Bi-objective hybrid flow shop scheduling with sequence-dependent setup times and limited buffers. **International Journal of Advanced Manufacturing Technology**, v. 58, n. 1-4, p. 309-325, 2012.

ACKOFF, R. L.; SASIENI, M. W. **Pesquisa operacional**. Tradução de José L. Moura Marques e Cláudio Graell Reis. 1. ed. 1971. 523 p. (Coleção Universitária de Administração, vol. 4).

ARAUJO, D. C.; NAGANO, M. S. A new effective heuristic method for the no-wait flowshop with sequence-dependent setup times problem. **International Journal of Industrial Engineering Computations**, v. 2, n. 1, p. 155-166, 2011.

ARMENTANO, V. A.; RONCONI, D. P. Minimização do tempo total de atraso no problema de flowshop com buffer zero através de busca tabu. **Gestão e Produção**, v. 7, n. 3, p. 352-363, 2000.

BAGCHI, T. P.; GUPTA, J. N.; SRISKANDARAJAH, C. A review of TSP based approaches for flowshop scheduling. **European Journal of Operational Research**, v. 169, n. 3, p. 816-854, 2006.

BAUTISTA, J.; CANO, A.; COMPANYS, R.; RIBAS, I. Solving the Fm/block/Cmax problem using Bounded Dynamic Programming. **Engineering Applications of Artificial Inteligence**, v. 25, n. 6, p. 1235-1245, 2012.

BELLMAN, R. E.; ESOGBUE, A. O.; NABESHIMA, I. **Mathematical Aspects of Scheduling and Applications**. 1982. 329 p.

BISKUP, D.; HERRMANN, J. Single-machine scheduling against due dates with past-sequence-dependent setup times. **European Journal of Operational Research**, v. 191, n. 2, p. 587-592, 2008.

BOIKO, T. J. P. **Métodos heurísticos para a programação em flow shop permutacional com tempos de setup separados dos tempos de processamento e independentes da seqüência de tarefas**. 2008. 220 p. Dissertação (Mestrado em engenharia da produção) - Escola de engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

CARAFFA, V.; IANES, S.; BAGCHI, T. P.; SRISKANDARAJAH, C. Minimizing makespan in a blocking flowshop using genetic algorithms. **International Journal of Production Economics**, v. 70, n. 2, p. 101-115, 2001.

CHANG, J.; YAN, W.; SHAO, H. **Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times**. 2004 American Control Conference. Boston: ACC. 2004. p. 1412-1416.

CHANG, P. -C.; CHEN, S. -H. The development of a sub-population genetic algorithm II (SPGA II) for multi-objective combinatorial problems. **Applied Soft Computing**, v. 9, n. 1, p. 173-181, 2009.

CHEN, H.; ZHOU, S.; LI, X.; XU, R. A hybrid differential evolution algorithm for a two-stage flow shop on batch processing machines with arbitrary release times and blocking. **International Journal of Production Research**, v. 52, n. 19, p. 5714-5734, 2014.

COMPANYS, R.; MATEO, M. Different behaviour of a double branch-and-bound algorithm on Fm|prmu|Cmax and Fm|block|Cmax problems. **Computers & Operations Research**, v. 34, n. 4, p. 938-953, 2007.

DAVENDRA, D.; BIALIC-DAVENDRA, M. Scheduling flow shops with blocking using a discrete self-organising migrating algorithm. **International Journal of Production Research**, v. 51, n. 8, p. 2200-2218, 2013.

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm (NSGA-II). **IEEE Transactions on Evolutionary Computation**, v. 6, n. 2, p. 182-197, 2002.

DING, J.-Y.; SONG, S.; GUPTA, J. N. D.; WANG, C.; ZHANG, R.; WU, C. New block properties for flowshop scheduling with blocking and their application in an iterated greedy algorithm. **International Journal of Production Research**, v. 54, p. DOI: 10.1080/00207543.2015.1076941, 2015.

DORIGO, M. **Optimization, Learning and Natural Algorithms [em italiano]**. 1992. 140 p. Ph. D. Thesis (Systems and Information Engineering) - Politecnico di Milano, Milano, 1992.

ECKER, K. H.; GUPTA, J. N. D. Scheduling tasks on a flexible manufacturing machine to minimize tool change delays. **European Journal of Operational Research**, v. 164, n. 3, p. 627-638, 2005.

FRAMINAN, J. M.; GUPTA, J. N. D.; LEISTEN, R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. **Journal of the Operational Research Society**, v. 55, n. 12, p. 1243-1255, 2004.

FRENCH, S. **Sequencing and Scheduling:** An introduction to the mathematics of the job shop. 1982. 245 p.

GEEM, Z. W.; KIM, J. H.; LOGANATHAN, G. V. A new heuristic optimization algorithm: Harmony search. **Simulations**, v. 76, n. 2, p. 60-68, 2001.

GLOVER, F. Future Paths for Integer Programming and Links to Artificial Intelligence. **Computers and Operations Research**, v. 13, n. 5, p. 533-549, 1986.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 1. ed. 1989. 372 p.

GÓMEZ-GASQUET, P.; ANDRÉS, C.; LARIO, F. C. An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan. **Expert Systems with Applications**, v. 39, n. 9, p. 8095-8107, 2012.

GONG, H.; TANG, L.; DUIN, C. W. A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. **Computers & Operations Research**, v. 37, n. 5, p. 960-969, 2010.

GONZÁLEZ, M. A.; VELA, C. R.; VARELA, R. A competent memetic algorithm for complex scheduling. **Natural Computing**, v. 11, n. 1, p. 151-160, 2012.

GRABOWSKI, J.; PEMPERA, J. The permutation flow shop problem with blocking: A tabu search approach. **Omega**, v. 35, n. 3, p. 302-311, 2007.

GRABOWSKI, J.; SKUBALSKA, E.; SMUTNICKI, C. On flow shop scheduling with release and due dates to minimize maximum lateness. **Journal of Operational Research Society**, v. 34, n. 7, p. 615-620, 1983.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; RINNOOY KAN, A. H. G. **Optimization and approximation in deterministic sequencing and scheduling:** A survey. Discrete Optimization Symposium. Vancouver: Annals of Discrete Mathematics. 1979. p. 287-326.

GROOVER, M. P. **Automação Industrial e Sistemas de Manufatura**. Tradução de Jorge Ritter; Luciana do Amaral Teixieira e Marcos Vieira. 3a. ed. 2008. 582 p.

GUANLONG, D.; ZHENHAO, X.; XINGSHENG, G. A Discrete Artificial Bee Colony Algorithm for Minimizing the Total Flow Time in the Blocking Flow Shop

Scheduling. **Chinese Journal of Chemical Engineering**, v. 20, n. 6, p. 1067-1073, 2012.

GUPTA, J. N. D.; DARROW, W. P. The two-machine sequence dependent flowshop scheduling problem. **European Journal of Operational Research**, v. 24, n. 3, p. 439-446, 1986.

GUPTA, J. N.; STRUSEVICH, V. A.; ZWANEVELD, C. M. Two-stage no-wait scheduling models with setup and removal times separated. **Computers & Operations Research**, v. 24, n. 11, p. 1025-1031, 1997.

HALL, N. G.; SRISKANDARAJAH, C. A survey of machine scheduling problems with blocking and no-wait in process. **Operations Research**, v. 44, n. 3, p. 510-525, 1996.

HAN, Y.-Y.; DUAN, J. H.; YANG, Y. J.; ZHANG, M.; YUN, B. **Minimizing the Total Flowtime Flowshop with Blocking Using a Discrete Artificial Bee Colony**. 7th international conference on Advanced Intelligent Computing Theories and Applications: with aspects of artificial intelligence. Berlin: Lecture Notes in Computer Science. 2012a. p. 91-97.

HAN, Y.-Y.; GONG, D.; LI, J.; ZHANG, Y. Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. **International Journal of Production Research**, p. DOI: 10.1080/00207543.2016.1177671, 2016.

HAN, Y.-Y.; GONG, D.; SUN, X. A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. **Engineering Optimization**, v. 47, n. 7, p. 927-946, 2015.

HAN, Y.-Y.; LIANG, J. J.; PAN, Q.-K.; LI, J.-Q. Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. **The International Journal of Advanced Manufacturing Technology**, v. 67, n. 1, p. 397-414, 2013.

HAN, Y.-Y.; PAN, Q. K.; LI, J. Q.; SANG, H. Y. An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. **International Journal Advanced Manufacturing Technology**, v. 60, n. 9-12, p. 1149-1159, 2012b.

HILLIER, F. S.; LIEBERMAN, G. J. **Operations Research**. 2a. ed. 1974. 800 p.

IGNALL, E.; SCHRAGE, L. Application of the branch and bound technique to some flow-shop scheduling problems. **Operations research : the journal of the Operations Research Society of America**, v. 13, n. 3, p. 400-412, 1965.

JERALD, J.; ASOKAN, P.; PRABAHARAN, G.; SARAVANAN, R. Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. **International Journal of Advanced Manufacturing Technology**, v. 25, n. 9-10, p. 964-971, 2005.

JOHSON, S. M. Optimal two- and three-stage production schedules with setup times included. **Naval Research Logistics Quarterly**, v. 1, n. 1, p. 61-68, 1954.

KENNEDY, J.; EBERHART, R. **Particle swarm optimization. In:** International Conference on Neural Networks. International Conference on Neural Networks. Perth: IEEE International Conference on Neural Networks. 1995. p. 1942-1948. DOI: 10.1109/ICNN.1995.488968.

KIM, K.; JEONG, I. J. Flow shop scheduling with no-wait flexible lot streaming using an adaptive genetic algorithm. **International Journal of Advanced Manufacturing Technology**, v. 44, n. 11-12, p. 1181-1190, 2007.

KIM, Y.-D. Minimizing mean tardiness inpermutation flowshops. **European Journal of Operational Research**, v. 85, n. 3, p. 541-555, 1995.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. **Science**, v. 220, n. 4598, p. 671-680, 1983.

LEE, C. H.; LIAO, C. J.; CHAO, C. W. Scheduling with multi-attribute setup times. **Computer & Industrial Engineering**, v. 63, n. 2, p. 494-502, 2012.

LEE, D. Y.; DICESARE, F. Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search. **IEEE Transactions on Robotics and Automation**, v. 10, n. 2, p. 123-132, 1994.

LEISTEN, R. Flowshop sequencing problems with limited buffer storage. **International Journal of Production Research**, v. 28, n. 11, p. 2085-2100, 1990.

LI, X.; CHEHADE, H.; YALAOUI, F.; AMODEO, L. **Lorenz dominance based metaheuristic to solve a hybrid flowshop scheduling problem with sequence dependent setup times**. 2011 International Conference on Communication, Computing and Control Applications. Hammamet: Browse Conference Publications. 2011. p. 1-6.

LIN, S. W.; YING, K. C. Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. **Omega**, v. 41, n. 2, p. 383-389, 2013.

LIU, B.; WANG, L.; JIN, Y. H. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. **Computers and Operation Research**, v. 35, n. 9, p. 2791-2806, 2008.

LIU, S. Q.; KOZAN, E. Scheduling a flow shop with combined buffer conditions. **International Journal Production Economics**, v. 117, n. 2, p. 371-380, 2009.

LOW, C.; WU, T. H.; HSU, C. M. Mathematical modelling of multi-objective job shop scheduling with dependent setups and re-entrant operations. **International Journal of Advanced Manufacturing Technology**, v. 27, n. 1-2, p. 181-189, 2005.

MACCARTHY, B. L.; LIU, J. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. **International Journal of Production Research**, v. 31, n. 1, p. 59-79, 1993.

MALEKI-DAROUNKOLAEI, A.; MODIRI, M.; TAVAKKOLI-MOGHADDAM, R.; SEYYEDI, I. A three-stage assembly flow shop scheduling problem with blocking and sequence-dependent set up times. **Journal of Industrial Engineering International**, v. 8, n. 1, p. 1-7, 2012.

MARTINEZ, S.; DAUZÈRE-PÉRÈS, S.; GUÉRET, C.; MATI, Y.; SAUER, H. Complexity of flowshop scheduling problems with a new blocking constraint. **European Journal of Operational Research**, v. 169, n. 3, p. 855-864, 2006.

MCCORMICK, S. T.; PINEDO, M. L.; SHENKER, S.; WOLF, B. Sequencing in an Assembly Line with Blocking to Minimize Cycle Time. **Operations Research**, v. 37, n. 6, p. 925 - 935, 1989.

MOCCELLIN, J. V. **Uma contribuição à programação de operações em sistemas de produção intermitente 'flow-shop'**. 1992. 126 p. Tese (Livre-docência em engenharia da produção) - Escola de engenharia de São Carlos, Universidade de São Paulo, São Carlos, 1992.

MOCCELLIN, J. V.; NAGANO, M. S. Heuristic for Flow Shop Sequencing with Separated and Sequence Independent Setup Times. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, v. 33, n. 1, p. 74-78, 2011.

MONTEVECHI, J. A.; TURRIONI, J. B.; ALMEIDA, D. A.; MERGULHÃO, R. C. Análise comparativa entre regras heurísticas de seqüenciamento da produção aplicada em job shop. **Produto e Produção**, v. 6, n. 2, p. 12-18, 2002.

MOSLEHI, G.; KHORASANIAN, D. Optimizing blocking flow shop scheduling problem with total completion time criterion. **Computer and Operations Research**, v. 40, n. 7, p. 1874-1883, 2013.

NAGANO, M. S.; MESQUITA, M. S. Métodos heurísticos para o problema de programação flow shop com tempos de setup separados. **Revista Produção Online**, v. 12, n. 2, p. 499-521, 2012. Disponível em: http://producaoonline.org.br/rpo/article/view/939/916. Acesso em: 19 Nov. 2012.

NAWAZ, M.; ENSCORE, E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega**, v. 11, n. 1, p. 91-95, 1983.

NORMAN, B. A. Scheduling flowshops with finite buffers and sequence-dependent setup times. **Computer & Industrial Engineering**, v. 16, n. 1, p. 163-177, 1999.

NOWICKI, E. The permutation flow shop with buffers: A tabu search approach. **European Journal of Operational Research**, v. 116, n. 1, p. 205-219, 1999.

OLIVEIRA, A. C. M.; LORENA, L. A. N. **Detecting promising areas by evolutionary clustering search**. 17th Brazilian Symposium on Artificial Intelligence. São Luiz: Lecture Notes in Computer Science. 2004. p. 385-394.

PAN, Q. K.; TASGETIREN, M. F.; LIANG, Y. C. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. **Computers and Industrial Engineering**, v. 55, n. 4, p. 795-816, 2008.

PAN, Q. K.; WANG, L. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. **Omega**, v. 40, n. 2, p. 218-229, 2012.

PAN, Q. K.; WANG, L.; GAO, L. A chaotic harmony search algorithm for the flow shop scheduling problem with limited buffers. **Applied Soft Computing**, v. 11, n. 8, p. 5270-5280, 2011b.

PAN, Q. K.; WANG, L.; GAO, L.; LI, W. D. An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers. **Information Sciences**, v. 181, n. 3, p. 668-685, 2011a.

PAN, Q.-K.; WANG, L.; SANG, H.-Y.; LI, J.-Q.; LIU, M. A High Performing Memetic Algorithm for the Flowshop Scheduling Problem With Blocking. **IEEE Transactions on Automation Science and Engineering**, v. 10, n. 3, p. 741-756, 2013.

PAPADIMITRIOU, C.; KANELLAKIS, P. Flow-shop scheduling with limited temporary storage. **Journal of the Association for Computing Machinery**, v. 27, n. 3, p. 533-549, 1980.

PINEDO, M. L. **Scheduling:** Theory, algorithms, and systems. 3. ed. 2008. 671 p.

QIAN, B.; WANG, L.; HUANG, D. X.; WANG, X. An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. **International Journal of Production Research**, v. 36, n. 1, p. 209-233, 2009.

REDDI, S. S.; RAMAMOORTHY, C. V. On the flowshop sequencing problem with no-wait in process. **Operational Research Quarterly**, v. 23, n. 3, p. 323-330, 1972.

RIBAS, I.; COMPANYS, R. Efficient heuristic algorithms for the blocking flow shop scheduling problem with total flow time minimization. **Computers & Industrial Engin eering**, v. 87, p. 30-39, 2015.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. An iterated greedy algorithm for the flowshop scheduling with blocking. **Omega**, v. 39, n. 3, p. 293-301, 2011.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. **International Journal of Production Research**, v. 51, n. 17, p. 5238-5252, 2013a.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. A competitive variable neighbourhood search. **European J. Industrial**, v. 7, n. 6, p. 729-754, 2013b.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. An efficient Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with total flowtime minimization. **Expert Systems with Applications**, v. 42, n. 15-16, p. 6155-6167, 2015.

RIOS-MERCADO, R. Z.; BARD, J. F. A Branch-and-Bound Algorithm for Flowshop Scheduling with Setup Times. **IIE Transactions on Scheduling & Logistics**, v. 31, n. 8, p. 721-731, 1999.

RONCONI, D. P. A note on constructive heuristics for the flowshop problem with blocking. **International Journal of Production Economics**, v. 87, n. 1, p. 39-48, 2004.

RONCONI, D. P. A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking. **Annals of Operations Research**, v. 138, n. 1, p. 53 - 65, 2005.

RONCONI, D. P.; ARMENTANO, V. A. Lower bounding schemes for flowshops with blocking in-process. **Journal of the Operational Research Society**, v. 52, n. 11, p. 1289-1297, 2001.

RONCONI, D. P.; BIRGIN, E. G. Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness. In: RÍOS-MERCADO, R. Z.; RÍOS-SOLÍS, Y. A. **Just-in-Time Systems**. 1. ed. Nova York: Springer, v. 60, 2012. Cap. 5, p. 91-105.

RONCONI, D. P.; HENRIQUES, L. R. D. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. **Omega**, v. 37, n. 2, p. 272-281, 2009.

SABOUNI, M. T. Y.; LOGENDRAN, R. Carryover sequence-dependent group scheduling with the integration of internal and external setup times. **European Journal of Operational research**, v. 224, n. 1, p. 8-22, 2013.

SADAQA, M.; MORAGA, R. J. **Scheduling Blocking Flow Shops Using Meta-RaPS**. Complex Adaptive Systems San Jose. San Jose: Procedia Computer Science. 2015. p. 533-538.

SAIDI-MEHRABAD, M.; FATTAHI, P. Flexible job shop scheduling with tabu search algorithms. **International Journal of Advanced Manufacturing Technology**, v. 32, n. 5-6, p. 563-570, 2007.

STÜTZLE, T. **An Ant Approach to the Flow Shop Problem**. 6th European Congress on Intelligent Techniques & Soft Computing. Aachen: Eufit. 1998. p. 1560-1564.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, v. 64, n. 2, p. 278-285, 1993.

TASGETIREN, M. F.; PAN, Q. K.; SUGANTHAN, P. N.; CHEN, A. H. L. A discrete artificial bee colony algorithm for the total flow time minimization in permutation flow shops. **Information Sciences**, v. 181, n. 16, p. 3459-3475, 2011.

TOUMI, S.; JARBOUI, B.; EDDALY, M.; REBAÏ, A. **Solving Blocking Flowshop Scheduling Problem**. International Conference on Control, Decision and Information Technologies (CoDIT). Hammamet: IEEE. 2013a. p. 756-761.

TOUMI, S.; JARBOUI, B.; EDDALY, M.; REBAÏ, A. **Lower bounds on the total tardiness and total weighted tardiness for scheduling flowshop with blocking**. 2013 International Conference on Advanced Logistics and Transport (ICALT). Sousse: IEEE. 2013b. p. 296-301.

TOUMI, S.; JARBOUI, B.; EDDALY, M.; REBAÏ, A. **An efficient Branch and Bound algorithm to solve the permutation flowshop scheduling problem with blocking constraints**. 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO), 2013. Hammamet: IEEE. 2013c. p. 1-6.

TRABELSI, W.; SAUVEY, C.; SAUER, N. Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems. **Computers & Operations Research**, v. 39, n. 11, p. 2520-2527, 2012.

TUBINO, D. F. **Planejamento e controle da produção:** Teoria e prática. 1a. ed. 2007. 190 p.

VALLADA, E.; RUIZ, R. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. **European Journal of Operational Research**, v. 211, n. 3, p. 612-622, 2011.

WANG, C.; SONG, S.; GUPTA, J. N. D.; WU, C. A three-phase algorithm for flowshop scheduling with blocking to minimize makespan. **Computers & Operations Research**, v. 39, n. 11, p. 2880-2887, 2012.

WANG, L.; PAN, Q. K.; SUGANTHAN, P. N.; WANG, W. H.; WANG, Y. M. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. **Computers & Operations Research**, v. 37, n. 3, p. 509-520, 2010a.

WANG, L.; PAN, Q. K.; TASGETIREN, M. F. Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. **Expert Systems with Applications**, v. 37, n. 12, p. 7929-7936, 2010b.

WANG, L.; PAN, Q. K.; TASGETIREN, M. F. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. **Computers & Industrial Engineering**, v. 61, n. 1, p. 76-83, 2011.

WANG, L.; ZHANG, L.; ZHENG, D. Z. The ordinal optimisation of genetic control parameters for flow shop scheduling. **International Journal of Advanced Manufacturing Technology**, v. 23, n. 11-12, p. 812-819, 2004.

WANG, L.; ZHANG, L.; ZHENG, D. Z. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. **Computers & Operations Research**, v. 33, n. 10, p. 2960-2971, 2006.

WANG, X.; TANG, L. A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. **Applied Soft Computing**, v. 12, n. 2, p. 652-662, 2012.

XI, Y.; JANG, J. Scheduling jobs on identical parallel machines with unequal future ready time and sequence dependent setup: An experimental study. **International Journal of Production Economics**, v. 137, n. 1, p. 1-10, 2012.