UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE MECÂNICA
CURSO DE ENGENHARIA MECÂNICA

FELIPE BORREIRO SANCHES

COMPARAÇÃO ENTRE DIFERENTES FORMAS DE OBTENÇÃO DA SOLUÇÃO INICIAL PARA A APLICAÇÃO DO MÉTODO *BRANCH-AND-BOUND* PARA SOLUCIONAR PROBLEMAS DE SEQUENCIAMENTO EM AMBIENTES *FLOW-SHOP* COM BLOQUEIO

TRABALHO DE CONCLUSAO DE CURSO

Cornélio Procópio
2015

FELIPE BORREIRO SANCHES

COMPARAÇÃO ENTRE DIFERENTES FORMAS DE OBTENÇÃO DA SOLUÇÃO INICIAL PARA A APLICAÇÃO DO MÉTODO BRANCH-AND-BOUND PARA SOLUCIONAR PROBLEMAS DE SEQUENCIAMENTO EM AMBIENTES FLOW-SHOP COM BLOQUEIO

Trabalho de Conclusão de Curso
apresentado como requisito parcial
à obtenção do título de Bacharel
em Engenharia Mecânica da
Universidade Tecnológica Federal
do Paraná.

Orientador: Prof. Me. Mauricio Iwama Takano

CORNÉLIO PROCÓPIO
2015

Universidade Tecnológica Federal do Paraná¡
Campus Cornélio Procópio
Departamento Acadêmico de Mecânica
Curso de Engenharia Mecânica

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

FOLHA DE APROVAÇÃO

Felipe Borreiro Sanches

Comparação entre diferentes formas de obtenção da solução inicial para a aplicação do método Branch-and-Bound para solucionar problemas de sequenciamento em ambientes flow-shop com bloqueio

Trabalho de conclusão de curso apresentado às hs do dia como requisito parcial para a obtenção do título de Engenheiro Mecânico no programa de Graduação em Engenharia Mecânica da Universidade Tecnológica Federal do Paraná. O candidato foi arguido pela Banca Avaliadora composta pelos professores abaixo assinados. Após deliberação, a Banca Avaliadora considerou o trabalho aprovado.

_____

Prof(a). Me(a). Mauricio Iwama Takano - Presidente (Orientador)

_____

Prof(a). Dr(a). José Tomadon Júnior - (Membro)

_____

Prof(a). Me(a). Vitor Miranda de Souza - (Membro)

A folha de aprovação assinada encontra-se na coordenação do curso.

À minha família e a todos os amigos.

AGRADECIMENTOS

Agradeço primeiramente a Deus por minha vida, família, amigos e pelo grupo En-Hacoré.

Agradeço aos meus pais, Tânia e Nilson, por todo amor e carinho que sempre me deram e por todo apoio durante toda minha graduação.

Ao meu orientador Prof. Me. Mauricio Iwama Takano, pela excelente orientação, disponibilidade e paciência durante todo este trabalho.

Aos meus colegas e professores da UTFPR – CP que sempre estiveram prontos a ajudar de alguma forma na realização deste trabalho

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS E ABREVIAÇÕES

$C_{máx}$	Tempo total de programação
DR	Desvio relativo
DM	Resultado obtido
DM*	Melhor resultado obtido
Fm	Flow Shop
J_j	Tarefa j
MM	*MinMax*
M_k	Máquina k
M_f	Tempo de fluxo
m	Número de máquinas
n	Número de tarefas
PF	*Profile Fitting*
PW	*Profile Weighted*
P_{jk}	Tempo de processamento
Prmu	Permutacional
TC	Tempo computacional
wPF	*Weighted Profile Fitting*

SANCHES, FELIPE B. (2015). Comparação entre diferentes formas de obtenção da solução inicial para a aplicação do método Branch-and-Bound para solucionar problemas de sequenciamento em ambientes Flow-Shop com bloqueio. Cornélio Procópio: Curso Superior de Engenharia Mecânica, Universidade Tecnológica Federal do Paraná.

RESUMO

Este trabalho tem o objetivo de comparar o uso de diferentes métodos para a obtenção de uma solução inicial para o algoritmo *Branch-and-Bound* com o objetivo de minimizar o *makespan* em um ambiente *flow-shop* sem estoque intermediário. Como o problema é conhecido por ser *NP-Hard*, o algoritmo *Branch-and*-Bound pode tomar muito tempo computacional para encontrar a melhor solução. A utilização de uma solução inicial pode reduzir o tempo computacional proporcionando um limitante superior inicial. A eficiência da utilização de uma solução inicial para o algoritmo *Branch-and-Bound* foi medida comparando o algoritmo tradicional (sem uma solução inicial) e quatro outros algoritmos que utilizam soluções iniciais diferentes fornecidas por quatro heurísticas construtivas de I fase (conforme classificação de Framinan et al.,2004). Os métodos heurísticos utilizados para fornecer as soluções iniciais são: *MM* (Ronconi, 2004); *PF* (McCormick et al., 1989); *PW* e *wPF* (Pan e Wang, 2012). O algoritmo *Branch-and-Bound* utilizado, bem como o limite inferior foram propostos por Ronconi (2005). Os cinco métodos foram testados utilizando uma base de dados 180 problemas. Os resultados mostram que o uso de uma solução inicial é capaz de reduzir o tempo computacional considerável, principalmente em problemas de grande porte.

Palavras Chave: *Branch-and-Bound*, *flow-shop*, bloqueio, *makespan*, sequenciamento.

SANCHES, FELIPE B. (2015). Comparison among heuristic methods to provide an initial solution for a Branch-and-Bound algorithm to minimize the makespan in a flow shop with blocking environment. Cornélio Procópio: Degree in Mechanical Engineering, Federal University of Technology of Parana.

ABSTRACT

The objective of this paper is to compare the use of different methods to obtain an initial solution for the Branch-and-Bound algorithm with the objective of minimizing the makespan in a flow shop with zero buffer environment. As the problem is known to be NP-Hard, the Branch-and-Bound algorithm may take great computational time to find the best solution. The use of an initial solution may reduce the computational time, by providing an initial upper bound. The efficiency of the use of an initial solution for the Branch-and-Bound algorithm is measured by comparing the traditional algorithm (without an initial solution) and four other algorithms which use different initial solutions provided by four different I phase constructive heuristics (according to Framinan et al., 2004 classification) each. The heuristic methods used to provide the initial solutions are: *MM* (Ronconi,2004); *PF* (McCormick et al., 1989); *PW* and *wPF* (Pan and Wang, 2012). The Branch-and-Bound algorithm used as well as the lower bound were proposed by Ronconi (2005). The five methods were tested using a 180 problems database. Results show that the use of an initial solution does reduce considerable computational time, mostly in large problems.

Key-words: Branch-and-Bound, flow shop, block, makespan, scheduling.

1. INTRODUÇÃO

Sequenciamento da produção é um processo de tomada de decisões que é usado diariamente em muitos serviços e manufaturas na indústria, com o objetivo de otimizar um ou mais objetivos, como por exemplo minimizar o *makespan* de um processo, número de pedidos atrasados, entre outros. Ele lida com a alocação de operações em recursos (PINEDO, 2008).

Em específico na produção, o sequenciamento é o processo de decisão da ordem de processamento das tarefas, ou seja, qual tarefa deve ser processada primeiro e quais devem ser processadas em seguida (PINEDO, 2008).

Segundo (PINEDO, 2008) uma boa programação pode levar a uma minimização dos custos, de desperdícios e mão de obra de uma empresa, permitindo uma superior projeção do crescimento.

Em ambientes *flow-shop* a restrição de bloqueio com *buffer* zero representa a ausência de estoques intermediários entre máquinas, sendo assim, não há como uma tarefa sair de uma máquina e esperar em um estoque intermediário, bloqueando assim a máquina em que ela se encontra até que a próxima máquina esteja liberada.

O *makespan* é equivalente à data de término da ultima tarefa da sequência. A minimização do *makespan* é um objeto de estudo na indústria que tem como finalidade conseguir o aumento da produtividade na empresa, sendo assim, sua importância é significativa dentro do contexto moderno de indústria.

Existem variados métodos para encontrar uma solução para o problema de sequenciamento da produção que satisfaça o critério de avaliação. Dentre os métodos existem os métodos exatos (ou matemáticos), que proporcionam uma ótima qualidade nas suas soluções, porém, possuem um elevado tempo computacional para achar soluções ideais para seus problemas. Neste contexto, este trabalho tem como objetivo identificar uma forma de reduzir o tempo computacional dos métodos exatos, utilizando métodos heurísticos (que é caracterizado por proporcionar soluções próximas às ótimas, mas com um reduzido tempo computacional)como forma de obtenção de uma solução inicial para o problema.

1.1. OBJETIVOS

1.1.1. Objetivo Geral

Comparação entre diferentes métodos de obtenção da solução inicial, para o *Branch-and-Bound* em ambientes *flow-shop* permutacional com bloqueio e com objetivo de minimizar o *makespan*.

1.1.2. Objetivos Específicos

- Programar o limitante inferior conforme modelo proposto por Ronconi (2005);
- Programar o método *Branch-and-Bound* sem utilização da solução inicial;
- Programar os melhores métodos heurísticos de I Fase encontrados na literatura para problemas em ambiente *flow-shop* permutacional com bloqueio e com o objetivo de minimizar o *makespan*;
- Programar o *Branch-and-Bound* utilizando os diferentes métodos heurísticos para obtenção da solução inicial;
- Validar os métodos utilizando base de dados conhecida, variando número de tarefas e número de máquinas;
- Comparar os resultados analisando o tempo computacional e o número de ramificações.

1.2. ESTRUTURA DO TRABALHO

Este trabalho foi dividido em cinco capítulos, o primeiro apresenta a introdução de conceitos do sistema de produção e de sequenciamento da produção. No segundo capítulo, referencial teórico, são descritos os ambientes operacionais, as descrições de problemas, a definição e classificação das heurísticas e a definição dos métodos exatos, é comentado também os métodos de solução encontrados na literatura. No terceiro capítulo, é feito um detalhado o modelo *Branch-and-Bound* com e sem o uso de uma solução inicial. No quarto

capítulo é especificado o desenvolvimento, mostrando como foi aplicada e executada a programação do modelo *Branch-and-Bound* e dos modelos com soluções iniciais na base de dados selecionada. O quinto capítulo são apresentadas as conclusões que puderam ser retiradas por meio da experimentação computacional feita e mostrando possíveis extensões a este trabalho.

2. SEQUENCIAMENTO DA PRODUÇÃO

Para o estudo dos problemas de sequenciamento da produção, algumas notações são utilizadas, são elas:

- **Tempo de processamento** (P_{jk}): Representa o tempo de processamento da tarefa *j* na máquina *k*;
- **Data de liberação** (R_j): Representa a data a partir da qual a tarefa *j* pode iniciar o seu processamento na primeira máquina;
- **Prazo de entrega** (d_j): É a data de término do processamento da tarefa *j*, data na qual ela deve ser terminada;
- **Tempo de *setup*** (S_j): É o tempo de preparação (*setup*) da máquina antes dela começar a processar a tarefa *j*;
- **Peso** (w_j): É definido como a importância, ou prioridade, dada à tarefa *j* em relação às outras tarefas.

Segundo (PINEDO, 2008) os problemas de sequenciamento podem ser definidos por três campos: α|β|ɣ. No campo α é descrito o ambiente de máquina, no campo β são representadas as características ou restrições tecnológicas do problema e no campo ɣ é apresentado o critério de avaliação escolhido. O campo α possui somente uma entrada, o campo β pode ter nenhuma, somente uma ou várias entradas ao mesmo tempo, e o campo ɣ pode conter somente uma ou várias entradas.

Os possíveis ambientes de máquina no **campo α** são (PINEDO, 2008):

- **Máquina única** (1): Somente uma máquina está disponível para o processamento das tarefas;
- **Máquinas idênticas em paralelo** (P_m): Existem *m* máquinas idênticas em paralelo com *n* tarefas para serem processadas, essas tarefas podem ser processadas em qualquer uma dessas *m* máquinas e somente uma máquina é necessária para o processamento;

- **Máquinas em paralelo com diferentes velocidades** (Q_m): Existem *m* máquinas em paralelo com diferentes velocidades. A velocidade da máquina é denotada por v_k. O tempo P_{jk} que a tarefa *j* demora para ser processada na máquina *k* é igual a P_j / v_k;
- **Máquinas independentes em paralelo** (R_m): É uma generalização do modelo anterior. Uma máquina *k* pode processar uma tarefa *j* a uma velocidade v_{jk}, ou seja, a velocidade depende da máquina e da operação que está sendo realizada;
- ***Flow-shop*** (F_m): Existem *m* máquinas em série e cada tarefa tem que passar por cada uma das máquinas, sendo que todas as tarefas devem seguir o mesmo fluxo;
- ***Flow-shop* flexível** (FF_c): É uma generalização do *flow-shop* e do ambiente de máquinas em paralelo. Ao invés de *m* máquinas em série, existem *c* centros de trabalho agrupando máquinas idênticas que funcionam em paralelo. Todas as tarefas seguem o mesmo fluxo, mas necessita de apenas uma única máquina em cada estação de trabalho;
- ***Job shop*** (J_m): Em um *Job shop* com *m* máquinas, cada tarefa tem um fluxo pré-determinado para seguir. Neste modelo, cada tarefa segue um caminho diferente. É caracterizado por ter uma alta flexibilidade, porém um baixo volume de produção;
- ***Job Shop* flexível** (FJ_c): É uma generalização do *Job shop* e do ambiente de máquinas em paralelo. Ao invés de ter *m* máquinas em série, existem *c* centros de trabalho no qual cada um possuí um número idêntico de máquinas em paralelo. Uma tarefa *j* requer processamento em cada centro de trabalho e apenas em uma máquina e qualquer máquina pode ser utilizada.
- ***Open Shop*** (O_m): Existem *m* máquinas. Cada tarefa tem que ser processada novamente em cada uma das *m* máquinas. Não há restrições no que diz respeito a rota de cada tarefa através do ambiente de máquina.

No **campo β** algumas das possíveis entradas são (PINEDO, 2008):

- **Data de lançamento** (r_j): Se este símbolo aparecer no campo β então a tarefa *j* não pode começar seu processo antes da sua data de lançamento;
- ***Preempção*** (*prmp*): Implica que não é necessário manter uma tarefa *j* até o término de seu processamento na máquina, o processamento pode ser interrompido a qualquer momento e outra tarefa pode entrar na máquina. Quando esta tarefa voltar à máquina, o tempo necessário para finalizar seu processamento é igual ao tempo restante do processamento;
- **Restrições de precedência** (*prec*): Ela indica que uma ou mais tarefas devem ser completadas antes que a próxima tarefa possa começar;
- **Permutação** (*prmu*): Uma restrição que pode aparecer no ambiente *flow-shop* é que o estoque intermediário de cada máquina opere de acordo com a regra do *First in First Out(FIFO)*. Esta regra indica que a ordem (ou permutação) na qual as tarefas entram no sistema é mantido durante todas as operações.
- **Bloqueio** (*block*): É um fenômeno que ocorre em ambientes *flow-shop* quando há um estoque intermediário limitado entre máquinas e este encontra-se cheio. Isto faz com que a peça fique aguardando dentro da máquina a liberação para o estoque intermediário, impedindo (bloqueando) a máquina de receber as próximas tarefas da sequência. Quando o símbolo *block* não aparece no campo β, então considera-se que o estoque intermediário entre todos os pares de máquina é infinito.

O **campo ɣ** envolve os critérios de avaliação. Alguns possíveis critérios que podem ser adicionados ao campo são (PINEDO, 2008):

- ***Makespan*** (C_{max}): O *makespan* é equivalente à data de término da ultima tarefa da sequência;
- **Atraso máximo** (L_{max}): É o tempo máximo de atraso das datas de término;

- **Tempo de fluxo ponderado** ($\Sigma w_j C_j$): O tempo de fluxo ponderado é a soma ponderada dos tempos de término de todas as *n* tarefas. O peso atribuído a cada tarefa pode indicar, por exemplo, o custo de se manter uma tarefa parada na linha.

2.1. MÉTODOS DE SOLUÇÂO

A aplicação do sequenciamento da produção pode ser executada por meio de métodos de pesquisa operacional, dentre eles os métodos exatos (matemáticos) e os Heurísticos.

A aplicação destes métodos se dá em busca de otimizar os resultados e buscar maneiras de melhor atender os critérios de avaliação adotados.

2.1.1. Métodos Heurísticos

Durante os últimos 40 anos, o problema de sequenciamento com o objetivo de minimização do *makespan* no ambiente *flow-shop* com restrição de bloqueio tem sido atração de estudo de muitos pesquisadores. Este problema, que é caracterizado como $F_m|block|C_{max}$, envolve a determinação da ordem de processamento de *n* tarefas em *m* máquinas.

Segundo Framinan, Gupta e Leisten (2004) nestes últimos anos vários métodos heurísticos foram desenvolvidos baseados em diferentes aproximações e como cada um era de uma natureza, cada um apresentava uma característica única, tal como complexidade, tempo de computação, ou requisitos de memória.

A ideia básica dos métodos heurísticos é dividir as tarefas a serem feitas em grupos menores e resolvê-los para achar uma solução mais exata, após isso, esses pequenos grupos são agrupados novamente para achar uma aproximação a solução inicial do problema.

Os métodos heurísticos são compostos por 3 fases, são elas (FRAMINAN; GUPTA e LEISTEN, 2004):

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

Fase I

O Desenvolvimento do índice segundo Framinan, Gupta e Leisten (2004) é a fase na qual, as tarefas são arranjadas geralmente de acordo com o tempo de processamento de cada tarefa em cada máquina. O resultado desta fase é uma sequência das tarefas que pode ser usado como uma solução inicial para as próximas fases, ou então usado como solução do problema.

Fase II

Nesta fase é construída uma solução de uma maneira que é inserida em uma sequência parcial, em uma ou mais posições, as tarefas ainda não sequenciadas, de forma a criar um sequenciamento completo (FRAMINAN, 2004). De tal maneira, esta fase consiste em várias repetições desta ação.

Esta fase é dividida em duas etapas:

- Seleção das tarefas, isto é, seleção das tarefas que serão inseridas em uma sequência parcial;
- Inserção das tarefas, isto é, onde inserir as tarefas selecionadas na sequência parcial.

A seleção das tarefas pode ser de acordo com qualquer índice selecionado a partir da Fase I, ou por meio da tentativa de inseri-la para obter o melhor resultado.

Após selecionada a tarefa, a próxima etapa é a inserção desta tarefa na sequência parcial, afim de decidir sua melhor posição e obter a melhor sequência como resultado desta fase.

Fase III

Nesta última fase uma solução existente é melhorada por meio de alguma meta-heurística, ou seja, a partir de alguma solução inicial o processo procura mudar a sequência das tarefas, para melhorar a solução (FRAMINAN; GUPTA e LEISTEN, 2004). Dessa forma, a solução encontrada nesta fase sempre será melhor ou igual à solução inicial.

São exemplos de métodos de terceira fase:

- ***Simulated annealing***: É uma meta-heurística de otimização que usa da analogia de um metal se resfriando lentamente, em busca de encontrar um estado de menor energia e busca aplicar isso num sistema qualquer (BERTSIMAS; TSITSIKLIS,1993);
- **Busca Tabu**: É uma meta-heurística que guia um algoritmo na exploração dentro de um espaçkko de tempo. A partir de uma solução inicial tenta avançar para uma solução otimizada até que o critério de parada seja satisfeito (PUREZA; MORABITO, 2003);
- **Algoritmo Genético**: É um algoritmo de otimização global, baseado nos mecanismos de seleção natural e da genética, no qual acontece várias iterações, e durante cada uma delas, os princípios de seleção e reprodução são aplicados a uma população de candidatos que pode variar (LUCAS, 2002).

2.1.2. Métodos Exatos

Métodos exatos segundo Kharbeche (2011), tem resolvido um vasto leque de problemas por meio de métodos de programação dinâmica (*DP*[1]) e programação linear (*LP*[2]). A maior vantagem destes métodos é a qualidade encontrada na solução deles, pois diferente dos métodos heurísticos, é garantido uma sequência ideal a partir destas formulações.

---

[1] Dynamic Programming
[2] Linear Programming

Embora este método tenha uma ótima qualidade nas suas soluções, isto vem a um preço: eles não podem ser resolvidos de maneira polinomial, ou seja, seu tempo computacional é alto, e sua dificuldade de programar é elevada (KHARBECHE, 2011). Uma solução encontrada para problemas considerados *NP-hard*, isto é, problemas com alto tempo computacional, é o método chamado *Branch-and-Bound*.

Branch-and-Bound

No algoritmo de *Branch-and-Bound* cada nó da árvore corresponde á um subproblema, que é definido por uma sub-sequência de tarefas. Cada subsequência recebe o nome de sequência parcial, *PS*, e a sequência de tarefas fora de uma *PS* é chamado *NPS*. Quando um nó é ramificado um ou mais nós podem ser gerados ao se adicionar uma ou mais tarefas à sequência parcial associada ao nó que foi ramificado. A seleção do próximo nó a ser ramificado é a partir da regra no qual, o nó escolhido será o que tiver mais tarefas na sequência parcial. Em caso de empates o algoritmo seleciona um nó com o menor limitante inferior. O limitante inferior é calculado no *makespan* para cada nó gerado (RONCONI, 2005). Ao final é verificado qual a sequência parcial com o menor valor de limitante inferior na última camada, se nenhuma outra sequência parcial possuir limitante inferior menor que o selecionado, pode-se afirmar que a sequência escolhida é a sequência ótima para o problema, obtendo assim a sequência que melhor sequencia as suas tarefas com o menor tempo de *makespan*.

O método *Branch-and-Bound* segundo Kharbeche (2011) é um método amplamente usado para resolver difíceis problemas de otimização combinatória, conforme Apêndice G. Tipicamente o método *Branch-and-Bound* é caracterizado por dois procedimentos fundamentais (KHARBECHE, 2011):

- Ramificação (*branching*): Processo de divisão de um problema em um ou mais subproblemas menores.
- Limitação (*bounding*): Processo de cálculo de um limitante (inferior e/ou superior) para o critério de avaliação utilizado.

O procedimento da ramificação substitui o problema inicial por um conjunto de novos problemas menores que o original. Este método é usado para achar a solução mais precisa, sistematicamente analisando as sub-sequências das soluções viáveis, ou seja, por meio de uma árvore solução com nós que representam essas sub-sequências, conforme Figura 1.

Nesta árvore pode-se observar suas camadas, que representam, cada uma delas, uma sequência parcial (sub-sequência). Em cada sub-sequência, é calculado um limitante inferior (*LB*[3]) que representa o menor tempo do *makespan* daquela sub-sequência.

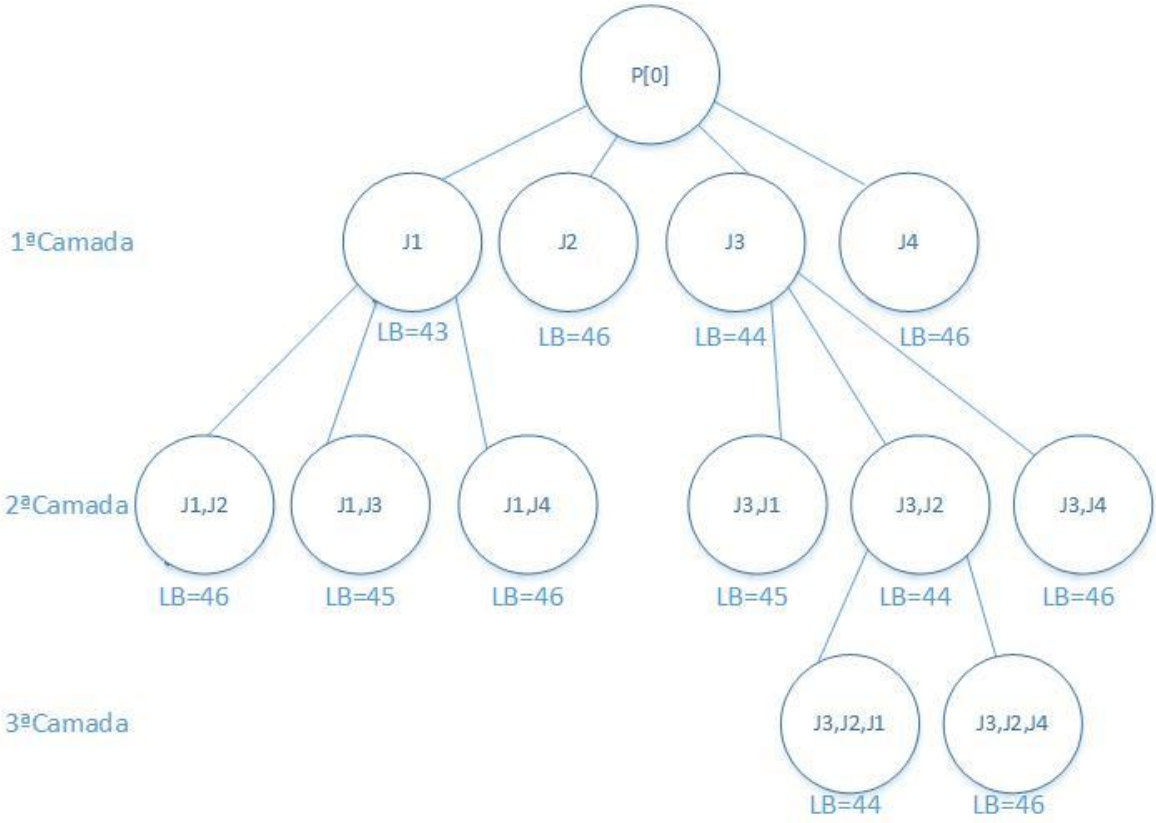

Figura 1 – Árvore de soluções (Adaptado de IGNALL e SCHRAGE, 1964).

[3] *Lower Bound*

No exemplo a seguir considera-se um problema do tipo $F_3|prmu|C_{max}$, o qual possuí 4 tarefas e 3 máquinas. O primeiro passo para resolver este problema é o cálculo do limitante inferior de cada sub-sequência da primeira camada, conforme apresentado na Figura 2.

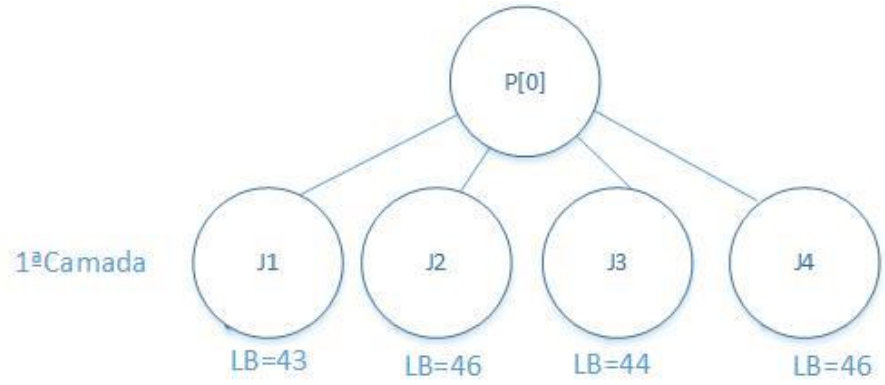

Figura 2 – Primeira camada e suas ramificações.

Como resultado desta ramificação, verifica-se que o limitante inferior da sub-sequência $\{J_1\}$ foi o menor dentre todas sub-sequências da primeira camada, por isso, ramifica-se a sub-sequência $\{J_1\}$, conforme Figura 3:

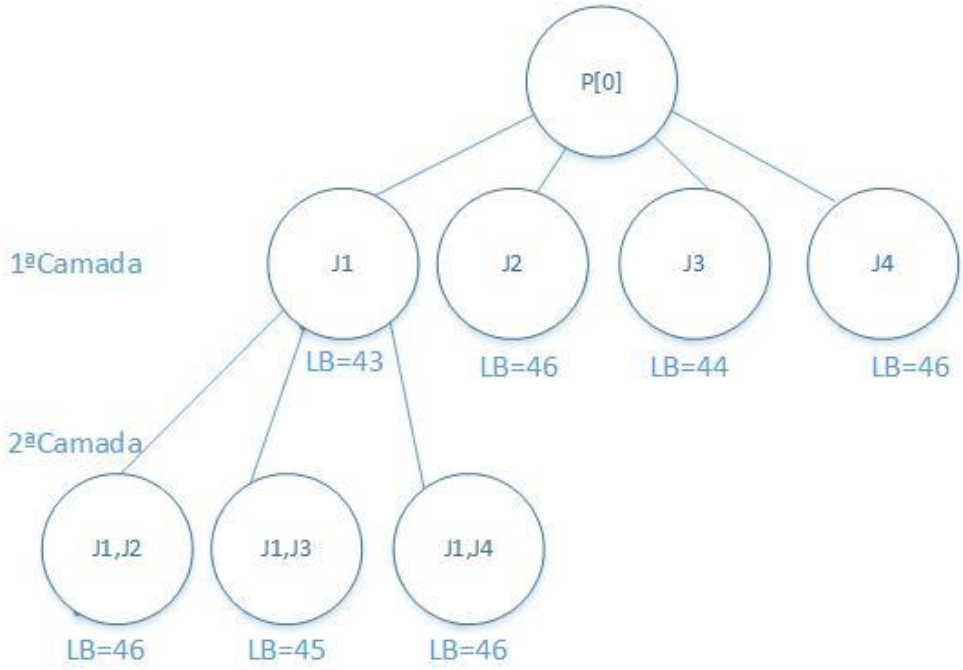

Figura 3 – Segunda camada e ramificações do exemplo.

Como a sequência parcial $\{J_3\}$, na primeira camada, possui um limitante inferior menor que o menor limitante inferior da segunda camada, é necessário

ramifica-la. Sendo assim, a segunda camada terá seis sequências parciais, conforme Figura 4:

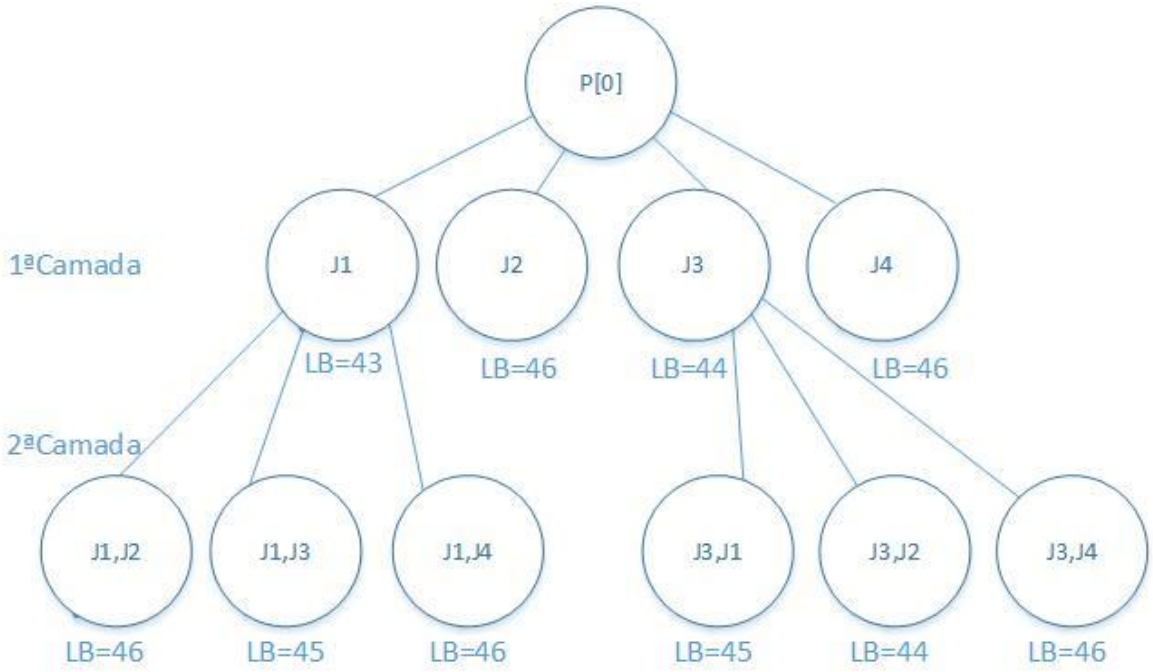

Figura 4 – Segunda camada com duas ramificações.

Como o menor limitante inferior foi o da sequência parcial $\{J_3,J_2\}$, ela será ramificada para comparar qual o melhor caminho a seguir:

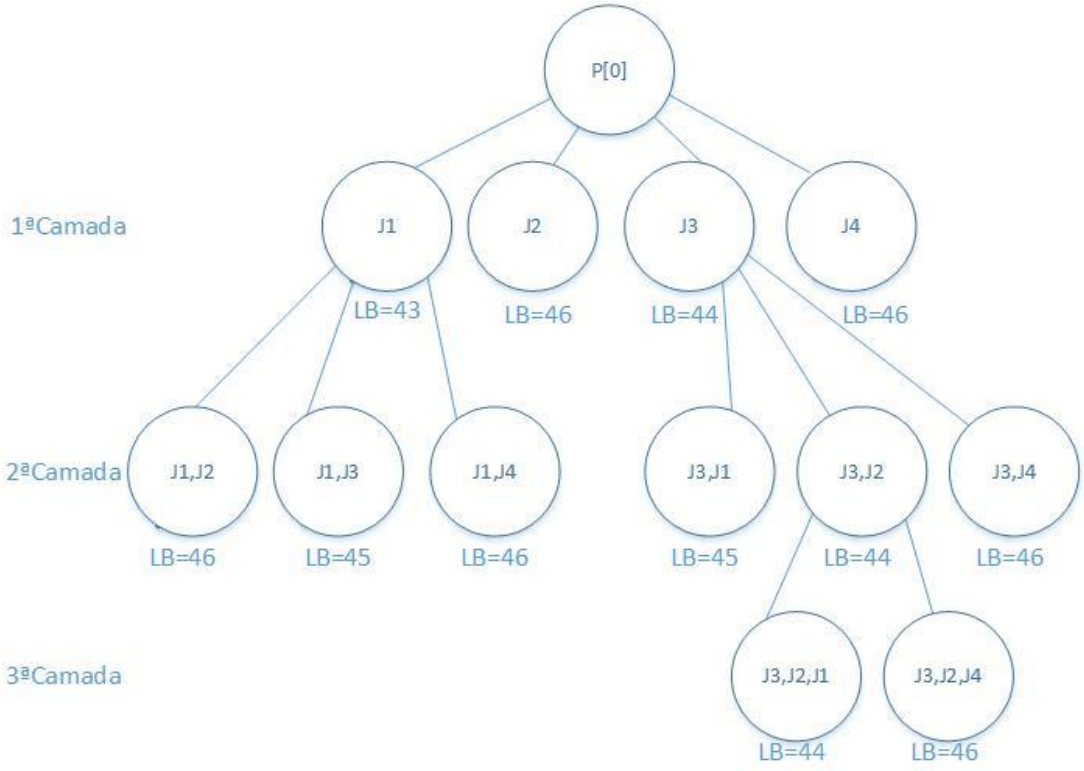

Figura 5 – Terceira camada com sua ramificação.

Como nenhuma outra sequência parcial possui um limitante inferior menor que o menor limitante inferior encontrado na última camada, pode-se afirmar que a sequência ótima para o problema é a sequência {J_3,J_2,J_1,J_4}, com um limitante inferior, no valor de LB=44. Obtendo assim a sequência que melhor distribuí suas tarefas com o menor tempo de *makespan*.

Limitante Inferior

O *makespan* de um limitante inferior é obtido calculando um limitante inferior no tempo de término da última tarefa de *NPS* na última máquina, conforme Apêndice B. O limitante inferior escolhido para o algoritmo de *Branch-and-Bound* foi criado por Ronconi (2005). Ronconi (2005) considera uma sequência arbritária de tarefas em NPS com |NPS| tarefas. O tempo de término da última tarefa na máquina *m* é maior ou igual á LB2, onde:

$$L2(k) = D_{|\mathrm{PS}|,\mathrm{k}} + \sum_{g=1}^{|\mathrm{NPS}|} \max\,(\mathrm{a}_{\mathrm{k}}^{\mathrm{g}}, b_{k+1}^{g}) + \sum_{q=k+1}^{m} p_{\min_q} \quad (1)$$

$$LB2 = \max_{1 \le k \le m} \{L2(k)\} \quad (2)$$

$p_{\min_q}$ é o menor tempo de processamento na máquina k dentre todas as tarefas em NPS;

$\mathrm{a}_{\mathrm{k}}^{\mathrm{g}}$ é o g-menor tempo de processamento na máquina k dentre as tarefas em NPS;

b_{k+1}^{g} é o g menor valor entre $D_{|PS|,k+1}$- $D_{|\mathrm{PS}|,\mathrm{k}}$ e o tempo de processamento das tarefas em NPS na máquina k+1 sem $p_{\min_{k+1}}$;

b_{m+1}^{g}= 0 para todo g.

2.2. APLICAÇÃO DA SOLUÇÃO INICIAL

Pode-se aplicar o uso de uma solução inicial, ao exemplo anterior, para se verificar se o número de nós a serem ramificados e o tempo computacional serão menores.

O valor do *makespan* obtido pelo método heurístico será usado como um limitante superior (UB[4]) para se achar a solução ótima. Considerando um método heurístico qualquer que tenha obtido como resultado a sequência $\{J_3,J_2,J_1,J_4\}$ e um *makespan* de UB=44, temos então na primeira camada:

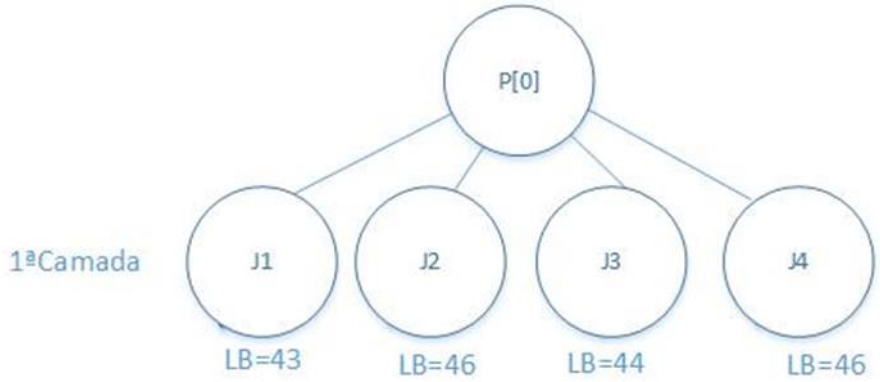

Figura 2 – Primeira camada e suas ramificações.

Como obtemos um UB=44 como solução inicial, será ramificado somente as sequência $\{J_1\}$, pois as demais sequências possuem valores de LB igual ou superior ao limitante superior. Sendo assim obtemos:

[4] *Upper Bound*

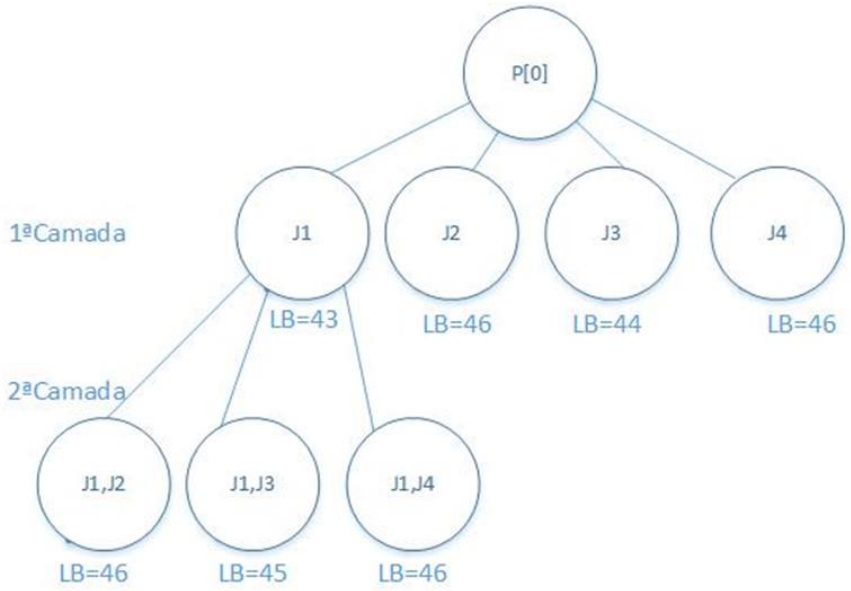

Figura 3 – Segunda camada e ramificações do exemplo.

Com este resultado percebe-se que todas as ramificações de $\{J_1\}$ possuem valores dos limitantes inferiores maiores que o valor de UB obtido pela solução inicial. Como nenhuma outra sequência parcial possui um limitante inferior menor que o menor limitante inferior encontrado na última camada, pode-se afirmar que a sequência ótima para o problema é a sequência $\{J_3,J_2,J_1,J_4\}$, com um limitante inferior no valor de 44. Obtendo assim a sequência que melhor distribuí suas tarefas com o menor tempo de *makespan*.

Com o uso da solução inicial consegue-se alcançar o resultado ótimo com um número menor de ramificações, consequentemente com um tempo computacional menor. Porém, para se calcular o tempo computacional do método *Branch-and-Bound*, o tempo computacional para a execução do método heurístico é somado ao tempo computacional das ramificações. Portanto, caso o tempo de execução do método heurístico seja muito elevado, pode ocasionar até um aumento no tempo computacional para a execução do *Branch-and-Bound*.

Os melhores métodos heurísticos de I fase para problemas Fm|block|Cmax foram usadas para fornecer e solução inicial para o algoritmo *Branch-and-Bound*. Essas são *MinMax* (MM), *Profile fitting* (PF), Weighted Profile Fitting (WPF) e PW.

No algoritmo *MinMax* (MM), proposto por Ronconi (2004), são usados três regras: 1. Definir a primeira tarefa com o menor tempo de processamento na

primeira máquina; 2. Definir a última tarefa da sequência a que tem o menor tempo de processamento na última máquina; 3. Para os trabalhos restantes, o trabalho consecutivo (*c*) do trabalho já sequenciados (*i*) é a que recebe o menor resultado na equação 3. Quando α é uma constante usada para ponderar os dois termos da expressão, conforme se observa no Apêndice C.

$$\alpha\sum_{l=1}^{\text{m-1}}\left|P_{[c],l}-P_{[i],l+1}\right|+(1-\alpha)\sum_{k=1}^{m}P_{[c],k} \tag{3}$$

Para o algoritmo *Profile fitting* (PF), Apêndice D, criado por McCormick et al. (1989), duas regras são usadas: 1. Defina a primeira tarefa na sequência, a qual possui a menor soma dos tempos de processamento em todas as máquinas; 2. Em seguida, a equação 4 é usado para calcular o possível tempo ocioso e de bloqueio, previsto pela inserção de cada tarefa que ainda não foi sequenciada na posição *c* da sequência . O trabalho (*j*) que obtiver o menor valor para $\delta_{\text{j,c}}$ é determinado como o próximo trabalho (c+1) na sequência.

$$\delta_{j,c}=\sum_{k=1}^{\text{m}}\left(D_{[c+1],k}-D_{[c],k}-P_{j,k}\right) \tag{4}$$

Tempo ocioso e bloqueios causados por tarefas anteriores e por máquinas anteriores pode ter efeitos maiores sobre o valor makespan do que aqueles causados por trabalhos e máquinas posteriores. Portanto, o algoritmo *Weighted Perfil Fitting* (wPF), proposto por Pan e Wang (2012) , aplica-se um fator de peso (sem) para $\delta_{\text{j,c}}$, o que diferencia o efeito de máquinas em diferentes estágios e empregos em posições diferentes . A primeira tarefa da sequência é a tarefa com a menor soma dos tempos de processamento em todas as máquinas. Em seguida, a tarefa seguinte (c+1) na sequência é a tarefa j com o menor valor de $\delta_{\text{j,c}}$ calculada pela equação 5, conforme Apêndice E .

$$\delta_{j,c}=\sum_{k=1}^{\text{m}}w_k\left(D_{[c+1],k}-D_{[c],k}-P_{j,k}\right) \tag{5}$$

Onde w_i é definido por:

$$w_k=m/\left(k+c\left(m-k\right)/\left(n-2\right)\right) \tag{6}$$

A tarefa selecionada *j* também afeta o tempo ocioso e de bloqueio dos trabalhos restantes na |NPS|. Portanto, Pan e Wang (2012) propôs o algoritmo PW que se encontra no Apêndice F, que tenta minimizar o tempo ocioso e os tempos de bloqueio e os efeitos sobre os horários de início e de término de tarefas posteriores. Equação 5 é utilizada para estimar o tempo ocioso e de bloqueio causados pela indexação da tarefa *j* na posição (c+1) da sequência . Então a equação 7 é usada para estimar os tempos ocioso e bloqueio dos trabalhos restantes na |NPS| :

$$x_{j,c} = \sum_{k=1}^{m} w_k \left(D_{[c+2],k} - D_{[c+1],k} - P_{v,k} \right) \tag{7}$$

Onde:

$P_{v,k}$ é o tempo médio de processamento de todos os tarefas restantes |NPS| , e é calculada pela equação 8;

$D_{[c+2],k}$ é a hora de saída de $P_{v,k}$.

$$P_{v,k} = \sum_{\substack{q \in |NPS| \\ q \neq j}} P_{q,k} / (n - c - 1) \tag{8}$$

Combinando o tempo ocioso estimado e de bloqueio causado por tarefas e *j* e *v*, é obtido $f_{j,c}$, calculada pela equação 9. (n-c-2) é usado para equilibrar os tempo ocioso e de bloqueio causados por tarefas *j* e os efeitos de tarefas *j* em trabalhos posteriores. A primeira tarefa na sequência é a *j* tarefa com o menor valor para $f_{j,0}$. Então, para as posições restantes da sequência, a tarefa *j* que obtiver o menor valor para $f_{j,c}$ é a próxima tarefa na sequência (c+1).

$$f_{j,c} = (n - c - 2)\delta_{j,c} + x_{j,c} \tag{9}$$

2.3. MÉTODOS DE SOLUÇÃO NA LITERATURA

Companys e Mateo (2005) propuseram um algoritmo de *Branch-and-Bound* melhorado, para resolver esses problemas e heurísticas auxiliares, para

obter uma boa solução inicial em problemas do tipo $F_m|prmu|C_{max}$ e $F_m|block|C_{max}$, no qual, o termo *prmu* refere-se ao ambiente permutacional. As heurísticas auxiliares propostas são construídas por dois passos: no primeiro passo uma permutação é obtida, e no segundo passo um procedimento de busca local é aplicado. Companys e Mateo (2005) concluíram que a diferença de comportamento entre os problemas $F_m|prmu|C_{max}$ e $F_m|block|C_{max}$, quando ambos são submetidos a heurísticas, mostra que o tempo computacional exigido pelo problema do tipo $F_m|block|C_{max}$ é maior que no permutacional. Companys e Mateo (2005) propõe também um algoritmo de LOMPEN, que tem como idéia principal a habilidade de rodar simultaneamente dois algoritmos de *Branch-and-Bound*. Eles obtiveram um ótimo resultado, resolvendo 8 problemas não solucionados pelos outros métodos.

Ronconi e Armentano (2001) apresentam um algoritmo de *Branch-and-Bound* para minimizar o atraso total em ambientes *flow-shop* com bloqueio. Eles propõe um limitante inferior para este critério, e ainda limitantes inferiores para o *makespan* e para o tempo de fluxo. Ronconi e Armentano (2001) concluíram que os testes pelo método *Branch-and-Bound* por eles realizados, obtiveram resultados satisfatórios, diminuindo consideravelmente os números de nós a serem computados.

Ronconi (2005) propõe um algoritmo que explora a ocorrência de bloqueio a fim de minimizar o tempo de *makespan*. Sua conclusão em experimentos numéricos mostraram que o esquema de limitação, funciona melhor que o limitante inferior proposto por Ronconi e Armentano (2001). Como extensão de seu trabalho Ronconi (2005) propõe o desenvolvimento de uma regra de dominância.

Kharbeche (2011) propõe o estudo de um método exato para o problema por ele chamado, Problema Celular Robótico (PCR), que é um caso de *flow-shop* genérico. Kharbeche (2011) explica seu novo método matemático de programação, que é usado computacionalmente para resolver problemas de tamanho pequeno, e em seguida, explica e apresenta seu algoritmo de *Branch-and-Bound* e comenta que o modelo de *Branch-and-Bound* pode resolver problemas maiores, pois requer menor tempo de processamento. Kharbeche (2011) teve como resultado que o seu método matemático falhou em resolver

55% dos problemas propostos, enquanto o método de *Branch-and-Bound* falhou em resolver 35% deles.

McCormick, Pinedo e Shenker (1989) criaram um algoritmo nomeado *Profile Fitting (PF)*, o qual é um algoritmo para ambientes com tamanho finito de *buffer*, ocorrendo bloqueio quando estes *buffers* estiverem cheios. Este algoritmo adota como primeira tarefa na sequência de processamento, aquela na qual a soma do tempo de processamento entre todas as máquinas for a menor. O resultado obtidos através de 5 testes realizados entre duas variações do *PF* revela que os dois algoritmos tiveram bons resultados para os testes desenvolvidos, sendo qualificado como um dos melhores métodos heurísticos já programados.

Pan and Wang (2012) adaptou o algoritmo *PF* de (McCormick *et al., 1989*) criando uma forma melhorada do mesmo nomeado de *PW* e o algoritmo *Weighted Profile Fitting (wPF)* que segue as mesmas regras que o método *PF*, com a diferença que existe um fator peso w_i que pondera os termos da ordem de sequenciamento. Os resultados mostraram que ambos *wPF* e *PW*, produziram resultados muitos melhores que os métodos heurísticos *MM* e *PF* já existentes, as técnicas de inserção no sequenciamento por eles introduzidos, resultou em uma melhora de 1,5% no tempo de processamento de CPU.

Ronconi (2004) analisou a minimização do *makespan* para um problema em ambiente *flow-shop* com bloqueio. Foi proposto um algoritmo denominado *MinMax (MM)* e comparado ao *PF* considerado o melhor algoritmo na literatura. Os resultados mostraram que embora o *MM* obteve bons resultados, ele foi superado pelo *PF*, o qual mostrou potencial para problemas grandes.

3. DESENVOLVIMENTO

O modelo *Branch-and-Bound*, embora viável de ser usado para o cálculo de *makespan*, pode consumir um tempo computacional elevado quando se trata de problemas *NP-HARD*. Uma saída para estes casos é a implementação de uma solução inicial para o problema, que pode gerar um possível ganho no tempo computacional do mesmo, diminuindo o número de ramificações necessárias.

A solução inicial fornecida pelos métodos heurísticos foi aplicada aliada ao algoritmo *Branch-and-Bound* como uma forma de se alcançar o resultado de uma forma mais rápida, isso significa, diminuir o número de ramificações por meio de um resultado prévio fornecido pelo método heurístico. Este resultado diminui a árvore de possibilidades do algoritmo *Branch-and-Bound*, logo, diminuindo o número de nós a serem ramificadas, por meio da adoção de um limitante superior igual à solução inicial. Os modelos do algoritmo *Branch-and-Bound* associado aos métodos de solução inicial são apresentados nos apêndices H, I, J e K.

Neste trabalho além de se avaliar se o uso de uma solução inicial como um método eficiente de diminuir o tempo computacional e o número de ramificações, será avaliado qual, dentre os métodos heurísticos selecionados, apresenta a melhor porcentagem de melhora ao algoritmo.

O algoritmo *Branch-and-Bound* utilizado, bem como o limitante inferior foram propostos por Ronconi (2005).

A fim de avaliar se o uso ou não de uma solução inicial é favorável a um tempo de processamento mais baixo foi comparado os programas para ver qual tinha o menor tempo de processamento e o menor número de nós. Para determinar qual o melhor algoritmo foi utilizado como parâmetro o desvio relativo, a equação (10), que mede a distância da solução encontrada por este algoritmo (DM) ao melhor resultado obtido (DM*) de todos os algoritmos . Neste trabalho foram utilizados os desvios relativos médios para comparar tempos computacionais e número de nós para encontrar o melhor algoritmo para cada classe de problemas. Em primeiro lugar, foi comparado os valores obtidos a partir dos mesmos grupos, como mostrado na Tabela 1. A Tabela 2 mostra o desvio relativo médio de todas as nove classes de problemas para determinar quais,

entre os métodos de comparação, tiveram o menor tempo de processamento e o número mínimo de nós.

$$DR = \frac{DM - DM^*}{DM^*} \tag{10}$$

4. EXPERIMENTAÇÃO COMPUTACIONAL

Neste trabalho foi feita a programação do método *Branch-and-Bound* utilizando o limitante inferior proposto por (RONCONI, 2005). Em seguida, os melhores métodos heurísticos de I Fase do tipo $F_m|block|C_{max}$ foram utilizados como forma de obtenção da solução inicial para o *Branch-and-Bound*. Os métodos heurísticos utilizados foram: *MM* (Ronconi,2004); *PF* (McCormick *et al*., 1989); *PW* e *wPF* (Pan e Wang, 2012).

Os testes foram realizados no *software* MatLab© (R2014b) em um computador com processador Intel® Core™ i5-4460 com 3.2GHz, 8 GB de memória RAM e com sistema operacional Windows 7. O algoritmo *Branch-and-Bound* foi aplicado a 180 problemas, propostos por Ronconi (2005), divididos em nove grupos. Os grupos variam em quantidade de tarefas (*n*) e máquinas (*m*), sendo eles matrizes *nxm*: 10x2, 10x5, 10x10, 12x2, 12x5, 12x10, 14x2, 14x5 e 14x10. Não pode ser feita a análise da base completa de dados fornecida por Ronconi (2005) devido a questão de tempo, porém sugere-se como extensão deste trabalho o uso da base de dados completa com os 540 problemas sugeridos por Ronconi (2005).

Os resultados obtidos do desvio relativo médio do tempo computacional e do número de nós, para todos os algoritmos e classes de problemas foram tabulados no *software* Microsoft Excel.

Tabela 1 - Desvio relativo médio dos tempos de CPU e número de nós de todas as classes de problema (continua).

	10x2		10x5		10x10	
Algoritmos	Tempo de CPU (%)	Número de nós (%)	Tempo de CPU (%)	Número de nós (%)	Tempo de CPU (%)	Número de nós (%)
wPF	52340,36%	58607,50%	2,91%	0,87%	4,43%	2,50%
PF	50471,35%	58607,50%	1,91%	1,04%	3,29%	2,81%
PW	50582,37%	58607,50%	1,16%	0,09%	0,47%	0,38%
MM	4,53%	0,00%	1,63%	0,97%	2,80%	2,41%
Clássico	50739,78%	58607,50%	2,18%	1,14%	3,68%	2,98%

Tabela 1 - Desvio relativo médio dos tempos de CPU e número de nós de todas as classes de problema (continuação).

	12x2		12x5		12x10	
Algoritmos	Tempo de CPU (%)	Número de nós (%)	Tempo de CPU (%)	Número de nós (%)	Tempo de CPU (%)	Número de nós (%)
wPF	6,92%	0,00%	1,15%	0,04%	1,80%	0,90%
PF	3,78%	0,00%	0,37%	0,06%	2,34%	1,83%
PW	3,10%	0,00%	0,43%	0,01%	1,32%	1,23%
MM	1,49%	0,00%	0,19%	0,07%	1,80%	1,51%
Clássico	2,52%	0,00%	0,26%	0,07%	2,15%	1,86%

Tabela 1 - Desvio relativo médio dos tempos de CPU e número de nós de todas as classes de problema (continuação).

	14x2		14x5		14x10	
Algoritmos	Tempo de CPU (%)	Número de nós (%)	Tempo de CPU (%)	Número de nós (%)	Tempo de CPU (%)	Número de nós (%)
wPF	3,78%	0,00%	1,26%	0,00%	7,72%	6,58%
PF	2,43%	0,00%	0,13%	0,02%	9,09%	8,36%
PW	3,27%	0%	0,25%	0,01%	0,16%	0,09%
MM	1,98%	0,00%	0,47%	0,02%	9,32%	8,48%
Clássico	2,74%	0,00%	0,40%	0,02%	9,35%	8,50%

Tabela 2 – Desvio relativo médio dos tempos de CPU e número de nós dos algoritmos *Branch-and-Bound*.

Algoritmos	Tempo de CPU (%)	Número de nós (%)
wPF	5818,93%	6513,15%
PF	5610,52%	6513,51%
PW	5621,39%	6512,15%
MM	2,69%	1,50%
Clássico	5640,34%	6513,56%

Analisando os dados da Tabela 1 pode-se notar que na primeira classe de problemas (com 10 tarefas e 2 máquinas) o método *MM* associado ao método *Branch-and-Bound* superou todos os outros métodos. Os resultados obtidos nesta classe podem mostrar um resultado final falso, pois a primeira classe de problemas obteve um desvio relativo médio fora do padrão devido a alguma interferência. Portanto outra comparação foi feita sem considerar a primeira classe de problemas. A nova comparação é apresentada na Tabela 3.

Tabela 3 – Desvio relativo médio dos tempos de CPU e número de nós dos algoritmos *Branch-and-Bound* sem considerar a primeira classe de problemas.

Algoritmos	Tempo de CPU (%)	Número de nós (%)
wPF	3,75%	1,36%
PF	2,92%	1,76%
PW	1,27%	0,23%
MM	2,46%	1,68%
Clássico	2,91%	1,82%

A partir dos dados apresentados na Tabela 2 pode ser observado que o método *MM* associado ao *Branch-and-Bound* obteve os melhores resultados entre todos os métodos de comparação. No entanto, este resultado pode ter sido influenciado pelos resultados obtidos na primeira classe de problemas (com 10 tarefas e duas máquinas), onde o método resultou em um tempo computacional e número de nós baixo em comparação com os outros métodos. Então, outra comparação foi feita sem considerar a primeira classe de problemas, e neste novo cenário, o método *PW* associado ao *Branch-and-Bound* proporcionou os melhores resultados.

Pode-se notar a partir da Tabela 1 que o método *PW* associado ao *Branch-and-Bound* tende a proporcionar melhores resultados conforme o número de máquinas aumenta. Com menos máquinas o método *MM* associado ao *Branch-and-Bound* parece fornecer melhores resultados do que *PW*. Isto pode ser explicado com a qualidade dos resultados obtidos pelo algoritmo *PW*, o que é melhor do que os fornecidos pelo algoritmo *MM* (Pan e Wang, 2012), porém exige um tempo computacional maior. Em alguns casos, mesmo quando o número de nós ramificados em problemas menores para o método *MM* associado ao *Branch-and-Bound* for maior ou igual, ao número de nós ramificados para o método *PW* associado ao *Branch-and-Bound*, o tempo computacional do método *MM* associado ao *Branch-and-Bound* foi menor. Isto é devido ao tempo de processamento exigido para calcular o limitante inferior para cada nó, que é menor para problemas com poucas máquinas. Assim, para problemas com poucas máquinas, o tempo computacional necessário para resolver a heurística

(que fornece a solução inicial) pode ter mais efeito sobre o tempo computacional do que a redução do número de nós a serem ramificados.

5. CONCLUSÕES E EXTENSÕES

Este trabalho comparou o uso de diferentes métodos para a obtenção de uma solução inicial para o algoritmo *Branch-and-Bound* com o objetivo de minimizar o *makespan* em um ambiente *flow-shop* com *buffer* zero.

Foram apresentados os melhores métodos heurísticos encontrados na literatura e, estes, foram aplicados como forma de obter uma solução inicial para o *Branch-and-Bound*. A título de comparação foram realizados testes computacionais baseados em uma base de dados conhecida fornecida por Ronconi (2005). Os experimentos realizados mostraram que o uso do algoritmo *Branch-and-Bound* obtém uma boa performance para problemas considerados *NP-Hard*. Em termos de tempo computacional e número de nós, o uso de uma solução inicial obteve melhores resultados do que a aplicação clássica de *Branch-and-Bound* em oito das nove classes dos problemas analisados.

Como o tamanho do problema parece afetar os resultados recomenda-se que os testes futuros devam ser executados em um banco de dados mais diversificado. Recomenda-se usar toda a base de dados fornecida pelo Ronconi (2005) para analisar os resultados. À medida que a qualidade e tempo computacional da solução inicial afetaram o tempo de processamento do algoritmo *Branch-and-Bound*, é recomendado o uso de outros métodos heurísticos (fase II ou III) associados com o algoritmo *Branch-and-Bound* para comparar os resultados.

O tempo necessário para calcular o limitante inferior de cada nó tem uma consequência importante para o tempo de processamento, portanto algum outro limitante inferior para o *makespan* pode ser testado para comparação. Também pode ser aplicada uma regra de dominância para reduzir o número de nós que precisam ser ramificados, analisando os ganhos obtidos em termos de tempo de processamento.

6. REFERÊNCIAS BIBLIOGRÁFICAS

PINEDO, M. L. **Scheduling Theory, Algorithms, and Systems**. New York: Springer Science, 2008.

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**. Seville, v. 55, n 12, p. 1243-1255, June 2004.

BERTSIMAS, D.; TSITSIKLIS, J. Simulated Annealing. **Statistical Science**. Jstor, v. 8, n 1, p. 10-15, Mar. 2007. Disponível em:< https://stat.duke.edu/~scs/Courses/Stat376/Papers/TemperAnneal/AnnealStatSci1993.pdf>. Acesso em: 05 nov. 2014.

PUREZA, V.; MORABITO, R. Uma heurística de busca tabu simples para o problema de carregamento de paletes do produtor. **Scielo**. Rio de Janeiro, v. 23, n 2, Aug. 2003.Disponível em:< http://www.scielo.br/scielo.php?pid=S0101-74382003000200007&script=sci_arttext>. Acesso em 05 nov. 2014.

LUCAS, Diogo C. **Algoritmos Genéticos: uma Introdução**. UFRGS. Disponível em: <http://www.inf.ufrgs.br/~alvares/INF01048IA/ApostilaAlgoritmosGeneticos.pdf>. Acesso em: 05 nov. 2014.

KHARBECHE, Mohamed de. **Exact and Heuristic Methods for Variants of the Permutation Flow Shop Problems**. 2011. Tese (Doutorado em Gestão) – Universidade de Tunis, Tunis, 2011.

IGNALL, E.; SCHRAGE, L. **Application of the Branch-and-Bound technique to some flow-shop scheduling problems.** Cornell University. New York, 1964.

RONCONI, D. P. A Branch-and-Bound Algorithm to Minimize the Makespan in a Flowshop with Blocking. **Annals of Operations Research**. São Paulo. v.138, p. 53-65. 2005.

RONCONI D.P.; ARMENTANO V.A. Lower boundings schemes for flow-shops with blocking in-process. . **Journal of the Operational Research Society**. São Paulo. v. 52. n 11. p. 1289-1297. 2001.

COMPANYS R.; MATEO M. Different behavior of a double branch-and-bound algorithm on Fm|prmu|C_{max} and Fm|block|C_{max} problems. **Computers & Operations Research.** Catalunya. v. 34. p.938-953. July 2005.

MARTINEZ S.; DAUZÈRE-PÉRÈZ S.; GUÉRET C.; MATI Y.; SAUER N. Complexity of flowshop scheduling problems with a new blocking constraint. **European Journal of Operational Research.** v. 169. p. 855-864. March 2005.

PAN; WANG (2012). “Effective Heuristics for the Blocking Flowshop Scheduling Problem with Makespan Minimization.” ***Omega*** 40, 218-229.

MCCORMICK (1989). “Sequencing in an Assembly Line with Blocking to Minimize Cy.” ***Operations Research*** 37, 925-935.

APÊNDICE A – Algoritmo Lowerbound

```
function [LB]=Lowerbound(Pjk,PS,NPS)

LBk=zeros(1,size(Pjk,2));
[D,~]=makespanblock(Pjk,PS);

for k=1:size(Pjk,2)-1
    a=sort(Pjk(NPS,k));
    b=sort(Pjk(NPS,k+1));
    b(1)=[];
    b=sort([b;D(end,k+1)-D(end,k)]);
    LBk(k)=D(end,k)+sum(max(a,b))+sum(min(Pjk(NPS,k+1:end),[],1));
end
LBk(end)=D(end,end)+sum(Pjk(NPS,end));

LB=max(LBk);
```

APÊNDICE B – Algoritmo do Cálculo do *makespan*

```
function [D,makespan]=makespanblock(Pjk,ordem)

n=length(ordem);
m=size(Pjk,2);

D=zeros(n,m);
D(1,1)=Pjk(ordem(1),1);
for k=2:m
    D(1,k)=D(1,k-1)+Pjk(ordem(1),k);
end
if n>1
    for j=2:n
        D(j,1)=max(D(j-1,1)+Pjk(ordem(j),1),D(j-1,2));
        for k=2:m-1
            D(j,k)=max(D(j,k-1)+Pjk(ordem(j),k),D(j-1,k+1));
        end
        D(j,m)=D(j,m-1)+Pjk(ordem(j),m);
    end
end

makespan=D(end,end);
```

APÊNDICE C – Algoritmo *MM*

```
function [Ordem,Makespan,Tempo_Computacional,D]=MM(Alpha,Pjk)
tic
[n,m]=size(Pjk);
Ordem=zeros(1,n);
PjkV=Pjk;
Ordem(1)=find(PjkV(:,1)==min(PjkV(:,1)),1);
PjkV(Ordem(1),:)=NaN;
Ordem(n)=find(PjkV(:,m)==min(PjkV(:,m)),1);
PjkV(Ordem(n),:)=NaN;
for posicao=2:n-1
    Eq=zeros(n,1);
    for c=1:n
        Eq(c)=(1-Alpha)*(sum(PjkV(c,:)));
        for L=1:m-1
            %Eq(c)=roundn(Eq(c)+Alpha*abs(PjkV(c,L)-Pjk(Ordem(posicao-
1),L+1)),-2);
            Eq(c)=Eq(c)+Alpha*abs(PjkV(c,L)-Pjk(Ordem(posicao-1),L+1));
        end
    end
    Ordem(posicao)=find(Eq==min(Eq),1,'first');
    PjkV(Ordem(posicao),:)=NaN;
end
D=zeros(n,m+1);
D(1,1)=0;
for k=2:m+1
    for q=1:k-1
        D(1,k)=D(1,k)+Pjk(Ordem(1),q);
    end
end
for i=2:n
    D(i,1)=D(i-1,2);
    for k=2:m
        D(i,k)=D(i,k)+max(D(i,k-1)+Pjk(Ordem(i),k-1),D(i-1,k+1));
    end
    D(i,m+1)=D(i,m)+Pjk(Ordem(i),m);
end
```

```
Makespan=D(n,m+1);
Tempo_Computacional=toc;
```

APÊNDICE D – Algoritmo *PF*

```
function [Ordem,Makespan,Tempo_Computacional,D]=PF(Pjk)
tic
[n,m]=size(Pjk);
Ordem=zeros(1,n);
PjkV=Pjk;
SomaP=sum(Pjk,2);
Ordem(1)=find(SomaP==min(SomaP),1,'first');
PjkV(Ordem(1),:)=NaN;
D=zeros(n,m+1);
D(1,1)=0;
for k=2:m+1
    for q=1:k-1
        D(1,k)=D(1,k)+Pjk(Ordem(1),q);
    end
end
for posicao=2:n
    Dt=zeros(n,m+1);
    Delta=zeros(n,1);
    for tarefa=1:n
        Dt(tarefa,1)=D(posicao-1,2);
        for k=2:m
            Dt(tarefa,k)=Dt(tarefa,k)+max(Dt(tarefa,k-1)+PjkV(tarefa,k-
1),D(posicao-1,k+1));
        end
        Dt(tarefa,m+1)=Dt(tarefa,m)+PjkV(tarefa,m);
        for k=2:m+1
            Delta(tarefa)=Delta(tarefa)+Dt(tarefa,k)-D(posicao-1,k)-
Pjk(tarefa,k-1);
        end
    end
    Ordem(posicao)=find(Delta==min(Delta),1,'first');
    D(posicao,:)=Dt(Ordem(posicao),:);
    PjkV(Ordem(posicao),:)=NaN;
end
Makespan=D(n,m+1);
Tempo_Computacional=toc;
```

APÊNDICE E – Algoritmo *wPF*

```
function [Ordem,Makespan,Tempo_Computacional,D]=wPF(Pjk)
tic
[n,m]=size(Pjk);
Ordem=zeros(1,n);
PjkV=Pjk;
SomaP=sum(Pjk,2);
Ordem(1)=find(SomaP==min(SomaP),1,'first');
PjkV(Ordem(1),:)=NaN;
D=zeros(n,m+1);
D(1,1)=0;
for k=2:m+1
    for q=1:k-1
        D(1,k)=D(1,k)+Pjk(Ordem(1),q);
    end
end
for posicao=2:n
    Dt=zeros(n,m+1);
    Delta=zeros(n,1);
    w=zeros(1,m);
    for tarefa=1:n
        Dt(tarefa,1)=D(posicao-1,2);
        for k=2:m
            Dt(tarefa,k)=Dt(tarefa,k)+max(Dt(tarefa,k-1)+PjkV(tarefa,k-
1),D(posicao-1,k+1));
        end
        Dt(tarefa,m+1)=Dt(tarefa,m)+PjkV(tarefa,m);
        for k=2:m+1
            w(k-1)=m/((k-1)+(posicao-1)*(m-(k-1))/(n-2));
            Delta(tarefa)=Delta(tarefa)+w(k-1)*(Dt(tarefa,k)-D(posicao-
1,k)-Pjk(tarefa,k-1));
        end
    end
    Ordem(posicao)=find(Delta==min(Delta),1,'first');
    D(posicao,:)=Dt(Ordem(posicao),:);
    PjkV(Ordem(posicao),:)=NaN;
end
```

```
Makespan=D(n,m+1);
Tempo_Computacional=toc;
```

APÊNDICE F – Algoritmo *PW*

```
function [Ordem,Makespan,Tempo_Computacional,D]=PW(Pjk)
[n,m]=size(Pjk);
Ordem=zeros(1,n);
PjkV=Pjk;
tic
Delta=zeros(n,1);
w=zeros(1,m);
pvk=zeros(n,m);
Dt=zeros(n,m+1);
Dtv=zeros(n,m+1);
X=zeros(1,n);
f=zeros(1,n);

for tarefa=1:n

    for k=1:m
        for q=1:n
            pvk(tarefa,k)=pvk(tarefa,k)+(Pjk(q,k))/(n-1);
        end
        pvk(tarefa,k)=pvk(tarefa,k)-Pjk(tarefa,k)/(n-1);
    end

    for k=2:m+1
        Dt(tarefa,k)=Dt(tarefa,k-1)+Pjk(tarefa,k-1);
        w(k-1)=m/(k-1);

        Delta(tarefa)=Delta(tarefa)+w(k-1)*(Dt(tarefa,k)-Pjk(tarefa,k-
1));
    end

    Dtv(tarefa,1)=Dt(tarefa,2);

    for k=2:m

        Dtv(tarefa,k)=max(Dtv(tarefa,k-1)+pvk(tarefa,k-
1),Dt(tarefa,k+1));
```

```
         X(tarefa)=X(tarefa)+w(k-1)*(Dtv(tarefa,k)-Dt(tarefa,k)-
pvk(tarefa,k-1));
    end

    Dtv(tarefa,m+1)=Dtv(tarefa,m)+pvk(tarefa,m);
    X(tarefa)=X(tarefa)+w(m)*(Dtv(tarefa,m+1)-Dt(tarefa,m+1)-
pvk(tarefa,m));
    f(tarefa)=(n-2)*Delta(tarefa)+X(tarefa);
end
Ordem(1)=find(X==min(X(f==min(f))),1,'first');
PjkV(Ordem(1),:)=NaN;

D=zeros(n,m+1);
D(1,:)=Dt(Ordem(1),:);

for posicao=2:n-1
    Delta=zeros(n,1);
    w=zeros(1,m);
    pvk=zeros(n,m);
    Dt=zeros(n,m+1);
    Dtv=zeros(n,m+1);
    X=zeros(1,n);
    f=zeros(1,n);

    for tarefa=1:n

        for k=1:m
            for q=1:n
                pvk(tarefa,k)=pvk(tarefa,k)+(Pjk(q,k))/(n-(posicao-1)-1);
            end
            pvk(tarefa,k)=pvk(tarefa,k)-Pjk(tarefa,k)/(n-(posicao-1)-1);
            for ordenadas=1:posicao-1
                pvk(tarefa,k)=pvk(tarefa,k)-Pjk(Ordem(ordenadas),k)/(n-
(posicao-1)-1);
            end
        end

        Dt(tarefa,1)=D(posicao-1,2);
        for k=2:m
            Dt(tarefa,k)=Dt(tarefa,k)+max(Dt(tarefa,k-1)+PjkV(tarefa,k-
1),D(posicao-1,k+1));
```

```
        end
        Dt(tarefa,m+1)=Dt(tarefa,m)+PjkV(tarefa,m);

        Dtv(tarefa,1)=Dt(tarefa,2);
        for k=2:m
            Dtv(tarefa,k)=Dtv(tarefa,k)+max(Dtv(tarefa,k-1)+pvk(tarefa,k-
1),Dt(tarefa,k+1));
        end
        Dtv(tarefa,m+1)=Dtv(tarefa,m)+pvk(tarefa,m);

        for k=2:m+1
            w(k-1)=m/((k-1)+(posicao-1)*(m-(k-1))/(n-2));
            Delta(tarefa)=Delta(tarefa)+w(k-1)*(Dt(tarefa,k)-D(posicao-
1,k)-Pjk(tarefa,k-1));
            X(tarefa)=X(tarefa)+w(k-1)*(Dtv(tarefa,k)-Dt(tarefa,k)-
pvk(tarefa,k-1));
        end
        f(tarefa)=(n-(posicao-1)-2)*Delta(tarefa)+X(tarefa);
    end
    if sum(f==min(f))>1
        posicao
    end
    Ordem(posicao)=find(X==min(X(f==min(f))),1,'first');
    D(posicao,:)=Dt(Ordem(posicao),:);
    PjkV(Ordem(posicao),:)=NaN;
end
Ordem(n)=find(sum(PjkV,2)==min(sum(PjkV,2)));
D(n,1)=D(n-1,2);
for k=2:m
    D(n,k)=D(n,k)+max(D(n,k-1)+PjkV(Ordem(n),k-1),D(n-1,k+1));
end
D(n,m+1)=D(n,m)+PjkV(Ordem(n),m);

Makespan=D(n,m+1);
Tempo_Computacional=toc;
```

APÊNDICE G – Algoritmo *Branch-and-Bound*

```
function
[ordem,makespan,ramificacoes,tempo_computacional,D]=Branchandbound(Pjk)
tic
n=size(Pjk,1);
PS=[(1:n)',zeros(n,n-2)];
NPS=cumsum([2:n;-eye(n-1)]);
LB=zeros(n,1);
for j=1:n
    [LB(j)]=Lowerbound(Pjk,PS(j,1),NPS(j,:));
end
nodes=LB==min(LB);
nos=PS(nodes,:);
LB(nodes)=[];
PS(nodes,:)=[];
LBhist=LB;
PShist=PS;
ramificacoes=n;
UB=100000000;
loop=0;
while ~isempty(nos)

    Camada=sum(nos>0,2)+1;
    Novos=(n-Camada(1)+1);
    PS=zeros(sum(nodes)*Novos,n-1);

    for no=1:size(nos,1)
        aux=1:n;
        aux(ismember(aux,nos(no,:)))=[];

        PS((no-1)*Novos+1:no*Novos,:)=cumsum([nos(no,:);zeros(Novos-1,n-
1)]);
        PS((no-1)*Novos+1:no*Novos,Camada(1))=aux;
    end

    LB=zeros(size(PS,1),1);
```

```
    for j=1:size(PS,1)
        ramificacoes=ramificacoes+1;
        NPS=1:n;
        NPS(ismember(NPS,PS(j,:)))=[];
        [LB(j)]=Lowerbound(Pjk,PS(j,1:Camada(1)),NPS);
    end
    LBhist(end+1:end+size(PS,1))=LB;
    PShist(end+1:end+size(PS,1),:)=PS;
    if max(Camada)==n-1
        if min(LB)<UB
            UB=min(LB);
            best=PS(find(LB==min(LB),1,'first'),:);
        end
        loop=1;
    end

    if loop==1
        Cortar=LBhist>=UB;
        LBhist(Cortar)=[];
        PShist(Cortar,:)=[];

        Ult_Camada=max(sum(PShist>0,2));

nodes=LBhist(sum(PShist>0,2)==Ult_Camada)==min(LBhist(sum(PShist>0,2)==Ul
t_Camada));
        nos=PShist(sum(PShist>0,2)==Ult_Camada,:);
        nos=nos(nodes,:);

        LBhist(ismember(PShist,nos,'rows'))=[];
        PShist(ismember(PShist,nos,'rows'),:)=[];
    else
        nodes=LB==min(LB);
        nos=PS(nodes,:);
        LBhist(ismember(PShist,nos,'rows'))=[];
        PShist(ismember(PShist,nos,'rows'),:)=[];
    end

end

ultima_tarefa=sum(1:n)-sum(best);
ordem=[best,ultima_tarefa];
```

```
[D,makespan]=makespanblock(Pjk,ordem);
tempo_computacional=toc;
```

ALGORITMO H – Solução inicial *MM*

```
function
[ordem,ramificacoes,makespan,tempo_computacional,D]=SolucaoinicialMM(Pjk)
tic
[Ordem,UB,~,~]=MM(0.6,Pjk);
n=size(Pjk,1);
Ordem(n)=[];
best=Ordem;
PS=[(1:n)',zeros(n,n-2)];
NPS=cumsum([2:n;-eye(n-1)]);
LB=zeros(n,1);
for j=1:n
    [LB(j)]=Lowerbound(Pjk,PS(j,1),NPS(j,:));
end
LBhist=LB;
PShist=PS;

    Cortar=LBhist>=UB;
    LBhist(Cortar)=[];
    PShist(Cortar,:)=[];
nodes=LBhist==min(LBhist);
nos=PShist(nodes,:);
LBhist(nodes)=[];
PShist(nodes,:)=[];

ramificacoes=n;
while ~isempty(nos)

    Camada=sum(nos>0,2)+1;
    Novos=(n-Camada(1)+1);
    PS=zeros(sum(nodes)*Novos,n-1);

    for no=1:size(nos,1)
        aux=1:n;
        aux(ismember(aux,nos(no,:)))=[];
```

```
        PS((no-1)*Novos+1:no*Novos,:)=cumsum([nos(no,:);zeros(Novos-1,n-
1)]);
        PS((no-1)*Novos+1:no*Novos,Camada(1))=aux;
    end

    LB=zeros(size(PS,1),1);

    for j=1:size(PS,1)
        ramificacoes=ramificacoes+1;
        NPS=1:n;
        NPS(ismember(NPS,PS(j,:)))=[];
        [LB(j)]=Lowerbound(Pjk,PS(j,1:Camada(1)),NPS);
    end
    LBhist(end+1:end+size(PS,1))=LB;
    PShist(end+1:end+size(PS,1),:)=PS;

    if max(Camada)==n-1
        if min(LB)<UB
            UB=min(LB);
            best=PS(find(LB==min(LB),1,'first'),:);
        end
    end

        Cortar=LBhist>=UB;
        LBhist(Cortar)=[];
        PShist(Cortar,:)=[];

        Ult_Camada=max(sum(PShist>0,2));

nodes=LBhist(sum(PShist>0,2)==Ult_Camada)==min(LBhist(sum(PShist>0,2)==Ul
t_Camada));
        nos=PShist(sum(PShist>0,2)==Ult_Camada,:);
        nos=nos(nodes,:);

        LBhist(ismember(PShist,nos,'rows'))=[];
        PShist(ismember(PShist,nos,'rows'),:)=[];

end

ultima_tarefa=sum(1:n)-sum(best);
ordem=[best,ultima_tarefa];
```

```
[D,makespan]=makespanblock(Pjk,ordem);
tempo_computacional=toc;
```

ALGORITMO I – Solução inicial *PF*

```
function
[ordem,ramificacoes,makespan,tempo_computacional,D]=SolucaoinicialPF(Pjk)
tic
[Ordem,UB,~,~]=PF(Pjk);
n=size(Pjk,1);
Ordem(n)=[];
best=Ordem;
PS=[(1:n)',zeros(n,n-2)];
NPS=cumsum([2:n;-eye(n-1)]);
LB=zeros(n,1);
for j=1:n
    [LB(j)]=Lowerbound(Pjk,PS(j,1),NPS(j,:));
end

LBhist=LB;
PShist=PS;

    Cortar=LBhist>=UB;
    LBhist(Cortar)=[];
    PShist(Cortar,:)=[];

nodes=LBhist==min(LBhist);
nos=PShist(nodes,:);
LBhist(nodes)=[];
PShist(nodes,:)=[];
ramificacoes=n;
while ~isempty(nos)

    Camada=sum(nos>0,2)+1;
    Novos=(n-Camada(1)+1);
    PS=zeros(sum(nodes)*Novos,n-1);

    for no=1:size(nos,1)
        aux=1:n;
        aux(ismember(aux,nos(no,:)))=[];
```

```
        PS((no-1)*Novos+1:no*Novos,:)=cumsum([nos(no,:);zeros(Novos-1,n-
1)]);
        PS((no-1)*Novos+1:no*Novos,Camada(1))=aux;
    end

    LB=zeros(size(PS,1),1);

    for j=1:size(PS,1)
        ramificacoes=ramificacoes+1;
        NPS=1:n;
        NPS(ismember(NPS,PS(j,:)))=[];
        [LB(j)]=Lowerbound(Pjk,PS(j,1:Camada(1)),NPS);
    end
    LBhist(end+1:end+size(PS,1))=LB;
    PShist(end+1:end+size(PS,1),:)=PS;
    if max(Camada)==n-1
        if min(LB)<UB
            UB=min(LB);
            best=PS(find(LB==min(LB),1,'first'),:);
        end
    end

        Cortar=LBhist>=UB;
        LBhist(Cortar)=[];
        PShist(Cortar,:)=[];

        Ult_Camada=max(sum(PShist>0,2));
nodes=LBhist(sum(PShist>0,2)==Ult_Camada)==min(LBhist(sum(PShist>0,2)==Ul
t_Camada));
        nos=PShist(sum(PShist>0,2)==Ult_Camada,:);
        nos=nos(nodes,:);

        LBhist(ismember(PShist,nos,'rows'))=[];
        PShist(ismember(PShist,nos,'rows'),:)=[];

end

ultima_tarefa=sum(1:n)-sum(best);
ordem=[best,ultima_tarefa];
[D,makespan]=makespanblock(Pjk,ordem);
tempo_computacional=toc;
```

ALGORITMO J – Solução inicial *wPF*

```
function
[ordem,ramificacoes,makespan,tempo_computacional,D]=SolucaoinicialwPF(Pjk
)
tic
[Ordem,UB,~,~]=wPF(Pjk);
n=size(Pjk,1);
Ordem(n)=[];
best=Ordem;
PS=[(1:n)',zeros(n,n-2)];
NPS=cumsum([2:n;-eye(n-1)]);
LB=zeros(n,1);
for j=1:n
    [LB(j)]=Lowerbound(Pjk,PS(j,1),NPS(j,:));
end
LBhist=LB;
PShist=PS;

    Cortar=LBhist>=UB;
    LBhist(Cortar)=[];
    PShist(Cortar,:)=[];
nodes=LBhist==min(LBhist);
nos=PShist(nodes,:);
LBhist(nodes)=[];
PShist(nodes,:)=[];

ramificacoes=n;
while ~isempty(nos)

    Camada=sum(nos>0,2)+1;
    Novos=(n-Camada(1)+1);
    PS=zeros(sum(nodes)*Novos,n-1);

    for no=1:size(nos,1)
        aux=1:n;
        aux(ismember(aux,nos(no,:)))=[];
```

```
        PS((no-1)*Novos+1:no*Novos,:)=cumsum([nos(no,:);zeros(Novos-1,n-
1)]);
        PS((no-1)*Novos+1:no*Novos,Camada(1))=aux;
    end

    LB=zeros(size(PS,1),1);

    for j=1:size(PS,1)
        ramificacoes=ramificacoes+1;
        NPS=1:n;
        NPS(ismember(NPS,PS(j,:)))=[];
        PS(j,1:Camada(1))
        [LB(j)]=Lowerbound(Pjk,PS(j,1:Camada(1)),NPS);
    end
    LBhist(end+1:end+size(PS,1))=LB;
    PShist(end+1:end+size(PS,1),:)=PS;

    if max(Camada)==n-1
        if min(LB)<UB
            UB=min(LB);
            best=PS(find(LB==min(LB),1,'first'),:);
        end
    end

        Cortar=LBhist>=UB;
        LBhist(Cortar)=[];
        PShist(Cortar,:)=[];

        Ult_Camada=max(sum(PShist>0,2));

nodes=LBhist(sum(PShist>0,2)==Ult_Camada)==min(LBhist(sum(PShist>0,2)==Ul
t_Camada));
        nos=PShist(sum(PShist>0,2)==Ult_Camada,:);
        nos=nos(nodes,:);

        LBhist(ismember(PShist,nos,'rows'))=[];
        PShist(ismember(PShist,nos,'rows'),:)=[];

end
ultima_tarefa=sum(1:n)-sum(best);
ordem=[best,ultima_tarefa];
```

```
[D,makespan]=makespanblock(Pjk,ordem);
tempo_computacional=toc;
```

ALGORITMO K – Solução inicial *PW*

```
function
[ordem,ramificacoes,makespan,tempo_computacional,D]=SolucaoinicialPW(Pjk)
tic
[Ordem,UB,~,~]=PW(Pjk);
n=size(Pjk,1);
Ordem(n)=[];
best=Ordem;
PS=[(1:n)',zeros(n,n-2)];
NPS=cumsum([2:n;-eye(n-1)]);
LB=zeros(n,1);
for j=1:n
    [LB(j)]=Lowerbound(Pjk,PS(j,1),NPS(j,:));
end
LBhist=LB;
PShist=PS;

    Cortar=LBhist>=UB;
    LBhist(Cortar)=[];
    PShist(Cortar,:)=[];
nodes=LBhist==min(LBhist);
nos=PShist(nodes,:);
LBhist(nodes)=[];
PShist(nodes,:)=[];

ramificacoes=n;
while ~isempty(nos)

    Camada=sum(nos>0,2)+1;
    Novos=(n-Camada(1)+1);
    PS=zeros(sum(nodes)*Novos,n-1);

    for no=1:size(nos,1)
        aux=1:n;
        aux(ismember(aux,nos(no,:)))=[];
```

```
        PS((no-1)*Novos+1:no*Novos,:)=cumsum([nos(no,:);zeros(Novos-1,n-
1)]);
        PS((no-1)*Novos+1:no*Novos,Camada(1))=aux;
    end

    LB=zeros(size(PS,1),1);

    for j=1:size(PS,1)
        ramificacoes=ramificacoes+1;
        NPS=1:n;
        NPS(ismember(NPS,PS(j,:)))=[];
        [LB(j)]=Lowerbound(Pjk,PS(j,1:Camada(1)),NPS);
    end
    LBhist(end+1:end+size(PS,1))=LB;
    PShist(end+1:end+size(PS,1),:)=PS;

    if max(Camada)==n-1
        if min(LB)<UB
            UB=min(LB);
            best=PS(find(LB==min(LB),1,'first'),:);
        end
    end

        Cortar=LBhist>=UB;
        LBhist(Cortar)=[];
        PShist(Cortar,:)=[];

        Ult_Camada=max(sum(PShist>0,2));

nodes=LBhist(sum(PShist>0,2)==Ult_Camada)==min(LBhist(sum(PShist>0,2)==Ul
t_Camada));
        nos=PShist(sum(PShist>0,2)==Ult_Camada,:);
        nos=nos(nodes,:);

        LBhist(ismember(PShist,nos,'rows'))=[];
        PShist(ismember(PShist,nos,'rows'),:)=[];

end

ultima_tarefa=sum(1:n)-sum(best);
ordem=[best,ultima_tarefa];
```

```
[D,makespan]=makespanblock(Pjk,ordem);
tempo_computacional=toc;
```