PEDRO LEITE ROCHA

UM PROLEMA DE SEQÜENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DE MÁQUINA E DA SEQÜÊNCIA: MODELOS E ALGORITMO EXATO

Belo Horizonte
20 de novembro de 2006

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

UM PROLEMA DE SEQÜENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DE MÁQUINA E DA SEQÜÊNCIA: MODELOS E ALGORITMO EXATO

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

PEDRO LEITE ROCHA

Belo Horizonte
20 de novembro de 2006

Resumo

Um problema de seqüenciamento em máquinas paralelas não-relacionadas, tempos de preparação dependentes de máquina e da seqüência, datas de entrega e tarefas ponderadas é considerado neste trabalho. Dois modelos de programação inteira e mista baseados em estratégias clássicas da literatura são avaliados. Um algoritmo *Branch-and-bound* (B&B) é desenvolvido e a solução encontrada pela metaheurística GRASP é utilizada como limite superior. Também propomos um conjunto de instâncias para este tipo de problema. Resultados computacionais são realizados em vários tipos de testes, onde o algoritmo tem um bom desempenho em instâncias de até 30 tarefas. Finalmente, conclusões e direções para futuros trabalhos são apresentadas.

Abstract

A scheduling problem with unrelated parallel machines, sequence and machine dependent setup times, due dates and weighted jobs is considered in this work. Two mixed integer programing (MIP) models based on classic strategies from the literature are evaluated. A Branch-and-Bound algorithm (B&B) is developed and a solution provided by the metaheuristic GRASP is used as an upper bound. We also propose a set of instances for this type of problem. Computational experiments are carried out in several types of tests, where the algorithm performs well on instances with up to 30 jobs. Finally, conclusions and directions for future work are presented.

Optimis parentibus

"For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mould what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable."

- Edsger Wybe Dijkstra

Agradecimentos

Agradeço a todos que contribuíram para que eu pudesse chegar até aqui. Agradeço principalmente a meus pais, a quem dedico esta dissertação.

À toda minha família, principalmente meus avós Antônio, Cota, Di e Dora, meu tio e padrinho João, meus tios Afrânio e Ricardo e minhas tias Elisa, France, Marília e Maria Inês, todos que sempre me deram força. Também a minha madrinha Sabrina e a "tia" Bel, que não são da família, mas é como se fossem.

Ao meu orientador, Robson, ao meu co-orientador, Martín, que é argentino mas é gente boa, e aos membros da banca que vieram prestigiar este trabalho.

Aos grandes amigos desde a infância até hoje: Luizinho, Ricardo e Thiago.

A todos os amigos do LaPO, em especial a Docinho, Dri, Fred, Gurvan, Lu e Olga.

Aos que tiveram que me agüentar morando debaixo do mesmo teto desde a graduação: Alla, minha irmã Diana, Lama, Manoel, Maurício, Menotti e Rafael.

Aos amigos para sempre: George, Ismael e Zeniel.

Ao Overflow: Gude e Torres, e as Overgirls: Alê, Camila e Lud.

Aos grandes amigos do GAAL: Ana, Carlos, Cazita, Delboni, Gaspar, Lidia e Luis.

Aos amigos do peito: Bebex, Clara, Fujii, Gripp, Igorele, Juliano, Liliane e Leonardo.

A todos os amigos do mestrado e doutorado: Aracelly e Pedrão, Bárbara, Bigonhinha, os cearenses César, Ju e Osama, Diego, Fabíola e Naka, Maurício e Ingrid, Raquel e Ruiter.

Aos chefes que tive nas empresas onde trabalhei durante o percurso: Alberto, Ana, Leo e Zé Carlos.

E a todas as outras pessoas que fizeram parte da minha vida de alguma forma ou se envolveram com este trabalho, vocês sabem quem são. Muito obrigado a todos.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

1.1 Sistema de Produção

O planejamento da produção é visto como o planejamento da aquisição de recursos e matérias primas, assim como o planejamento das atividades de produção, necessárias para transformar matéria prima em produtos acabados, atendendo a demanda dos consumidores da forma mais eficiente e econômica possível [Pochet e Wosley (2006)].

No contexto industrial, problemas de seqüenciamento estão relacionados à área de *Manufacturing Resource Planning* (MRP). A programação é feita de acordo com o horizonte de planejamento. Decisões de longo prazo têm características estratégicas, e portanto são tomadas pela alta administração. O médio prazo é conhecido como nível tático. Este trabalho foca no curto prazo, conhecido como nível operacional, onde o objetivo é ordenar a seqüência de produção com o objetivo de minimizar os custos de fabricação. Neste nível, a demanda e datas de entrega já estão definidas.

Neste tipo de problema, o objetivo principal é definir a seqüência de processamento para atender as demandas e datas de entrega. Além disso, os algoritmos desenvolvidos também procuram minimizar o tempo de produção e o atraso de cada produto.

1.2 Problemas de Seqüenciamento

O problema de seqüenciamento está associado à alocação de recursos a um conjunto de tarefas num dado tempo. Escalonar significa designar recursos para a execução das tarefas até que todas tenham sido executadas sob as restrições impostas com o objetivo de minimizar o comprimento do seqüenciamento, dado pelo tempo de processamento das tarefas, ou um outro critério estabelecido no contexto do problema [Błażewicz et al. (1996)]. Aplicações desse problema podem ser encontradas em diversas situações do mundo real, por exemplo, no planejamento da produção, no gerenciamento de projetos e no seqüenciamento de tarefas pela unidade central de processamento do computador.

Esse problema tem sido estudado desde o início do século XX, com os primeiros trabalhos inseridos no contexto da manufatura cujos recursos considerados são máquinas. Na década

de 50, surgiram as primeiras publicações com os trabalhos de Johnson (1954), Jackson (1955, 1956) e Smith (1956). Inicialmente, a maioria dos trabalhos relacionava-se com a análise de problemas com uma única máquina ou sistemas com máquinas paralelas. Nesses sistemas, considera-se um conjunto de máquinas idênticas em paralelo, sendo que as tarefas podem ser processadas em qualquer uma das máquinas. Com o passar do tempo, situações mais complexas de seqüenciamento de máquinas foram investigadas, como a utilização de máquinas com velocidades de processamento diferentes e a utilização de mais de uma máquina para a execução da mesma tarefa.

1.3 Notação para Problemas de Seqüenciamento

Na maioria dos problemas de seqüenciamento, o número de tarefas e o número de máquinas é assumido finito. O número de tarefas é denotado por n, e o número de máquinas por m. Normalmente, o sub-escrito j se refere a uma tarefa, e o sub-escrito i a uma máquina. Se uma tarefa requer um número de passos ou operações de processamento, então o par (i, j) se refere ao passo ou operação de processamento da tarefa j na máquina i. Essa notação foi retirada do trabalho de Pinedo (1995). Os dados seguintes são relacionados à tarefa j.

- **Tempo de processamento (p_{ij}).** O p_{ij} representa o tempo de processamento da tarefa j na máquina i. O sub-escrito i pode ser omitido se o tempo de processamento de j não depender da máquina, ou se j for processado apenas em uma máquina.

- **Tempo de disponibilidade (r_j).** É o tempo no qual a tarefa j chega ao sistema, isto é, o menor tempo para que o processamento da tarefa j possa começar.

- **Data de entrega (d_j).** A data de entrega d_j representa a data de entrega prometida da tarefa j. Atrasos na data de entrega podem ser permitidos, mas uma penalidade deve ser inserida. Quando a data de entrega tiver que ser absolutamente obedecida, ela é chamada de *deadline*.

- **Prioridade ou peso (w_j).** A prioridade ou peso w_j da tarefa j basicamente denota a importância da tarefa j relativa às outras tarefas no sistema. Por exemplo, essa prioridade pode representar o custo de armazenagem da tarefa.

Um problema de seqüenciamento é descrito por um trio $\alpha|\beta|\gamma$. O campo α descreve o ambiente de máquinas, e contém uma única entrada. O campo β fornece detalhes das características de processamento e restrições e pode ser vazio, conter uma única entrada ou conter entradas múltiplas. O campo γ contém o objetivo que deve ser minimizado e normalmente contém uma única entrada.

Os seguintes são exemplos de possíveis ambientes de máquinas:

- **Uma máquina (1).** O caso de uma única máquina é o mais simples de todos os ambientes de máquinas e é um caso especial de todos os outros ambientes de máquinas mais complexos.

- **Máquinas idênticas (Pm).** Existem m máquinas idênticas em paralelo. A tarefa j requer uma única operação e pode ser processada por qualquer uma das m máquinas ou por qualquer uma pertencente a um subconjunto. Se a tarefa j não pode ser processado em qualquer máquina, mas em um subconjunto, digamos o subconjunto M_j, então a entrada M_j aparece no campo β.

- **Máquinas paralelas com velocidades diferentes (Qm).** Existem m máquinas em paralelo com velocidades diferentes. As máquinas realizam o trabalho da mesma forma, diferindo apenas na velocidade, que é denotada por v_i para a máquina i. O tempo p_{ij} da tarefa j na máquina i é igual a $\frac{p_j}{v_i}$, assumindo que ela possa ser processada pela máquina i. Esse ambiente de máquinas também é conhecido como máquinas uniformes. Se todas as máquinas têm a mesma velocidade, isto é, $v_i = 1 \forall i$, então $p_{ij} = p_j$ e o ambiente se torna idêntico ao anterior.

- **Máquinas não-relacionadas (Rm).** Esse ambiente é uma generalização do anterior. Existem m máquinas diferentes em paralelo. A máquina i pode processar a tarefa j com uma velocidade v_{ij}. O tempo p_{ij} que a tarefa j gasta na máquina i é igual a $\frac{p_j}{v_{ij}}$. Se as velocidades das máquinas são independentes da tarefa, isto é, $v_{ij} = v_i \forall \{i, j\}$, então o ambiente se torna idêntico ao anterior.

- **Flow shop (Fm).** Existem m máquinas em série. Cada tarefa tem que ser processada por cada uma das m máquinas. Todas as tarefas têm o mesmo roteiro, isto é, devem ser processados primeiro pela máquina 1, depois pela máquina 2, e assim por diante. Após o término em uma máquina um tarefa entra na fila para a próxima máquina. Normalmente, todas as filas operam com *first in first out (FIFO)*, isto é, uma tarefa não pode "passar" outra na fila. Se FIFO é adotado, o flow shop é referido como flow shop com permutação, e o campo β deve incluir a entrada *prmu*, que é apresentada abaixo.

- **Flow shop flexível (FFs).** Um flow shop flexível é uma generalização dos ambientes flow shop e máquinas paralelas. Em vez de m máquinas em série, existem s estágios em série com um número de máquinas em paralelo em cada estágio. Cada tarefa deve ser processada primeiro no estágio 1, depois no estágio 2, e assim por diante. Um estágio funciona como um banco de máquinas paralelas: em cada estágio, uma tarefa j requer apenas uma máquina e, normalmente, qualquer máquina pode processar qualquer tarefa. As filas entre os vários estágios também operam, em geral, com a estratégia FIFO.

- **Open shop (Om).** Existem m máquinas. Cada tarefa tem que ser processada em cada uma das m máquinas. No entanto, alguns dos tempos de processamento podem ser zero. Não há restrições quanto ao roteiro de cada tarefa pelo ambiente de máquinas. É possível determinar um roteiro para cada tarefa, e tarefas diferentes podem ter roteiros diferentes.

- **Job shop (Jm).** Num job shop com m máquinas, cada tarefa tem seu próprio roteiro a ser seguido. Uma distinção é feita entre *job shops* onde cada tarefa visita qualquer

máquina no máximo uma vez e *job shops* onde uma tarefa pode visitar uma máquina mais de uma vez. No último, o campo β contém a entrada *recrc* para recirculação, que é apresentada abaixo.

As restrições especificadas no campo β podem incluir múltiplas entradas. São algumas das entradas possíveis:

- **Data de disponibilidade (r_j).** Se este símbolo estiver presente no campo β, a tarefa j não pode iniciar seu processamento antes da data de disponibilidade r_j. Se r_j não aparece em β, o processamento da tarefa j pode começar a qualquer hora. Em contraste com datas de disponibilidade, datas de entrega não são especificadas neste campo. O tipo de função objetivo dá indicação suficiente se existem datas de entrega ou não.

- **Tempos de preparação dependentes da seqüência (s_{jk}).** Representa o tempo de preparação dependente da seqüência entre as tarefas j e k; s_{0k} denota o tempo de preparação para a tarefa k se a tarefa k é a primeira na seqüência e s_{j0} denota o tempo de limpeza após a tarefa j se for a última na seqüência. Se o tempo de preparação entre as tarefas j e k depende da máquina, o sub-escrito i é incluído, isto é, s_{ijk}. Se s_{jk} não aparece no campo β, todos os tempos de preparação são assumidos zero ou independentes da seqüência, podendo ser simplesmente adicionados aos tempos de processamento.

- **Preempções (*prmp*).** Preempções sugerem que não é necessário manter um tarefa em uma máquina até o término. É possível interromper o processamento de um tarefa a qualquer hora, e colocar um tarefa diferente na máquina. A quantidade de tempo de processamento que uma tarefa interrompida já recebeu não é perdida. Quando uma tarefa interrompida volta à máquina (ou a outra máquina no caso de máquinas paralelas), ela só precisa da máquina para o tempo de processamento restante. Quando preempções são permitidas, *prmp* é incluído ao campo β.

- **Restrições de precedência (*prec*).** Restrições de precedência podem aparecer em ambientes de uma única máquina ou de máquinas paralelas, determinando que uma ou mais tarefas tenham que ter sido completadas antes que outra tarefa seja permitida começar. Existem várias formas especiais de restrições de precedência. Se cada tarefa tem no máximo uma sucessora, as restrições são referidas como uma árvore interna. Se cada tarefa tem no máximo uma predecessora, as restrições são referidas como árvore externa. Se cada tarefa tem no máximo uma predecessora e uma sucessora, as restrições são referidas como correntes.

- **Quebras (*brkdwn*).** Quebras de máquinas sugerem que máquinas não estão sempre disponíveis. Os períodos em que uma máquina não está disponível são assumidos fixos (por exemplo, devido a turnos ou manutenções). Se há máquinas idênticas em paralelo, o número de máquinas disponíveis é uma função do tempo $m(t)$.

- **Restrições de elegibilidade (M_j).** Uma restrição que pode aparecer no campo β quando o ambiente de máquinas contiver m máquinas em paralelo (Pm). Quando M_j está presente, nem todas as m máquinas são capazes de processar a tarefa j. O conjunto M_j denota o conjunto de máquinas capazes de processar a tarefa j. Se o campo β não contém M_j, a tarefa j pode ser processado por qualquer uma das m máquinas.

- **Permutação (*prmu*).** Uma restrição que pode ser aplicada no ambiente de *flow shop*, é que as filas na entrada de cada máquina operam de acordo com a política *FIFO*. Isso quer dizer que a ordem (ou permutação) em que as tarefas passam pela primeira máquina é mantida através do sistema.

- **Bloqueio (*block*).** Bloqueio é um fenômeno que ocorre em *flow shop*. Se um *flow shop* tem um buffer limitado entre duas máquinas sucessivas, pode ser que, quando um buffer ficar cheio, a máquina anterior não possa liberar a tarefa terminada. Esse fenômeno se chama bloqueio: a tarefa terminada tem que permanecer na máquina anterior evitando, ou bloqueando, a máquina de processar outros tarefas.

- **Sem espera (*nwt).*** A restrição sem-espera é outro fenômeno que pode ocorrer em *flow shop*. Tarefas não podem esperar entre duas máquinas sucessivas. Isso quer dizer que o tempo de início de uma tarefa pode ser atrasado para garantir que a tarefa possa atravessar o *flow shop* sem ter que esperar por nenhuma máquina. Um exemplo de uma operação é uma linha de fabricação, onde uma determinada peça de metal não pode esperar pela próxima etapa porque ela esfriaria.

- **Recirculação (*recrc*).** Recirculação pode ocorrer em *job shop*, quando uma tarefa pode visitar uma máquina mais de uma vez.

Outras entradas do campo β são auto-explicativas. Por exemplo, $p_j = p$ indica que todos os tempos de processamento são iguais, e $d_j = d$ indica que todas as datas de entrega são iguais. Como dito antes, datas de entrega, ao contrário das datas de disponibilidade, não são normalmente explicitadas no campo β; o tipo da função objetivo nos dá indicação suficiente se os tarefas têm data de entrega ou não.

O objetivo a ser minimizado é sempre uma função dos tempos de término das tarefas, que, é claro, dependem do seqüenciamento, ou também pode ser uma função das datas de entrega. O tempo de término da operação da tarefa j na máquina i é dado por c_{ij}. O tempo em que a tarefa j sai do sistema (o tempo de término na última máquina onde ele requer processamento) é denotado por c_j. O *lateness* de um tarefa j é definido como

$$L_j = c_j - d_j$$

que é um valor positivo quando a tarefa j é terminada atrasada e negativo quando é completada adiantada. O atraso, ou *tardiness*, da tarefa j é definida como

$$T_j = \max(c_j - d_j, 0) = \max(L_j, 0)$$

A diferença entre o atraso e *lateness* é que o valor do atraso nunca é negativo. A penalidade unitária da tarefa j é definida como

$$U_j = \begin{cases} 1 & \text{se } c_j > d_j \\ 0 & \text{c.c.} \end{cases}$$

A seguir, temos alguns exemplos de possíveis funções objetivo a serem minimizadas.

- ***Makespan*** **(c_{max}).** O *makespan*, definido como $\max(c_1, \ldots, c_n)$ é equivalente ao término da última tarefa a sair do sistema. Um *makespan* mínimo geralmente indica uma alta utilização das máquinas.

- ***Lateness*** **máximo (l_{max}).** O máximo *lateness*, l_{max}, é definido como $\max(l_1, \ldots, l_n)$. Mede a pior violação das datas de entrega.

- **Tempo de término total ponderado ($\sum w_j \cdot c_j$).** A soma dos tempos ponderados de término das n tarefas, utilizando a prioridade w_j como fator de ponderação, dá uma indicação do custo total do seqüenciamento. A soma dos tempos de término é freqüentemente referida na literatura como tempo de fluxo (*flow time*). O tempo de término total ponderado é chamado de tempo de fluxo ponderado.

- **Tempo de término total ponderado descontado ($\sum(1 - e^{-rc_j})$).** Essa é uma função de custo mais geral que a anterior, onde os custos são descontados a uma taxa de r, $0 < r < 1$, por unidade de tempo. Isso é, se a tarefa j não termina até o tempo t, um custo adicional $w_j r e^{-rt} dt$ é adicionado durante o período $[t, d+dt]$. Se a tarefa j termina no tempo t, o custo total do período $[0, t]$ é $w_j(1 - e^{-rt})$. O valor de r geralmente é perto de 0, como $0,1$ (ou 10%).

- **Atraso total ponderado ($\sum w_j T_j$).** Também é uma função de custo mais geral que o tempo de término total ponderado.

- **Número ponderado de tarefas atrasadas ($\sum w_j U_j$).** O número ponderado de tarefas atrasadas, não somente é uma métrica de interesse acadêmico, mas também é um objetivo freqüente na vida real.

1.4 Problema Abordado

O problema considerado neste trabalho é um problema de seqüenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes de máquina e da seqüência, datas de entrega e tarefas ponderadas. Este tipo de problema consiste em programar várias tarefas para serem processadas por várias máquinas. Cada tarefa deve ser seqüenciada em uma máquina específica, e a ordem em que cada máquina irá processar suas tarefas deve ser decidida. Cada tarefa tem um tempo de processamento diferente para cada máquina. Além disso, quando uma máquina termina de processar uma tarefa, há um tempo de preparação

da máquina antes que ela possa processar a próxima tarefa. Este tempo existe para que a máquina possa ser preparada para a próxima tarefa (e.g. limpa ou re-configurada) e também é diferente para cada par de tarefas e cada máquina. Cada tarefa tem também uma data de entrega e uma prioridade. Embora a data de entrega possa ser excedida, isso causa uma penalidade na função objetivo dependente da prioridade da tarefa. Seguindo a notação de Pinedo (1995) citada na Seção 1.3, este problema pode ser formalmente definido como um $Rm|s_{ijm}|c_{max} + \sum T_i \cdot w_i$, onde o significado de cada membro é:

- Rm: problema de seqüenciamento com m máquinas paralelas não-relacionadas,
- s_{ijm}: tempos de preparação dependentes de máquina e da seqüência,
- $c_{max} + \sum T_i \cdot w_i$ o objetivo é minimizar o *makespan* acrescido do atraso ponderado.

Optamos por uma abordagem exata para o problema, utilizando modelos baseados em estratégias clássicas da literatura, e desenvolvendo um algoritmo específico para ele baseado em Branch-and-Bound (B&B) e GRASP (Greedy Randomized Adaptive Search Procedure). Algoritmos baseados em B&B e na metaheurística GRASP não são incomuns ao se lidar com problemas de seqüenciamento, assim como em outros problemas similares [Roundy et al. (1999)]. Rabadi et al. (2004) propõe um B&B para um problema de seqüenciamento com datas de entrega iguais e uma máquina com tempos de preparação dependentes da seqüência (TPDS). Uma adaptação de GRASP [Resende e Feo (1995)] é usada para encontrar um limite superior para o problema explorado nesta dissertação. Também podem ser citados os trabalhos de Feo et al. (1991); Dachs et al. (1996), onde aplicações de GRASP para seqüenciamento em uma máquina são mostradas, e Resende et al. (2002), que mostram uma aplicação interessante da metaheurística para problemas de job-shop.

O estudo de problemas de seqüenciamento com TPDS tem atraído muita atenção nos últimos anos [Kim e Bobrowski (1994)]. Eles também são alguns dos problemas de seqüenciamento mais complexos. Considerando apenas uma máquina e um estágio, o problema é equivalente ao do caixeiro viajante [Błażewicz et al. (1996)]. Garey e Johnson (1979) mostraram que minimizar o *makespan* em duas máquinas idênticas é um problema NP-difícil. Certamente, um problema com máquinas não-relacionadas e datas de entrega também é NP-difícil.

Há muitos trabalhos considerando máquinas paralelas, mas poucos consideram máquinas paralelas e TPDS. Luh et al. (1998) aplicam uma solução baseada em relaxação lagrangeana, programação dinâmica e heurísticas para um flow-shop com TPDS. Liu e Liao (2000) consideram vários estágios, flexibilidade e TPDS. Eles também utilizam relaxação lagrangeana em um modelo que considera restrições de fluxo e máquinas idênticas em cada estágio. Acero e Delgado (2000) utilizam uma heurística baseada em busca tabu em um problema de flow-shop com máquinas não-relacionadas em cada estágio, mas sem considerar TPDS. Meyr (2000) utiliza busca local combinada com re-otimização dual para resolver um problema de lot-sizing, considerando um *flow-shop* numa linha de produção única com TPDS. Num trabalho posterior, Meyr (2002) adiciona máquinas paralelas não-relacionadas ao problema original.

Hans-Joachim e John (1996) exploram um problema de job-shop sem TPDS utilizando programação por restrições. Koulamas e Kyparisis (2004) considera um problema de minimização de makespan em máquinas paralelas uniformes com tempos de disponibilidade.

Na mesma linha, foram publicados vários trabalhos aplicando GRASP e B&B ao problema aqui estudado [Gómez Ravetti e Mateus (2003); Gómez Ravetti et al. (2004a,b, 2006); Rocha et al. (2004, 2006)]. No entanto, não foi possível encontrar trabalhos de outros autores considerando um problema com tantas características quanto o aqui estudado.

1.5 Organização

Este trabalho está organizado na seguinte seqüência. No Capítulo 2 é detalhado o problema abordado e são discutidas as formulações matemáticas utilizadas. O Capítulo 3 apresenta um algoritmo Branch-and-Bound, e como ele foi adaptado para o problema abordado. A geração de instâncias, testes e resultados são discutidos no Capítulo 4. Finalmente, o Capítulo 5 aponta as conclusões do trabalho e direções para trabalhos futuros.

Capítulo 2

Formulações Matemáticas

Dois modelos são apresentados nesta seção para produzir soluções ótimas. O primeiro é baseado no modelo de Manne (1960) para *Job Shop*, e já foi previamente proposto por Gómez Ravetti (2003). O segundo é baseado no modelo de Wagner (1959), também para *Job Shop*. Ambos têm os seguintes dados:

- N é o conjunto de tarefas que devem ser processadas;
- M é o conjunto de máquinas;
- i representa uma tarefa específica;
- m representa uma máquina específica;
- d_i é a data de entrega da tarefa i;
- w_i é a prioridade da tarefa i;
- p_{im} é o tempo de processamento da tarefa i na máquina m;
- $s_{ii'm}$ é o tempo de preparação da máquina m ao passar da tarefa i para a tarefa i';
- G é um valor inteiro muito grande que atua como penalidade.

2.1 Modelo Baseado no Trabalho de Manne

O primeiro modelo de programação inteira mista é baseado no modelo de Manne para *Job Shop*. Ele trabalha discretizando a ordem das tarefas nas máquinas e conta com as seguintes variáveis de decisão:

- t_i: tempo de início do processamento da tarefa i;
- α_{im}: $\begin{cases} 1 & \text{se a tarefa } i \text{ é processada pela máquina } m, \\ 0 & \text{caso contrário;} \end{cases}$

- $\beta_{ii'm}$: $\begin{cases} 1 & \text{se as tarefas } i \text{ e } i' \text{ são processadas na máquina } m \text{ e } i \text{ é processada antes de } i', \\ 0 & \text{caso contrário;} \end{cases}$
- ρ_i: atraso da tarefa i;
- Z: makespan (tempo para o término de todas as tarefas).

A formulação matemática do modelo é dada e descrita a seguir:

$$\min\left(Z + \sum_{i \in N} (\rho_i \cdot w_i)\right) \tag{2.1}$$

$$\sum_{m \in M} \alpha_{im} = 1, \qquad \forall i \tag{2.2}$$

$$Z \geq t_i + p_{im} - (1 - \alpha_{im}) \cdot G, \qquad \forall i, \forall m \tag{2.3}$$

$$\rho_i \geq t_i + p_{im} - d_i - (1 - \alpha_{im}) \cdot G, \qquad \forall i, \forall m \tag{2.4}$$

$$t_{i'} \geq t_i + p_{im} + s_{ii'm} - (1 - \alpha_{im}) \cdot G - (1 - \alpha_{i'm}) \cdot G - (1 - \beta_{ii'm}) \cdot G, \quad \forall i, \forall i' > i, \forall m \tag{2.5}$$

$$t_i \geq t_{i'} + p_{i'm} + s_{i'im} - (1 - \alpha_{im}) \cdot G - (1 - \alpha_{i'm}) \cdot G - \beta_{ii'm} \cdot G, \quad \forall i, \forall i' > i, \forall m \tag{2.6}$$

$$\alpha_{im} \in \{0, 1\}, \quad \forall i, \forall m \qquad\qquad \beta_{ii'm} \in \{0, 1\}, \quad \forall i, \forall i' > i, \forall m$$

(2.1) A função objetivo é minimizar o makespan somado ao atraso ponderado. Apesar de considerar dois objetivos (makespan e atraso), ela é considerada mono objetivo, pois apenas soma os valores dos dois objetivos. No caso deste trabalho, preferiu-se essa abordagem porque ela garante sempre a existência de uma solução viável, em contraste com uma abordagem em que as datas de entrega não poderiam ser violadas.

(2.2) Impõe que cada tarefa deve ser processada por exatamente uma máquina.

(2.3) Calcula o makespan. Ele deve ser maior ou igual ao tempo de início de cada tarefa i adicionado ao tempo de processamento de i na máquina m onde i é processada.

(2.4) O início do processamento da tarefa i adicionado ao tempo de processamento na máquina m onde i é processada deve ser menor ou igual à sua data de entrega d_i acrescida de um possível atraso ρ_i.

(2.5) Impõe que o processamento da tarefa i' poderá começar somente após o processamento da tarefa i e o tempo de preparação de i para i', se ambas forem processadas na máquina m ($\alpha_{im} = \alpha_{i'm} = 1$) e i' for processada após i na máquina m ($\beta_{ii'm} = 1$).

(2.6) Impõe o recíproco, que o processamento da tarefa i poderá começar somente após o processamento da tarefa i' e o tempo de preparação de i' para i, se ambas forem processadas na máquina m ($\alpha_{im} = \alpha_{i'm} = 1$) e i' não for processada após i na máquina m ($\beta_{ii'm} = 0$).

O modelo ainda seria válido se as restrições (2.5) e (2.6) fossem substituídas pelas Restrições (2.5) $\forall i \neq i'$ e adicionando $\beta_{ii'm} = 1 - \beta_{i'im}$, mas essa nova formulação exigiria duas vezes o número de variáveis β. Com a formulação atual, $\beta_{ii'm}$ existe apenas se $i < i'$, diminuindo o número de variáveis β pela metade e mantendo o número de restrições.

2.2 Modelo Baseado no Trabalho de Wagner

Enquanto o primeiro modelo utiliza α para discretizar a alocação de tarefas em máquinas, o segundo modelo adiciona mais informação a essas variáveis, utilizando-as para discretizar as posições na seqüência de processamento. Ele tem as seguintes variáveis de decisão:

- $t_m^{(l)}$: tempo de início do processamento da l^{a} tarefa na máquina m;
- $\alpha_{im}^{(l)}$: $\begin{cases} 1 & \text{se a tarefa } i \text{ for processada pela máquina } m \text{ na } l^{\text{a}} \text{ posição,} \\ 0 & \text{caso contrário;} \end{cases}$
- $\delta_m^{(l)}$: tempo de preparação necessário entre as l^{a} e $(l+1)^{\text{a}}$ posições na máquina m;
- ρ_i: atraso da tarefa i;
- Z: makespan.

A formulação matemática é dada e descrita abaixo:

$$\min \left(Z + \sum_{i \in N} (\rho_i \cdot w_i) \right) \tag{2.7}$$

$$\sum_{m \in M} \sum_{l=1}^{|N|} \alpha_{im}^{(l)} = 1, \qquad \forall i \tag{2.8}$$

$$\sum_{i \in N} \alpha_{im}^{(l)} \leq 1, \qquad \forall m, l = 1, \ldots, |N| \tag{2.9}$$

$$\sum_{i \in N} \alpha_{im}^{(l)} \leq \sum_{i \in N} \alpha_{im}^{(l-1)}, \qquad \forall m, l = 2, \ldots, |N| \tag{2.10}$$

$$Z \geq t_m^{(l)} + \sum_{i \in N} \left(\alpha_{im}^{(l)} \cdot p_{im} \right), \qquad \forall m, l = 1, \ldots, |N| \tag{2.11}$$

$$\rho_i \geq t_m^{(l)} + p_{im} - d_i - \left(1 - \alpha_{im}^{(l)} \right) \cdot G, \qquad \forall m, \forall i, l = 1, \ldots, |N| \tag{2.12}$$

$$t_m^{(1)} = 0, \qquad \forall m \tag{2.13}$$

$$\delta_m^{(l-1)} \geq s_{ii'm} - \left(2 - \alpha_{im}^{(l-1)} - \alpha_{i'm}^{(l)} \right) \cdot G, \qquad \forall m, \forall i, \forall i' \neq i, l = 2, \ldots, |N| \tag{2.14}$$

$$t_m^{(l)} \geq t_m^{(l-1)} + \delta_m^{(l-1)} + \sum_{i \in N} \left(\alpha_{im}^{(l-1)} \cdot p_{im} \right), \qquad \forall m, l = 2, \ldots, |N| \tag{2.15}$$

$$\alpha_{im}^{(l)}, \in \{0, 1\}, \quad \forall i, m, l = 1, \ldots, |N|$$

(2.7) A função objetivo é similar à do primeiro modelo: minimizar o makespan somado ao atraso ponderado.

(2.8) Cada tarefa é processada em exatamente uma máquina em uma posição.

(2.9) Há no máximo uma tarefa em cada posição de cada máquina.

(2.10) Deve haver uma tarefa na posição $l-1$ se houver outra na posição l na mesma máquina para $l \geq 2$.

(2.11) O makespan é maior ou igual ao tempo de início do processamento das tarefas em cada posição em cada máquina somado ao tempo de processamento da tarefa na máquina.

(2.12) O tempo de início do processamento da tarefa na l^{a} posição na máquina m adicionado ao tempo de processamento da tarefa deve ser menor ou igual à data de entrega da tarefa adicionada de um possível atraso ρ_i.

(2.13) O tempo do início do processamento de todas as tarefas das primeiras posições é igual a zero.

(2.14) O tempo de preparação entre as $(l-1)^{\text{a}}$ e l^{a} posições na máquina m é maior ou igual ao tempo de preparação necessário da tarefa i para i' na máquina m se i e i' são processadas por m nas $(l-1)^{\text{a}}$ e l^{a} posições respectivamente.

(2.15) O tempo de início do processamento para a tarefa na l^{a} posição é maior ou igual ao tempo de início do processamento da tarefa da posição anterior somado ao tempo de preparação necessário entre as $(l-1)^{\text{a}}$ e l^{a} posições da máquina m calculado em (2.14), e o tempo de processamento da tarefa na $(l-1)^{\text{a}}$ posição.

Ao utilizar um valor inteiro G muito grande para transformar o modelo em um modelo linear, as restrições contendo a constante G dão limites inferiores fracos durante a resolução se associadas a variáveis inteiras ou binárias, assim como no nosso modelo. Isso acontece porque, durante a resolução, o *solver* relaxa as variáveis binárias, permitindo que elas assumam valores não inteiros. No primeiro modelo, nas Restrições (2.3-2.6), se $0 < \alpha_{im} < 1$, o valor resultante no lado direito será fortemente negativo, não influenciando o limite inferior da variável do lado esquerdo. Assim, o limite inferior da função objetivo também será fraco.

No segundo modelo, apenas as Restrições (2.12) e (2.14) utilizam G, fazendo com que apenas $\delta_m^{(l)}$ e ρ_i tenham limites inferiores fracos. Grande parte do valor da função objetivo é dado pelo makespan, que é em grande parte dado pelo tempo de processamento de cada tarefa. Como a parte do modelo que calcula esses dados $\left(\sum \alpha_{im}^{(l-1)} \cdot p_{im}\right)$ não possui a constante G associada a variáveis binárias ou inteiras, o limite inferior não é prejudicado.

Vale a pena destacar que neste trabalho consideramos que cada máquina tem um número de posições igual ao número de tarefas. Embora o caso em que a solução ótima contenha todas as tarefas em uma única máquina seja muito raro, essa consideração é necessária para garantir a solução ótima.

Apesar de o segundo modelo ter um número de variáveis bem maior que o primeiro, ele é uma formulação mais forte com um limite inferior mais justo durante a resolução. É esperado que ele seja mais rápido para resolver problemas com números de tarefas maiores, quando o tempo de início de processamento (t_i ou $t_m^{(l)}$) puder ser muito maior que zero.

Capítulo 3

Algoritmo *Branch-and-Bound*

Um Branch-and-Bound (B&B) é uma estratégia específica de enumeração da árvore de soluções. Em um B&B, há três procedimentos principais: inicialização, *branching* e *bounding*. Durante a inicialização, uma rápida heurística é utilizada para encontrar uma boa solução inicial que servirá de limite superior (LS) no início do algoritmo.

Branching divide o problema em sub-problemas menores. Cada sub-problema representa uma solução parcial e é representado por um nó na árvore de soluções. Uma estratégia de busca deve estar associada ao procedimento de branching para decidir qual nó deve ser expandido a seguir. O limite superior ajuda a podar nós da árvore de busca que tiverem um limite inferior (LI) maior que o limite superior encontrado até o momento (num problema de

Algoritmo 3.1: Um Branch-and-Bound genérico

```
   Procedimento: BranchAndBound()
 1 Sol ← SoluçãoInicial();
 2 LS ← ValorDaSolução(Sol);
 3 LI ← LimiteInferior();
 4 Branch(primeiroNo);
 5 retorna Sol;

   Procedimento: Branch(no)
 6 se SoluçãoCompleta(no) então
 7 |  Sol ← no;
 8 |  LS ← ValorDaSolução(Sol);
 9 |  se LS = LI então
10 |  |  TerminaProcedimento();
11 |  fim se
12 |  retorna ;
13 fim se
14 para todo n ∈ FilhosDe(no) faça
15 |  se Bound(n) < LS então
16 |  |  Branch(n);
17 |  fim se
18 fim para todo
```

minimização).

O procedimento de *bounding* calcula o limite inferior de cada nó para decidir quais nós devem ser podados e quais devem ser expandidos. A estrutura básica de um *Branch-and-Bound* genérico é mostrada no Algoritmo 3.1. Nas seções seguintes, os três procedimentos são personalizados para este problema de seqüenciamento e descritos em maiores detalhes.

3.1 Inicialização com GRASP

Durante a inicialização, uma solução inicial completa é encontrada para servir de limite superior. Qualquer nó na árvore de enumeração com um limite inferior maior que o limite superior pode ser podado. Neste trabalho, um algoritmo baseado na metaheurística GRASP [Resende e Feo (1995)] é utilizado para encontrar essa primeira solução. GRASP é uma metaheurística de multi-início para problemas combinatórios [Resende e Feo (1995); Dachs et al. (1996)], e é composta por duas fases: a construção de uma solução aleatória viável e uma busca local. Essas duas fases são repetidas em cada iteração. Uma descrição está disponível no algoritmo 3.2.

Inicialmente, é construído um vetor com as tarefas ordenadas pela regra da menor data de entrega (earliest due date - EDD). Durante a fase de construção, o algoritmo re-ordena esse vetor de forma aleatória, seguindo uma função de probabilidade específica. Neste trabalho, a função utilizada é $f(x) \sim \frac{1}{x}$, onde $f(x)$ representa a probabilidade de a x^{a} tarefa ser escolhida como próxima. A cada passo da re-ordenação, uma nova tarefa é escolhida da lista de tarefas que ainda não fazem parte da nova ordenação. A probabilidade de uma tarefa específica ser escolhida é proporcional a $\frac{1}{x}$, onde x é a posição atual da tarefa na lista.

Após a re-ordenação, cada tarefa é alocada a uma máquina capaz de processá-la. Para

Algoritmo 3.2: Bloco principal do B&B e algoritmo de inicialização (GRASP)

```
   Procedimento: BranchAndBound()
 1 Sol ← GRASP(iteracoes);
 2 Branch1(1);
 3 retorna Sol;

   Procedimento: GRASP(iteracoes)
 4 tars ← Ordena(N);
 5 para i ← 1 até iteracoes faça
 6 |   tarsAleatorias ← ReOrdena(tars);
 7 |   solucaoLocal ← ConstróiSolução(tarsAleatorias);
 8 |   BuscaLocal(solucaoLocal);
 9 |   PathRelinking(solucaoLocal);
10 |   se ValorDaSolução(Sol) > ValorDaSolução(solucaoLocal) então
11 |   |   Sol ← solucaoLocal;
12 |   fim se
13 fim para
14 retorna Sol;
```

isso, uma função gulosa aloca cada uma na ordem resultante à máquina capaz de terminá-la primeiro. Um exemplo é exibido na Figura 3.1.

A busca local troca todos os pares de tarefas alocados a máquinas diferentes. Se uma troca melhora o valor da solução, a nova solução é armazenada e a antiga é abandonada. Nada é feito caso contrário. Um exemplo é mostrado na Figura 3.2. Foram feitos testes com várias estratégias de buscas, tanto buscas mais restritas como mais variadas. Embora essa estratégia seja computacionalmente pesada, ela foi a que encontrou melhores soluções em uma mesma quantidade de tempo.

Path-relinking é uma técnica utilizada como estratégia de busca em várias heurísticas [Glover (1996, 2000); Glover e Laguna (1997)], incluindo em conjunto com GRASP [Laguna e Martí (1999)]. Glover e Kochenberger (2003) apresentam duas estratégias principais:

- PR como uma ferramenta para melhorar a solução encontrada pelo GRASP.
- PR como uma estratégia para intensificar a busca local.

O algoritmo escolhe duas soluções e analisa o caminho de soluções entre elas. Esse caminho é feito através de movimentos de tarefas. Para cada movimento, a solução encontrada é analisada.

Neste trabalho, PR funciona como uma estratégia para intensificar a busca local. Mesmo assim, há várias formas de se implementar a técnica. Glover e Kochenberger (2003) citam algumas:

- Uso periódico de PR.
- Uso de PR em duas direções possíveis, isto é, da solução 1 à solução 2 e de 2 a 1.
- Uso de PR em apenas uma direção.
- Uso de PR truncado. Isto é, fazer a análise de apenas parte do caminho.

O algoritmo de PR deste trabalho funciona da seguinte forma:

1. O conjunto das cinco melhores soluções é mantido (as cinco melhores soluções, neste trabalho).

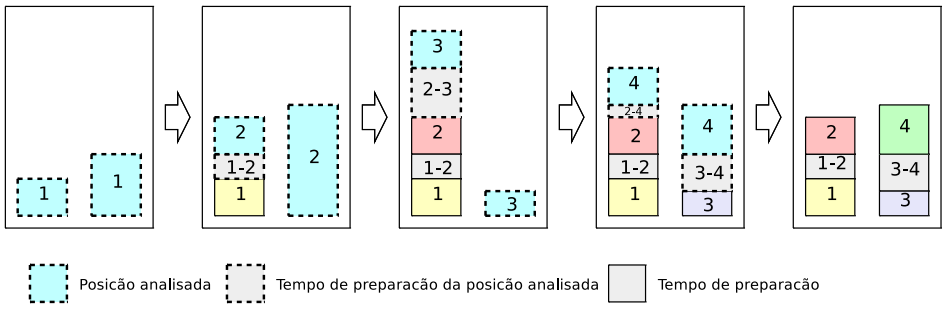


Figura 3.1: Seqüência de passos da construção com as posições analisadas para cada tarefa em destaque

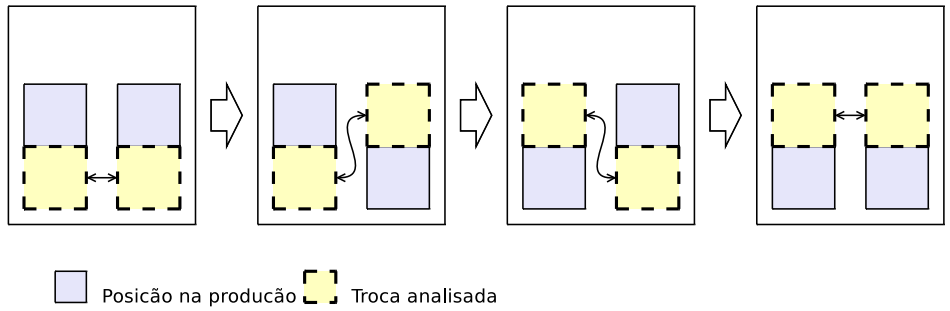


Figura 3.2: Seqüência de passos da busca local com as trocas analisadas em destaque

2. Cada vez que PR é usado, uma solução é escolhida aleatoriamente deste conjunto e todas as soluções no caminho entre a solução encontrada pela busca local e a solução selecionada do conjunto são analisadas.

3. Se uma solução melhor for encontrada, ela é adicionada ao conjunto.

As transformações feitas por PR são detalhadas no Algoritmo 3.3 e um exemplo com 2 máquinas e 4 tarefas é mostrado na Figura 3.3.

3.2 O Procedimento de *Branching*

O procedimento de *branching* constrói a árvore de enumeração. Ele é dividido em duas fases. Na primeira, as tarefas são alocadas às máquinas, mas a seqüência de produção não é definida. A seqüência de produção em cada máquina será decidida apenas na segunda fase.

A primeira fase de branching começa ordenando as tarefas utilizando a regra de EDD. A função `PróximaMáquina(`*tar*`)` retorna as máquinas seguindo a ordem em que cada máquina é capaz de processar a tarefa *tar*. Cada vez que uma tarefa é alocada, um novo nó na árvore

Algoritmo 3.3: Pseudo-código para o *Path-relinking* implementado

```
   Procedimento: PathRelinking(solOrig)
 1 solDest ← SoluçãoAleatória(poolDeMelhoresSolucoes);
 2 para m ← 1 até numeroDeMaquinas faça
 3 |   para cada posição l ocupada na máquina m de solDest faça
 4 |   |   se (solOrig.posicao[m][l] ≠ solDest.posicao[m][l]) então
 5 |   |   |   se existe uma tarefa em solOrig.posicao[m][l] então
 6 |   |   |   |   Troca as tarefas solOrig.posicao[m][l] com solDest.posicao[m][l] em
   |   |   |   |   solOrig;
 7 |   |   |   senão
 8 |   |   |   |   Retira a tarefa solDest.posicao[m][l] da posição em que ela estiver em
   |   |   |   |   solOrig e coloca na posição l da máquina m em solOrig;
 9 |   |   |   fim se
10 |   |   fim se
11 |   fim para
12 fim para
```

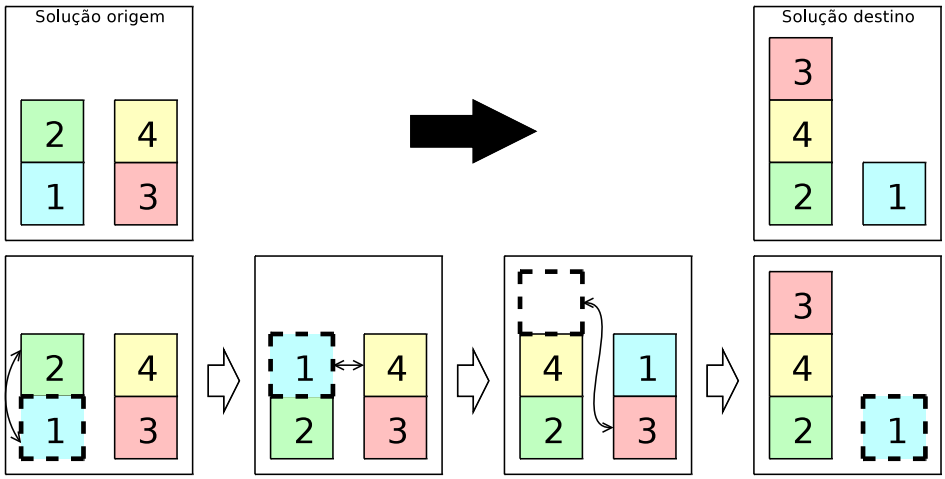


Figura 3.3: Seqüência de passos de PR com as posições analisadas em destaque

de enumeração é criado. A primeira fase é descrita no Algoritmo 3.4, e um exemplo de uma árvore de enumeração parcial para 2 máquinas e 6 tarefas é dado na Figura 3.4.

A segunda fase decide a seqüência de produção em cada máquina. A ordem é decidida para uma máquina de cada vez. Em cada máquina, a primeira tarefa a ser processada é escolhida, depois a segunda, até que não haja mais tarefas alocadas àquela máquina. A função `PróximaTarefaNaMáquina(`m`)` retorna cada tarefa na máquina m seguindo a regra da EDD. Cada vez que uma tarefa é seqüenciada em uma posição de produção específica na máquina, um novo nó na árvore de enumeração é criado. Esta fase é descrita no Algoritmo 3.5, e a Figura 3.5 mostra o exemplo da árvore de enumeração parcial continuando o nó em destaque na Figura 3.4.

Para evitar a enumeração completa da árvore de enumeração, um procedimento de *bounding* é utilizado para encontrar um limite inferior para cada nó. Para um certo nó i, se o limite inferior `Bound(`i`)` for pior que o limite superior, o nó é podado. O procedimento de *bounding* é descrito na próxima seção. A eficiência da poda depende enormemente da qualidade dos

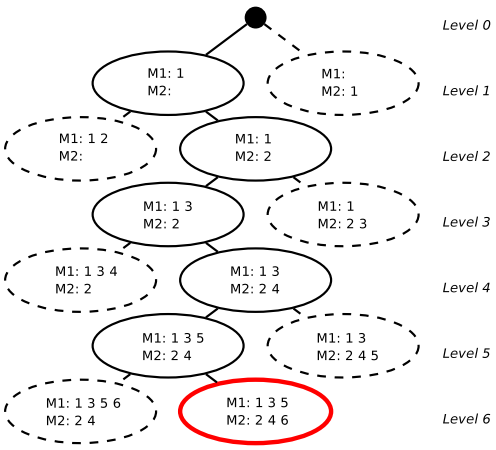


Figura 3.4: Primeira fase de branching

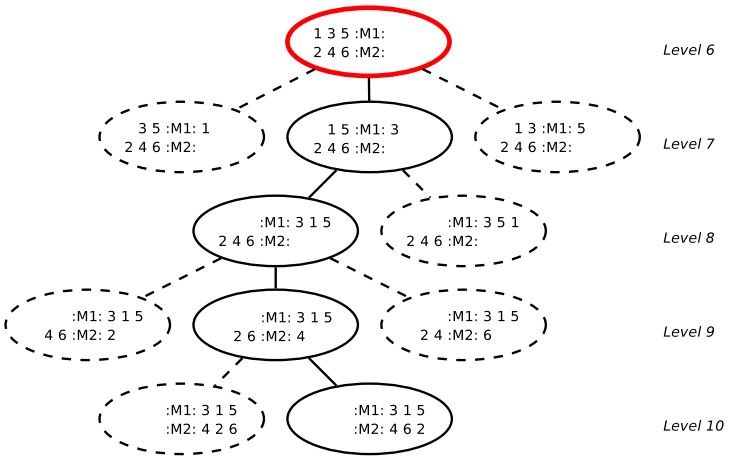


Figura 3.5: Segunda fase de branching

limites superior e inferior.

3.3 O Procedimento de *Bounding*

O procedimento de *bounding* é provavelmente o mais importante em um branch-and-bound. Um limite inferior preciso permite ao algoritmo podar um grande número de nós, eliminando muito processamento desnecessário. Seja:

LI^k: O limite inferior do *makespan*.

LI^t: O limite inferior do atraso ponderado.

LI^{kf}: O limite inferior do *makespan* considerando os tempos de todas as tarefas.

LI_m^{ka1}: O limite inferior do *makespan* considerando uma máquina específica durante a primeira fase de *branching*.

Algoritmo 3.4: Primeira fase de branching

Procedimento: `Branch1(`i`)`

```
1  se TodasAsTarefasEstãoAlocadas() então
2  |  Branch2(1, 1);
3  |  retorna ;
4  fim se

5  tar ← N[i];
6  para cada m ← PróximaMáquina(tar) faça
7  |  AlocaTarefa(tar, m);
8  |  se Bound(solucaoAtual) < ValorDaSolução(Sol) então
9  |  |  Branch1(i + 1);
10 |  fim se
11 |  DesalocaTarefa(tar, m);
12 fim para cada
```

N^u**:** O conjunto de tarefas não-alocadas.

N_m^a**:** O conjunto de tarefas alocadas à máquina m.

S_m^a**:** O conjunto de pares de tarefas $\{ii'\}$ pertencentes ao conjunto de tarefas N_m^a.

u**:** O menor tempo de preparação considerando todos os pares de tarefas em todas as máquinas.

O procedimento de *bounding* deste trabalho é baseado na estratégia de redução de domínio, retirada da área de Programação por Restrições [Lustig e Puget (2001)].

Neste trabalho, o procedimento de *bounding* calcula separadamente os valores de LI^k e LI^t em cada nó. Para calcular LI^k na primeira fase de branching, o procedimento considera duas formas, utilizando apenas o maior valor dentre elas, como mostra a Equação (3.2).

LI^k é formalmente definido nas Equações (3.2-3.3). Na Equação (3.4), LI^{kf} é dado pela soma dos menores tempos de processamento de todas as tarefas não-alocadas e os tempos de processamento das tarefas alocadas ao menor tempo de preparação multiplicado por n, onde n é o número de tempos de preparação necessários para todas as máquinas (neste caso, o número de tarefas subtraído do número de máquinas). LI_m^{ka1} é dado pela soma dos tempos de processamento de cada tarefa alocada à máquina m aos n menores tempos de preparação, onde n é o menor número de tempos de preparação necessário à máquina m (ou seja, o número de tarefas alocadas a m menos um).

$$LI = LI^k + LI^t \tag{3.1}$$

$$LI^k = \max\left(\max_{m \in M}\left(LI_m^{ka}\right), LI^{kf}\right) \tag{3.2}$$

Algoritmo 3.5: Segunda fase de branching

Procedimento: `Branch2`(i, m)

```
1  se SoluçãoEstáCompleta() então
2  |  Sol ← solucaoAtual;
3  |  retorna ;
4  fim se

5  se OrdemEstáDecididaNaMáquina(m) então
6  |  Branch2(1, m + 1);
7  |  retorna ;
8  fim se

9  para tar ← PróximaTarefaNaMáquina(m) faça
10 |  AlocaTarefaÀPosição(tar, m, i);
11 |  se Bound(solucaoAtual) < ValorDaSolução(Sol) então
12 |  |  Branch2(i + 1, m);
13 |  fim se
14 fim para
```

$$LI_m^{ka1} = \sum_{i \in N_m^a} (p_{im}) + \sum_{\{ii'\} \in S_m^a} (s_{ii'm}) \tag{3.3}$$

$$LI^{kf} = \sum_{i \in N^u} \min_{m \in M} (p_{im}) + \frac{\sum_{m \in M} \sum_{i \in N_m^a} p_{im} + n \cdot u}{|M|} \tag{3.4}$$

Durante a segunda fase de branching não existem tarefas não-alocadas. Portanto, as Equações (3.5-3.6) são utilizadas para calcular LI^k. Seja:

LI_m^{ka2}**:** O limite inferior do *makespan* considerando uma máquina específica durante a segunda fase de *branching*.

N_m^{as}**:** O conjunto de tarefas na máquina m seqüenciadas em uma posição específica.

S_m^{as}**:** O conjunto de tempos de preparação usados pelas tarefas em N_m^{as}.

N_m^{au}**:** O conjunto de tarefas na máquina m que ainda não foram seqüenciadas.

S_m^{au}**:** O conjunto dos n menores tempos de preparação entre as tarefas em N_m^{au} e a tarefa na última posição seqüenciada de m, onde n é o número de tempos de preparação necessários para as tarefas em N_m^{au}.

LI_m^{ka2} é dado pela soma do makespan parcial (considerando apenas as posições já seqüenciadas) aos tempos de processamento das tarefas não-seqüenciadas e os menores tempos de preparação entre elas. Isso é calculado somando os tempos de produção das tarefas alocadas à máquina m, aos tempos de preparação utilizados pelas tarefas já seqüenciadas e aos menores tempos de preparação das tarefas não alocadas. Isto é feito para cada máquina, e o maior valor encontrado fornece LI^k.

$$LI^k = \max_{m \in M} \left(LI_m^{ka2} \right) \tag{3.5}$$

$$LI_m^{ka2} = \sum_{i \in N_m^a} (p_{im}) + \sum_{\{ii'\} \in S_m^{as}} (s_{ii'm}) + \sum_{\{ii'm\} \in S_m^{au}} (s_{ii'm}) \tag{3.6}$$

O limite inferior para o atraso ponderado (LI^t) é calculado de forma similar a LI^k. Ele é dado de acordo com as Equações (3.7-3.12). Seja:

LI^{tu}**:** O limite inferior do atraso ponderado considerando apenas as tarefas não-alocadas.

LI_m^{ta}**:** O limite inferior do atraso ponderado considerando apenas as tarefas alocadas à máquina m.

t_i**:** O tempo de início do processamento da tarefa i no seqüenciamento atual.

t_i'**:** O menor tempo de início para a tarefa i como se i fosse a próxima seqüenciada.

t_i''**:** O tempo de início para a tarefa i assumindo um tempo de preparação fixo entre as tarefas.

LI_m^{ta} é dado considerando que as tarefas são processadas seguindo uma ordem não-decrescente de data de entrega, utilizando o tempo de processamento como critério de desempate, e um tempo de preparação fixo entre cada tarefa. Este tempo de preparação é igual ao menor encontrado entre as tarefas não-seqüenciadas da máquina m, e é utilizado para calcular o tempo de início t_i'' de acordo com essa seqüência. Seja:

$$LI^t = LI^{tu} + \sum_{m \in M} LI_m^{ta} \tag{3.7}$$

$$LI^{tu} = \sum_{i \in N^u} \left[w_i \cdot \max \left(\min_{m \in M} (p_{im}) - d_i, 0 \right) \right] \tag{3.8}$$

$$LI_m^{ta} = LI_m^{ta1} + \max \left(LI_m^{ta2}, LI_m^{ta3} \right), \tag{3.9}$$

onde LI_m^{ta1} representa a penalidade de atraso para as tarefas seqüenciadas e t_i é o tempo de início do processamento da tarefa i. O limite inferior para as tarefas não-seqüenciadas é calculado de duas formas diferentes (LI_m^{ta2} e LI_m^{ta3}), e apenas o maior valor é considerado. LI_m^{ta2} considera que todas as tarefas podem ser processadas na próxima posição disponível, e t_i' é o menor tempo de início para a tarefa i. LI_m^{ta3} considera que as tarefas são processadas seguindo uma ordem não-decrescente de data de entrega, utilizando o tempo de processamento como critério de desempate, e um tempo de preparação fixo entre cada tarefa. Este tempo de preparação é igual ao menor encontrado entre as tarefas não-seqüenciadas da máquina m, e é utilizado para calcular o tempo de início t_i'' de acordo com essa seqüência.

$$LI_m^{ta1} = \sum_{i \in N_m^{as}} \left[w_i \cdot \max \left(t_i + p_{im} - d_i, 0 \right) \right] \tag{3.10}$$

$$LI_m^{ta2} = \sum_{i \in N_m^{au}} \left[w_i \cdot \max \left(t_i' + p_{im} - d_i, 0 \right) \right] \tag{3.11}$$

$$LI_m^{ta3} = \min_{i \in N_m^{au}} (w_i) \cdot \sum_{i \in N_m^a} \max \left(t_i'' + p_{im} - d_i, 0 \right) \tag{3.12}$$

É importante destacar algumas diferenças entre estas variáveis durante as primeira e segunda fases de branching. Na primeira fase, t_i' é sempre zero, como não há tarefas seqüenciadas em qualquer posição. Na segunda fase, N^u é sempre vazio, pois cada tarefa já está alocada a alguma máquina. Portanto, LI^{tu} é sempre zero. No entanto, na primeira fase LI^{tu} é maior que zero apenas quando o menor tempo de processamento de alguma tarefa for maior que a sua data de entrega. Pequenas variações no limite inferior de uma solução parcial podem significar grandes quantidades de nós podados. Portanto, essa equação é mantida.

Como se pode perceber, neste trabalho todos os limites inferiores podem ser calculados de forma simples, a maioria com algoritmos $O(n)$.

Algoritmo 3.6: B&B Dicotômico

```
   Procedimento: BranchAndBound()
 1 Sol ← SoluçãoInicial();
 2 LI ← Bound(primeiroNo);
 3 LS ← LI + (ValorDaSolução(Sol)−LI) · d;
 4 enquanto ExisteSoluçãoInteiraEntreLIeLS() faça
 5 |   Branch(primeiroNo);
 6 |   LI ← LS;
 7 |   LS ← LI + (ValorDaSolução(Sol)−LI) · d;
 8 fim enquanto
 9 retorna Sol;

   Procedimento: Branch(no)
10 se SoluçãoCompleta(no) então
11 |   Sol ← no;
12 |   LS ← LI + (ValorDaSolução(Sol)−LI) · d;
13 |   retorna ;
14 fim se
15 para todo n ∈ FilhosDe(no) faça
16 |   se Bound(n) < LS então
17 |   |   Branch(n);
18 |   fim se
19 fim para todo
```

3.4 *Branch-and-Bound* Dicotômico

Além do B&B normal, foi criada uma variação dicotômica. Num algoritmo dicotômico clássico, ao encontrar uma solução melhor que o limite superior atual, o novo limite superior é dado por

$$LS = \frac{LI + \texttt{ValorDaMelhorSolução()}}{2},$$

em vez de ser dado diretamente pelo valor da nova solução. Ao terminar a busca, o limite inferior é atualizado com o valor do limite superior, o limite superior é atualizado com o valor da fórmula acima para o novo limite inferior, e a busca é reiniciada partindo da última solução encontrada. Isso é repetido até que não existam mais soluções inteiras entre LI e LS.

No dicotômico desenvolvido, esse conceito foi estendido. Em vez de o limite superior sempre ser a média do limite inferior e do valor da melhor solução atual, é utilizado um parâmetro d entre 0 e 1 para identificar à qual distância o novo LS está de LI. O limite superior é então dado por

$$LS = LI + (\texttt{ValorDaMelhorSolução()} - LI) \cdot d.$$

Limitar a busca a um limite superior menor que a melhor solução encontrada pode trazer bons resultados se existir uma solução de valor menor que esse limite. Caso contrário, a busca pode se tornar mais extensa, uma vez que a busca é repetida até não haver soluções inteiras

entre LI e LS. Se $d = 0,5$, a versão desenvolvida se torna idêntica à clássica, e se $d = 1$, o B&B deixa de ser dicotômico. Um B&B dicotômico genérico está descrito em maiores detalhes no Algoritmo 3.6. Testes com ambas as versões do B&B são apresentados na Seção 4.8.

Capítulo 4

Testes e Resultados

Resultados são apresentados comparando as soluções encontradas pelo B&B desenvolvido com as soluções encontradas por ambos os modelos quando resolvidos pelo CPLEX 9.0[1] na seção 4.3. Testes com o B&B resolvendo instâncias das várias classes descritas na seção 4.1 são mostrados nas seções seguintes. As classes de instâncias utilizadas neste trabalho estão definidas na seção 4.1.

Para apresentação dos resultados dos testes, é utilizada uma média truncada conhecida como inter-quartis [Wikimedia Foundation (2005)], que descarta 50% dos valores encontrados. Como há 20 instâncias de cada tamanho em cada classe, os 5 menores e os 5 maiores valores encontrados são descartados de cada amostra. A média truncada não é tão sensível a valores muito fora da média como é a média aritmética, mas utiliza mais informação da amostra. Além disso, o desvio padrão é apresentado em forma de porcentagem, calculado pela fórmula

$$\sigma = \left(\frac{100}{\overline{x}} \cdot \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \overline{x})^2} \right) \%,$$

onde x_i representa o valor medido para a instância i, e $\overline{x}$ representa a média aritmética de todos os x_i.

Durante os testes, busca-se variar o número de tarefas de apenas uma tarefa de uma instância para outra. No entanto, um problema com n tarefas não é sempre mais fácil de resolver que um com $n+1$ tarefas. Dependendo do número de máquinas e de como as tarefas estão nelas distribuídas, o último pode ser bem mais fácil de resolver que o primeiro. Nos gráficos apresentados, pode-se verificar algumas perturbações desse tipo.

O número de iterações do GRASP foi fixado em 1.000 para calcular o limite superior, a não ser quando for especificado o contrário, e todos os gráficos são mostrados em forma logarítmica. Para os testes do B&B, o tempo máximo de resolução foi fixo em 7200s (duas horas). Os resultados são apresentados até o número de tarefas em que o B&B conseguiu resolver, de forma que pelo menos 15 instâncias de cada conjunto de 20 não ultrapassaram o

[1] *CPLEX Software*: Um otimizador para problemas lineares desenvolvido pela ILOG. Mais informações no sítio: *http://www.ilog.com/products/cplex/*

tempo. As instâncias que ultrapassaram o tempo máximo, foram desconsideradas pela média inter-quartis.

Todos os testes foram realizados em computadores com processadores Pentium 4 de 2.4GHz e com 1GB de memória RAM. O sistema operacional em todos os casos foi Linux distribuição Debian, com kernel 2.6.16. Os algoritmos foram implementados em C e compilados com o gcc versão 4.1.2.

4.1 Instâncias

Para analisar os algoritmos e modelos desenvolvidos para este problema, várias classes de instâncias são definidas. Em cada classe há uma mudança em um dos dados de entrada. Todos os valores são gerados aleatoriamente utilizando uma distribuição uniforme. Os seus valores padrão das instâncias estão listados na Tabela 4.1, onde $U(x, y)$ é um valor gerado de uma distribuição uniforme entre x e y.

Para gerar as datas de entrega, várias variações das fórmulas encontradas em outros trabalhos [Lopes e de Carvalho (2007); Ho e Chang (2006)] foram testadas. Para garantir que a razão $\frac{\text{atraso ponderado}}{\text{solução total}}$ permanecesse a mesma para todos os tamanhos de instância de cada classe, o seguinte algoritmo foi utilizado para calcular o valor máximo das datas de entrega: na ordem de geração, cada tarefa é alocada para a máquina capaz de terminá-la primeiro. O makespan dessa solução é chamado de h. O maior valor para a data de entrega é dado por

$$\frac{2 \cdot h}{q},$$

onde q indica o nível de congestionamento do sistema de seqüenciamento [Lopes e de Carvalho (2007)]. Quanto maior o valor de q, mais congestionado o sistema será, e mais atraso haverá.

Dado	Valor padrão
Tempo de processamento	$U(5, 200)$
Tempo de preparação	$U(25, 50)$
Prioridade	$U(1, 3)$
Data de entrega	$U\left(\text{maior tempo de processamento}, \frac{2 \cdot h}{q}\right)$
q	1

Tabela 4.1: Valores padrão para as instâncias

As classes criadas são:

A: Contém o padrão para todos os valores, como definido na Tabela 4.1.

B: As datas de entrega são decrescidas ($q = 2$).

C: As datas de entrega são decrescidas ($q = 3$).

D: As datas de entrega são decrescidas ($q = 4$).

E: As datas de entrega são decrescidas ($q = 5$).

F: O tempo de processamento é levemente decrescido ($p = U(5, 150)$).

G: O tempo de processamento é bastante decrescido ($p = U(5, 100)$).

H: O tempo de preparação é levemente aumentado ($s = U(25, 100)$).

I: O tempo de preparação é bastante aumentado ($s = U(25, 150)$).

Algoritmo 4.1: Floyd-Warshall modificado

```
   /* S:  uma matriz com os tempos de preparação de uma máquina        */
   /* P:  um vetor com os tempos de processamento de uma máquina       */
   Procedimento: Floyd-Warshall(S, P)
1  n ← S.linhas;
2  para k ← 1 até n faça
3  |  para i ← 1 até n faça
4  |  |  para j ← 1 até n faça
5  |  |  |  s_ij ← Min(s_ij, s_ik + p_k + s_kj);
6  |  |  fim para
7  |  fim para
8  fim para
```

São geradas instâncias com 4 máquinas e 4 a 12 tarefas, e com 6 máquinas e 6 a 30 tarefas. Para cada tamanho de problema, 20 instâncias são geradas aleatoriamente utilizando sementes diferentes.

Para que o primeiro modelo e o B&B sejam válidos, os tempos de processamento e preparação devem satisfazer a desigualdade triangular $s_{ij} \leq s_{ik} + p_k + s_{kj}$. Em algumas linhas de produção, especialmente na indústria química, o processamento da tarefa k pode ser parte da preparação da tarefa i para j. Com isso em mente, o tempo de processamento da tarefa k foi incluído na desigualdade triangular.

Como os tempos de preparação foram gerados aleatoriamente, eles precisam ser corrigidos para satisfazer a desigualdade. Para tanto, o algoritmo de Floyd-Warshall (F-W) foi modificado como mostrado no Algoritmo 4.1 para incluir um peso em cada nó a ser considerado assim como o peso do arco. O algoritmo original de F-W é utilizado para encontrar os caminhos mais curtos entre todos os pares de nós em um grafo direcionado [Cormen et al. (1990)].

Um arquivo de instância com 6 máquinas e 20 tarefas é mostrado no Apêndice A. Todos os arquivos com as instâncias podem ser encontrados no sítio do *Grupo de Seqüenciamento da UFMG*[2].

4.2 Avaliando GRASP como limite superior

Nesta seção, tentamos estabelecer um número de iterações ideal para o GRASP alcançar um bom limite superior sem prejudicar muito o algoritmo. Foram feitos testes com GRASP a

[2] *http://www.dcc.ufmg.br/laboratorios/lapo/scheduling/*

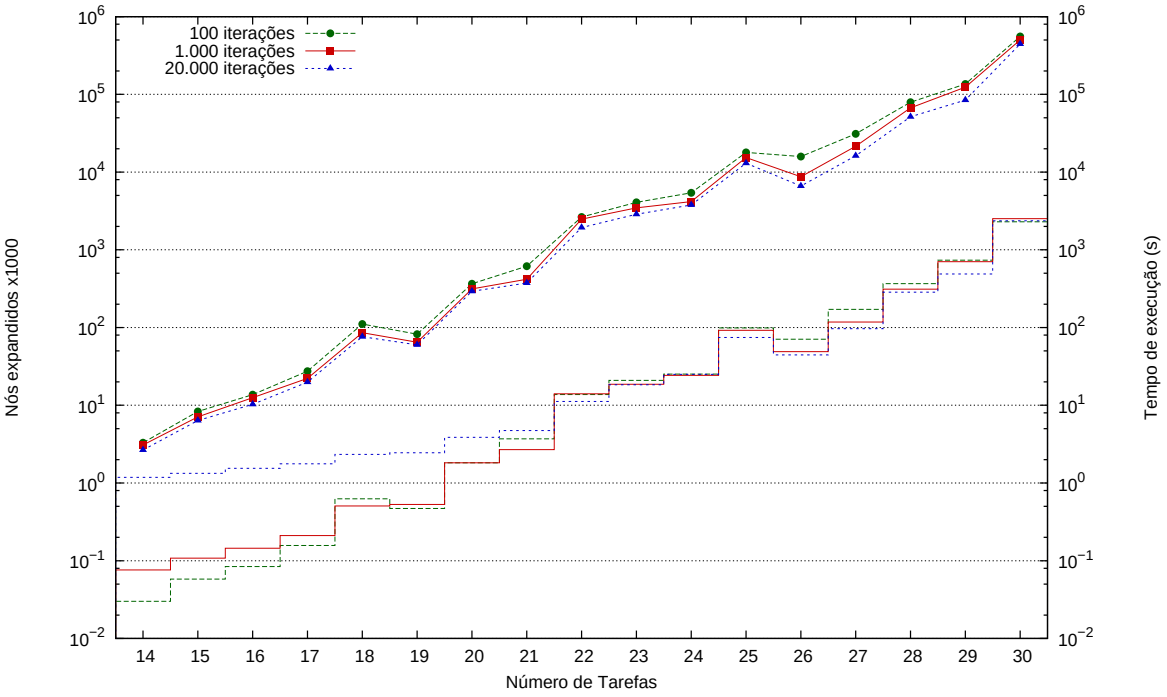


Figura 4.1: Número de nós expandidos variando o número de iterações do GRASP

100, 1.000 e 20.000 iterações. Os resultados são exibidos na Figura 4.1, onde o número de iterações é dado pelos pontos, e o tempo total de processamento é dado pelos pulsos. Além disso, ambos os dados são detalhados na Tabela 4.2.

Como esperado, o número de nós expandidos pelo B&B decresce quando se aumenta o número de iterações do GRASP, já que quanto mais iterações o GRASP realiza, maior a chance de melhorar o limite superior. Com 20.000 iterações e um pequeno número de tarefas (até 21 tarefas), o GRASP é responsável por grande parte do tempo total do algoritmo. Por

Tarefas	Dado	100 iterações	1.000 iterações	20.000 iterações
20	Nós	365.701,70 ± 14,70	313.626,80 ± 18,77	292.325,10 ± 17,87
	Tempo	1,82 ± 14,60	1,82 ± 18,47	3,87 ± 07,55
21	Nós	614.564,30 ± 11,14	417.956,20 ± 09,49	373.644,30 ± 08,72
	Tempo	3,69 ± 10,54	2,69 ± 09,69	4,73 ± 04,67
22	Nós	2.639.011,90 ± 38,12	2.484.061,60 ± 37,60	1.931.584,80 ± 33,98
	Tempo	13,75 ± 37,74	14,03 ± 37,35	11,18 ± 17,98
23	Nós	4.082.351,60 ± 11,89	3.466.514,30 ± 16,45	2.870.462,50 ± 18,33
	Tempo	20,93 ± 11,94	18,62 ± 13,24	18,33 ± 13,36
24	Nós	5.420.806,80 ± 20,55	4.173.514,90 ± 24,58	3.781.834,60 ± 26,23
	Tempo	25,23 ± 20,91	24,26 ± 22,88	25,03 ± 21,38
25	Nós	17.895.802,20 ± 14,37	15.413.490,80 ± 19,07	13.009.041,90 ± 14,51
	Tempo	98,86 ± 17,28	92,26 ± 20,32	74,52 ± 10,44
26	Nós	15.830.415,30 ± 19,39	8.705.022,80 ± 20,38	6.611.487,20 ± 20,62
	Tempo	70,80 ± 19,18	49,02 ± 20,91	44,48 ± 19,12
27	Nós	31.013.535,40 ± 19,83	21.481.047,50 ± 15,11	16.211.319,40 ± 11,13
	Tempo	171,28 ± 25,03	117,66 ± 15,42	96,76 ± 12,14
28	Nós	79.212.011,50 ± 17,45	67.076.597,60 ± 18,59	51.486.377,80 ± 19,21
	Tempo	366,93 ± 18,95	312,30 ± 16,78	284,95 ± 18,74
29	Nós	135.907.198,80 ± 24,96	123.603.675,80 ± 22,78	84.326.777,10 ± 24,94
	Tempo	735,25 ± 26,21	703,04 ± 24,53	488,97 ± 23,98
30	Nós	556.664.797,30 ± 12,41	500.890.594,50 ± 14,66	444.185.042,80 ± 16,31
	Tempo	2.298,87 ± 18,12	2.514,70 ± 18,66	2.365,09 ± 20,49

Tabela 4.2: Número de nós expandidos e desvio padrão variando número de iterações do GRASP

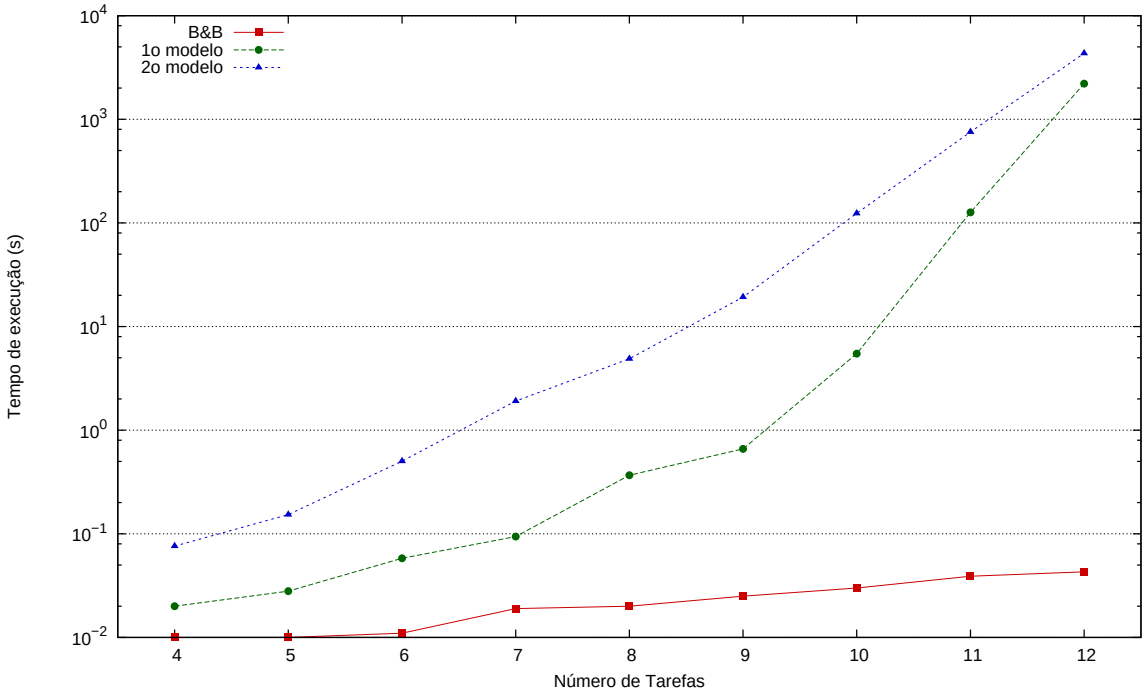


Figura 4.2: Modelos vs. B&B

isso, apesar de o problema ser resolvido expandindo um número menor de nós, o tempo total é superior ao GRASP com 1.000 iterações. A partir de 22 tarefas, o GRASP com 20.000 iterações passa a ser mais interessante, pois o tempo total gasto pelo algoritmo passa a ser menor que GRASP com 1.000 iterações.

4.3 Comparando os modelos

Neste teste, é feita a comparação entre o tempo de CPU gasto pelos dois modelos e o B&B para encontrar a solução ótima. Ambos os modelos são resolvidos pelo CPLEX, e a mesma solução inicial encontrada pelo GRASP é usada tanto no B&B quanto pelo CPLEX. As instâncias de classe A para 4 máquinas são utilizadas neste teste. Os resultados são apresentados na Figura 4.2 e detalhados na Tabela 4.3.

O primeiro modelo tem um desempenho próximo do B&B nas instâncias menores. Assim que o número de tarefas cresce, também cresce a diferença entre eles. O desvio padrão do

Tarefas	MIP baseado em Manne's Média ± Desvio (%)	MIP baseado em Wagner's Média ± Desvio (%)	Branch and Bound Média ± Desvio (%)
4	0,02 ± 00,00	0,08 ± 04,24	0,01 ± 00,00
5	0,03 ± 06,79	0,15 ± 07,10	0,01 ± 00,00
6	0,06 ± 11,64	0,50 ± 08,71	0,01 ± 08,64
7	0,09 ± 17,03	1,91 ± 06,84	0,02 ± 05,00
8	0,37 ± 31,45	4,89 ± 09,06	0,02 ± 00,00
9	0,66 ± 27,73	19,24 ± 10,28	0,03 ± 06,32
10	5,48 ± 43,33	123,72 ± 10,18	0,03 ± 00,00
11	126,42 ± 55,83	755,09 ± 19,16	0,04 ± 02,44
12	2.209,30 ± 48,73	4.337,65 ± 10,69	0,04 ± 03,37

Tabela 4.3: Tempo de CPU gasto e variação dos dois modelos e do B&B

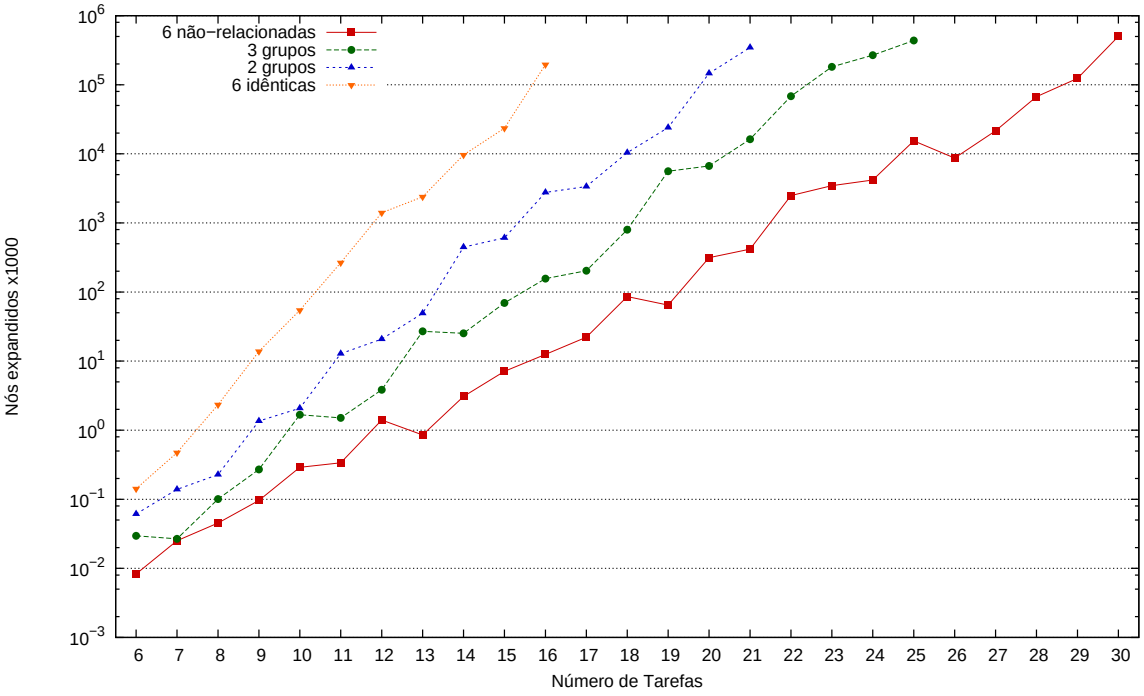


Figura 4.3: Número de nós expandidos variando o cenário de máquinas com o B&B

primeiro e do segundo modelo também cresce com o número de tarefas. O segundo modelo tem um desempenho pior que o primeiro e que o B&B, mas apresenta uma variação mais estável. Ele tem um desvio padrão menor que o primeiro modelo nas instâncias maiores. O tempo de CPU gasto pelo primeiro modelo começa a se aproximar do tempo gasto pelo segundo modelo quando o número de tarefas cresce. Isso nos leva a crer que o segundo modelo terá um desempenho melhor que o primeiro para instâncias maiores, mas não foi possível comprovar isso devido ao grande esforço computacional que seria necessário. O B&B chegou às mesmas soluções ótimas obtidas pelos modelos.

4.4 Diferentes Cenários de Máquinas

Neste conjunto de testes, é feita uma variação no número de máquinas iguais, para verificar se resolver um problema num cenário de máquinas iguais é mais fácil ou mais difícil para o B&B. São feitos testes com as instâncias da classe A considerando 6 máquinas idênticas, 2 conjuntos de 3 máquinas iguais em cada, 3 conjuntos de 2 máquinas iguais em cada, e em 6 máquinas não relacionadas.

Para melhorar o desempenho do algoritmo no caso de máquinas iguais, o B&B sofre uma pequena alteração. Quando duas máquinas são iguais, a solução que tem as mesmas tarefas na primeira máquina que uma outra solução tem na segunda máquina e vice-versa, são soluções simétricas. Por exemplo, com duas máquinas iguais (A e B) e três tarefas (1, 2 e 3), a solução com as tarefas 1 e 2 em A, e 3 em B é simétrica à solução com 1 e 2 em B, e 3 em A. Para eliminar um pouco da simetria em soluções, a segunda fase de branching só é executada se

o número de tarefas de uma máquina for maior ou igual ao número de tarefas nas máquinas iguais a ela de menor índice. Os resultados são exibidos na Figura 4.3 e detalhados na Tabela 4.4.

Os casos com maior número de máquinas diferentes foram resolvidos mais facilmente pelo B&B. À medida em que o número de máquinas diferentes diminui, as soluções vão ficando cada vez mais parecidas, e se torna cada vez mais difícil podar um nó. A Figura 4.4 mostra a variação no valor da solução no decorrer da execução do GRASP para uma instância de 30 tarefas. É possível ver como as soluções com 6 máquinas idênticas se concentram numa estreita faixa de valores, enquanto com 6 máquinas não-relacionadas as soluções se espalham em valores mais variados.

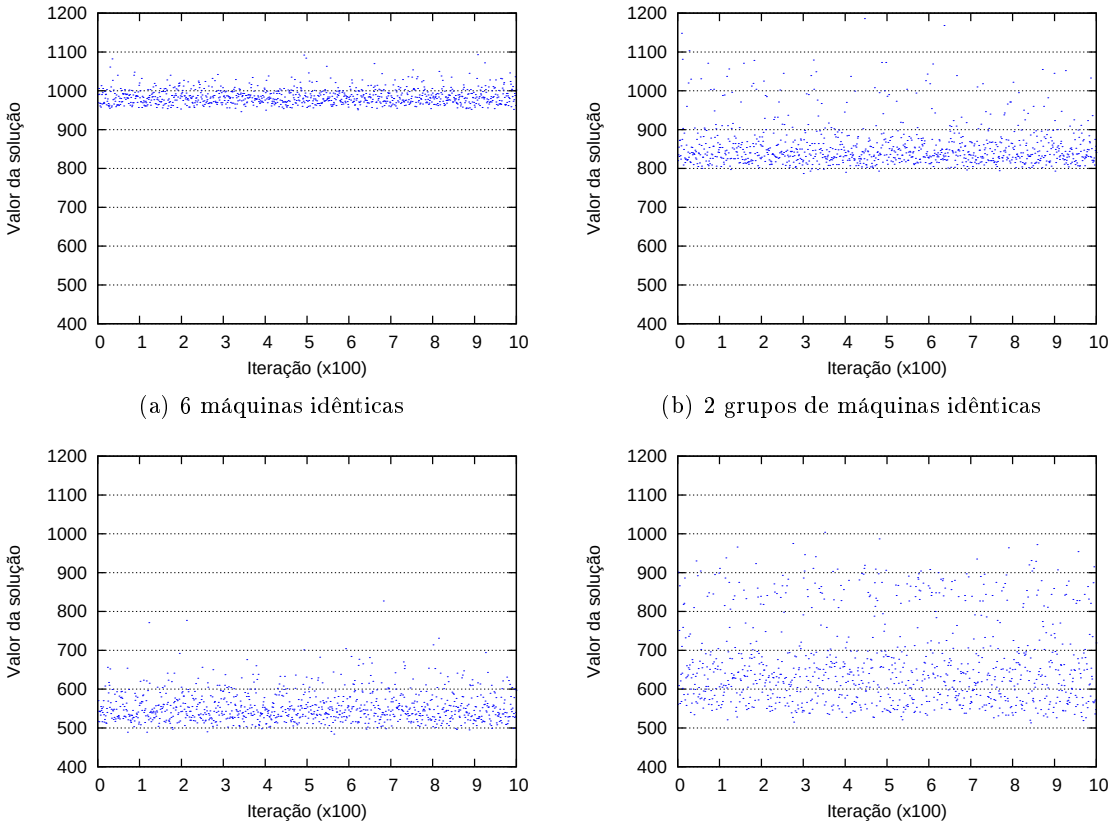
4.5 Variando a Data de Entrega

Neste teste é feita uma avaliação de como a variação das datas de entrega pode influenciar na execução do B&B. As instâncias de classes A a E para 6 máquinas são utilizadas neste teste. Os resultados são exibidos na Figura 4.5 e detalhados na Tabela 4.5.

As instâncias da classe E (datas de entrega muito apertadas) são as mais difíceis de serem resolvidas pelo algoritmo, seguidas pela classe D. A medida em que o valor de q aumenta, a dificuldade para o B&B resolver o problema também aumenta. É possível pensar que o oposto deveria acontecer, já que com datas de entrega mais apertadas, o número de soluções interessantes diminui, fazendo com que o universo de soluções pesquisadas seja menor. Mas ao dar uma olhada melhor no procedimento de bounding no B&B descrito na seção 3.3, é possível entender porque isso não ocorre. A fórmula utilizada para calcular o limite inferior do atraso ponderado utiliza duas formas diferentes, ficando com o maior resultado, mas nenhum dos dois é um limite bastante forte. No entanto, foram as formas encontradas para garantir a otimalidade da solução.

Tarefas	6 Não-relacionadas	3 Grupos	2 Grupos	6 Idênticas
6	8,30 ± 12,29	29,60 ± 16,76	61,30 ± 19,71	142,00 ± 52,63
7	25,10 ± 08,57	26,80 ± 17,28	139,00 ± 12,83	473,80 ± 38,81
8	45,30 ± 09,62	100,60 ± 19,84	227,50 ± 23,98	2.333,20 ± 29,50
9	96,20 ± 08,80	270,20 ± 15,39	1.355,70 ± 23,45	13.812,40 ± 26,56
10	290,00 ± 16,02	1.671,50 ± 07,83	2.080,90 ± 16,95	54.461,80 ± 11,73
11	337,10 ± 12,90	1.506,00 ± 15,21	12.815,30 ± 14,51	265.839,60 ± 06,51
12	1.401,80 ± 27,02	3.836,00 ± 10,71	20.787,60 ± 14,90	1.403.424,00 ± 14,74
13	851,90 ± 24,26	26.935,10 ± 15,04	49.226,50 ± 20,05	2.383.392,60 ± 12,14
14	3.079,40 ± 11,43	25.211,30 ± 21,91	445.592,40 ± 13,21	9.653.159,90 ± 08,29
15	7.098,50 ± 12,79	69.454,20 ± 18,64	608.657,60 ± 14,61	23.501.765,40 ± 12,54
16	12.515,00 ± 13,04	156.116,50 ± 11,87	2.756.850,20 ± 16,09	196.133.567,60 ± 05,60
17	22.207,00 ± 25,60	203.290,30 ± 16,83	3.364.578,70 ± 16,74	
18	85.882,90 ± 13,36	797.193,80 ± 25,65	10.404.258,30 ± 18,11	
19	64.742,90 ± 21,77	5.609.306,30 ± 35,03	23.995.024,90 ± 12,20	
20	313.626,80 ± 18,77	6.668.757,20 ± 18,29	145.984.498,40 ± 17,58	
21	417.956,20 ± 09,49	16.233.977,00 ± 15,35	345.447.845,00 ± 17,48	
22	2.484.061,60 ± 37,60	68.117.603,60 ± 11,88		
23	3.466.514,30 ± 16,45	181.622.763,10 ± 27,56		
24	4.173.514,90 ± 24,58	268.240.801,60 ± 17,80		
25	15.413.490,80 ± 19,07	437.526.495,20 ± 18,75		

Tabela 4.4: Número de nós expandidos e desvio padrão variando o cenário de máquinas com o B&B



(a) 6 máquinas idênticas

(b) 2 grupos de máquinas idênticas

(c) 3 grupos de máquinas idênticas

(d) 6 máquinas não-relacionadas

Figura 4.4: Soluções encontradas pelo GRASP

4.6 Variando o Tempo de Processamento

Nesta seção é feita uma avaliação de como a variação dos tempos de processamento pode influenciar na execução do B&B. As instâncias das classes A, F e G são utilizadas. Os resultados são exibidos na Figura 4.6 e detalhados na Tabela 4.6.

As instâncias com tempos de processamento variados em maior intervalo foram as mais fáceis de serem resolvidas pelo B&B. Acreditamos que isso tenha acontecido porque o espaço de busca de soluções interessantes seja diminuído quando a diferença entre os tempos de processamento de uma tarefa em duas máquinas diferentes seja muito grande. Quando uma tarefa é alocada a uma máquina onde o seu tempo de processamento é muito maior que nas outras máquinas, o limite inferior sobe bastante. É fácil perceber que quanto maior o tempo de processamento, mais rapidamente o limite irá crescer, e mais rápido o nó será podado.

4.7 Variando o Tempo de Preparação

Aqui, é feita uma avaliação de como a variação dos tempos de preparação pode influenciar na execução do B&B. As instâncias das classes A, H e I são utilizadas. Os resultados são exibidos na Figura 4.7 e detalhados na Tabela 4.7.

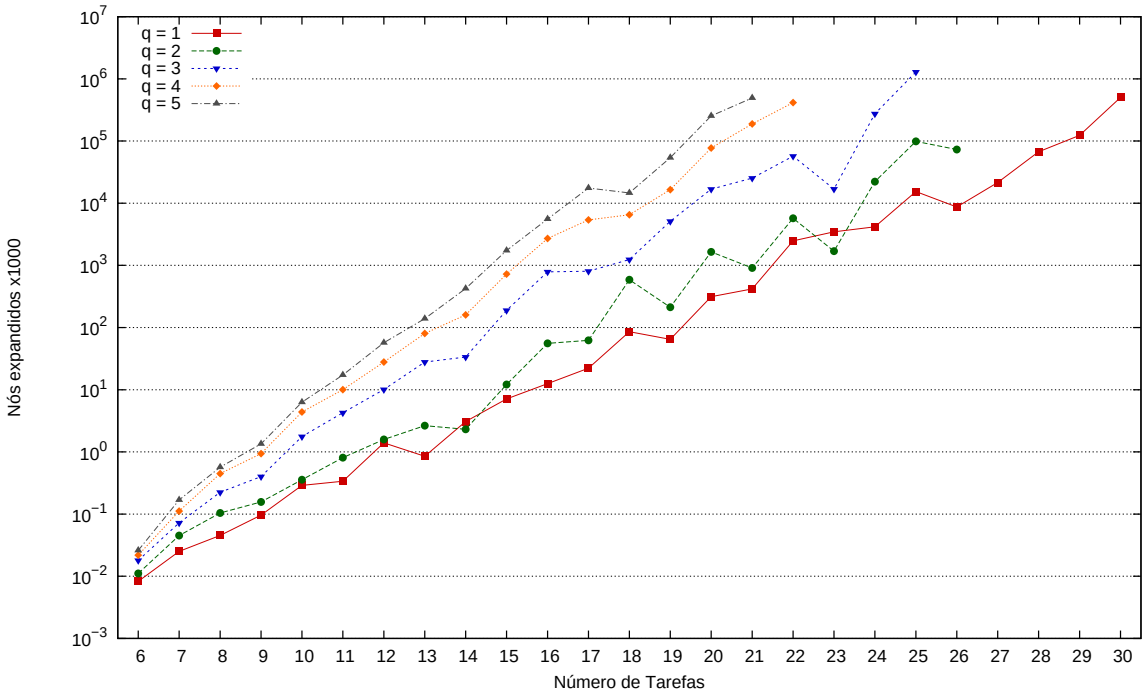


Figura 4.5: Número de nós expandidos variando a data de entrega com o B&B

As instâncias com a menor variação entre os tempos de preparação são as mais fáceis de serem resolvidas pelo B&B. É possível ver claramente um limite inferior fraco prejudicando o desempenho do algoritmo. Como explicado na Seção 3.3, o procedimento de bounding considera apenas os menores tempos de preparação para calcular o limite inferior. Quando o tempo de preparação varia em um intervalo muito grande, o limite inferior é enfraquecido, prejudicando o desempenho do B&B.

Tarefas	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$
6	0,01 ± 12,29	0,01 ± 09,28	0,02 ± 08,00	0,02 ± 08,17	0,03 ± 11,65
7	0,03 ± 08,57	0,05 ± 09,45	0,07 ± 10,06	0,11 ± 09,87	0,17 ± 16,50
8	0,05 ± 09,62	0,10 ± 09,13	0,22 ± 08,54	0,45 ± 08,99	0,57 ± 11,07
9	0,10 ± 08,80	0,16 ± 08,26	0,40 ± 19,23	0,94 ± 17,74	1,35 ± 14,89
10	0,29 ± 16,02	0,36 ± 10,82	1,76 ± 13,73	4,39 ± 13,12	6,29 ± 11,18
11	0,34 ± 12,90	0,81 ± 13,96	4,27 ± 19,09	10,01 ± 17,35	17,26 ± 16,48
12	1,40 ± 27,02	1,59 ± 12,45	10,06 ± 24,83	27,84 ± 21,27	56,89 ± 23,58
13	0,85 ± 24,26	2,64 ± 14,42	27,84 ± 18,86	80,54 ± 20,76	139,32 ± 17,70
14	3,08 ± 11,43	2,31 ± 09,91	33,55 ± 34,96	160,03 ± 27,31	423,67 ± 15,32
15	7,10 ± 12,79	12,14 ± 25,37	188,58 ± 24,25	723,00 ± 24,29	1.730,28 ± 16,21
16	12,52 ± 13,04	55,68 ± 32,46	787,28 ± 24,04	2.708,92 ± 26,24	5.573,77 ± 20,50
17	22,21 ± 25,60	62,35 ± 18,76	805,27 ± 13,45	5.378,69 ± 18,06	17.553,58 ± 29,82
18	85,88 ± 13,36	586,94 ± 34,38	1.239,95 ± 20,37	6.512,63 ± 18,04	14.612,31 ± 31,42
19	64,74 ± 21,77	212,17 ± 35,83	5.110,47 ± 39,54	16.483,99 ± 43,63	54.040,47 ± 25,83
20	313,63 ± 18,77	1.647,26 ± 37,72	16.785,00 ± 12,85	77.094,38 ± 15,29	253.402,95 ± 18,45
21	417,96 ± 09,49	909,43 ± 29,46	25.281,12 ± 31,13	188.603,33 ± 34,25	492.198,01 ± 21,53
22	2.484,06 ± 37,60	5.711,46 ± 27,28	57.002,79 ± 23,09	414.051,02 ± 38,34	
23	3.466,51 ± 16,45	1.694,49 ± 14,02	16.912,86 ± 20,03		
24	4.173,51 ± 24,58	22.185,30 ± 24,51	273.833,11 ± 29,16		
25	15.413,49 ± 19,07	98.638,25 ± 31,30	1.291.340,02 ± 11,66		

Tabela 4.5: Número de nós expandidos e desvio padrão variando a data de entrega com o B&B

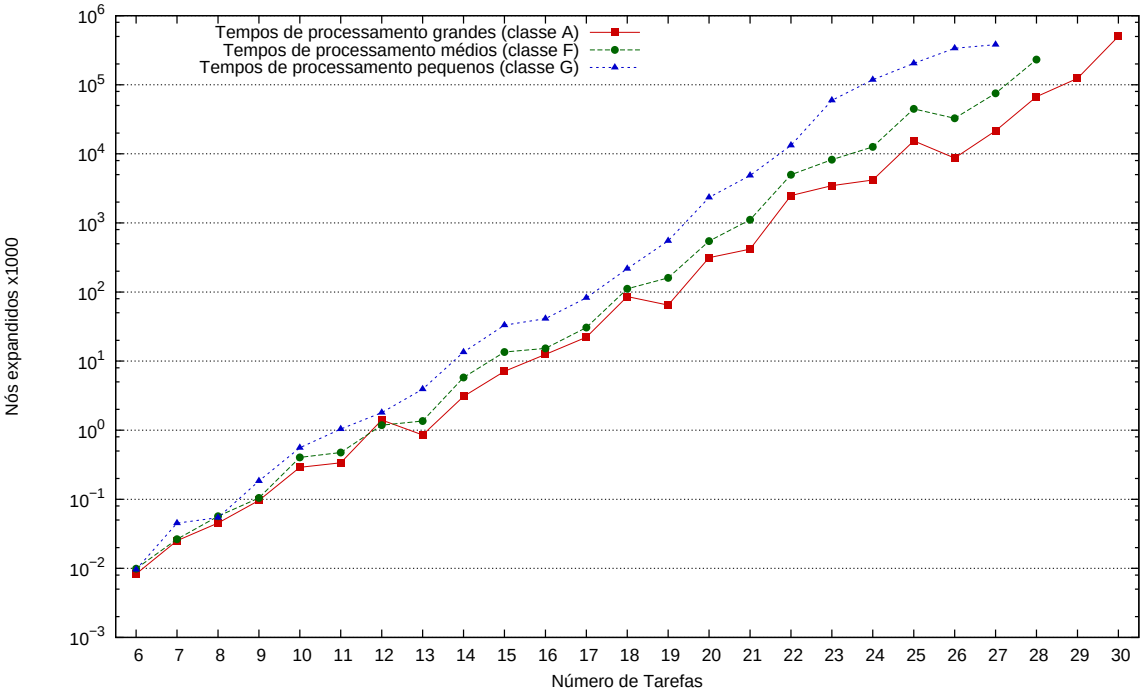


Figura 4.6: Número de nós expandidos variando os tempos de processamento com o B&B

4.8 *Branch-and-bound* Dicotômico

Nesta seção apresentamos testes variando o parâmetro d no B&B dicotômico. Foram utilizadas as instâncias padrão (classe A), e os valores testados para d foram $0,75$, $0,50$ e $0,25$, além da versão não-dicotômica. O resultado é mostrado na Figura 4.8 e detalhado na Tabela 4.8.

Na grande maioria dos testes, os melhores resultados foram obtidos pela versão não-dicotômica, seguida pelos valores $0,75$, $0,50$ e $0,25$ nessa ordem. Isso pode ser explicado pelo

Tarefas	Intervalo de variação pequeno	Intervalo de variação médio	Intervalo de variação grande
6	9,60 ± 08,65	9,90 ± 08,89	8,30 ± 12,29
7	45,20 ± 07,90	26,40 ± 11,82	25,10 ± 08,57
8	54,30 ± 07,97	56,80 ± 11,11	45,30 ± 09,62
9	183,60 ± 10,86	104,70 ± 10,90	96,20 ± 08,80
10	555,80 ± 13,86	403,80 ± 17,30	290,00 ± 16,02
11	1.039,60 ± 14,68	475,30 ± 13,44	337,10 ± 12,90
12	1.798,30 ± 10,68	1.181,30 ± 18,21	1.401,80 ± 27,02
13	3.919,10 ± 10,21	1.359,60 ± 15,48	851,90 ± 24,26
14	13.456,20 ± 18,45	5.783,90 ± 11,49	3.079,40 ± 11,43
15	33.155,50 ± 17,70	13.507,40 ± 14,56	7.098,50 ± 12,79
16	41.198,20 ± 11,92	15.272,30 ± 13,17	12.515,00 ± 13,04
17	82.378,80 ± 12,07	30.639,30 ± 17,11	22.207,00 ± 25,60
18	217.460,20 ± 26,50	111.445,10 ± 24,69	85.882,90 ± 13,36
19	550.450,20 ± 13,36	160.188,50 ± 21,41	64.742,90 ± 21,77
20	2.332.921,90 ± 09,57	545.886,50 ± 13,38	313.626,80 ± 18,77
21	4.861.076,50 ± 08,82	1.108.068,80 ± 14,77	417.956,20 ± 09,49
22	13.243.316,00 ± 12,54	4.977.339,80 ± 24,93	2.484.061,60 ± 37,60
23	59.518.655,00 ± 11,41	8.229.626,10 ± 13,13	3.466.514,30 ± 16,45
24	117.927.263,60 ± 18,52	12.583.325,90 ± 24,79	4.173.514,90 ± 24,58
25	204.010.031,10 ± 06,74	44.800.951,40 ± 11,47	15.413.490,80 ± 19,07
26	336.468.339,50 ± 16,44	32.762.384,40 ± 19,34	8.705.022,80 ± 20,38
27	380.854.795,60 ± 11,78	75.124.625,20 ± 10,75	21.481.047,50 ± 15,11
28		231.865.044,80 ± 11,56	67.076.597,60 ± 18,59

Tabela 4.6: Número de nós expandidos e desvio padrão variando os tempos de processamento com o B&B

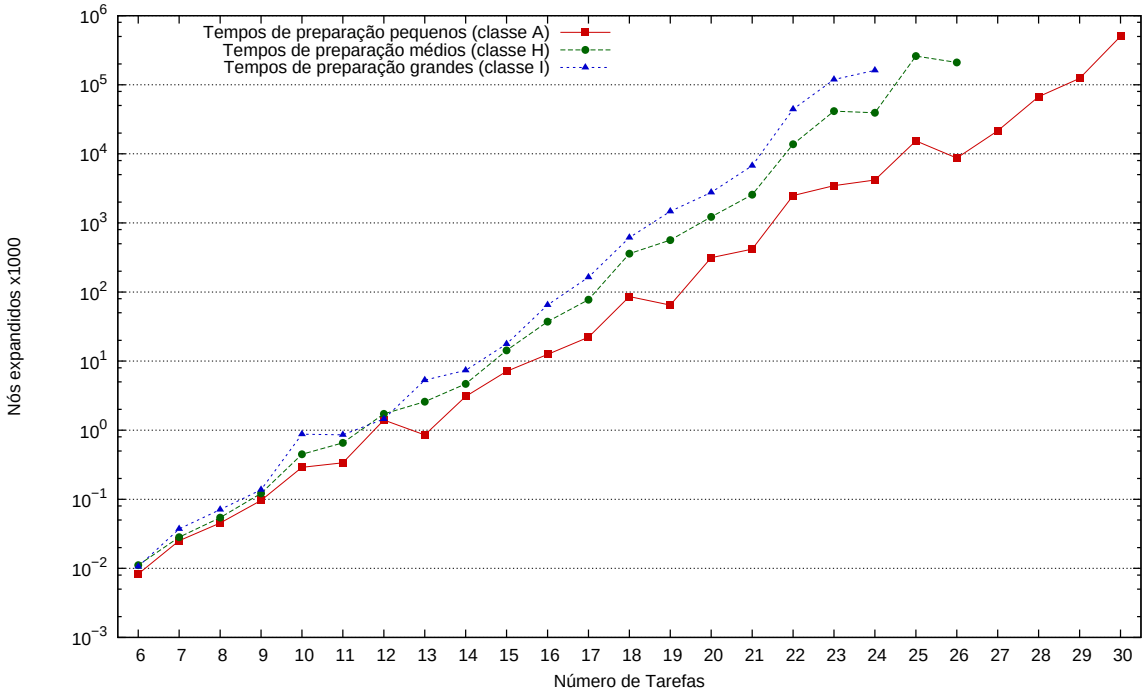


Figura 4.7: Número de nós expandidos variando os tempos de preparação com o B&B

fato de o GRASP obter uma solução próxima da ótima como limite superior. Assim, o B&B não consegue melhorar muito a solução, e os cortes feitos pelo limite superior dicotômico acabam prejudicando o algoritmo. À medida em que número de tarefas aumenta, o desempenho das versões se aproxima. Nas maiores instâncias a versão dicotômica tem um desempenho melhor que a original. Quando o número de tarefas é muito grande, a solução encontrada pelo GRASP se encontra mais distante do ótimo, então os cortes realizados pelo dicotômico podem fazer uma grande diferença.

Tarefas	Intervalo de variação pequeno	Intervalo de variação médio	Intervalo de variação grande
6	8,30 ± 12,29	11,10 ± 11,80	10,60 ± 10,09
7	25,10 ± 08,57	28,20 ± 13,76	37,30 ± 11,26
8	45,30 ± 09,62	54,20 ± 10,66	70,90 ± 08,24
9	96,20 ± 08,80	121,80 ± 13,27	137,30 ± 13,66
10	290,00 ± 16,02	448,30 ± 14,97	874,30 ± 14,17
11	337,10 ± 12,90	654,60 ± 17,83	855,20 ± 10,08
12	1.401,80 ± 27,02	1.718,90 ± 16,13	1.449,00 ± 14,19
13	851,90 ± 24,26	2.586,40 ± 26,94	5.299,10 ± 15,08
14	3.079,40 ± 11,43	4.681,60 ± 17,23	7.342,70 ± 12,51
15	7.098,50 ± 12,79	14.308,80 ± 15,69	17.668,60 ± 12,33
16	12.515,00 ± 13,04	37.197,50 ± 19,28	64.910,70 ± 10,67
17	22.207,00 ± 25,60	77.480,50 ± 19,46	163.406,40 ± 12,31
18	85.882,90 ± 13,36	359.674,80 ± 17,86	613.709,80 ± 21,10
19	64.742,90 ± 21,77	563.697,60 ± 11,78	1.468.219,50 ± 13,93
20	313.626,80 ± 18,77	1.223.759,50 ± 14,87	2.766.689,50 ± 11,22
21	417.956,20 ± 09,49	2.560.609,50 ± 10,84	6.722.599,30 ± 08,48
22	2.484.061,60 ± 37,60	13.747.178,30 ± 29,18	44.148.887,90 ± 24,78
23	3.466.514,30 ± 16,45	41.486.062,30 ± 16,92	119.185.197,40 ± 17,40
24	4.173.514,90 ± 24,58	39.287.762,20 ± 17,98	161.331.684,90 ± 26,47
25	15.413.490,80 ± 19,07	260.215.776,90 ± 16,68	
26	8.705.022,80 ± 20,38	210.029.911,00 ± 28,07	

Tabela 4.7: Número de nós expandidos e desvio padrão variando os tempos de preparação com o B&B

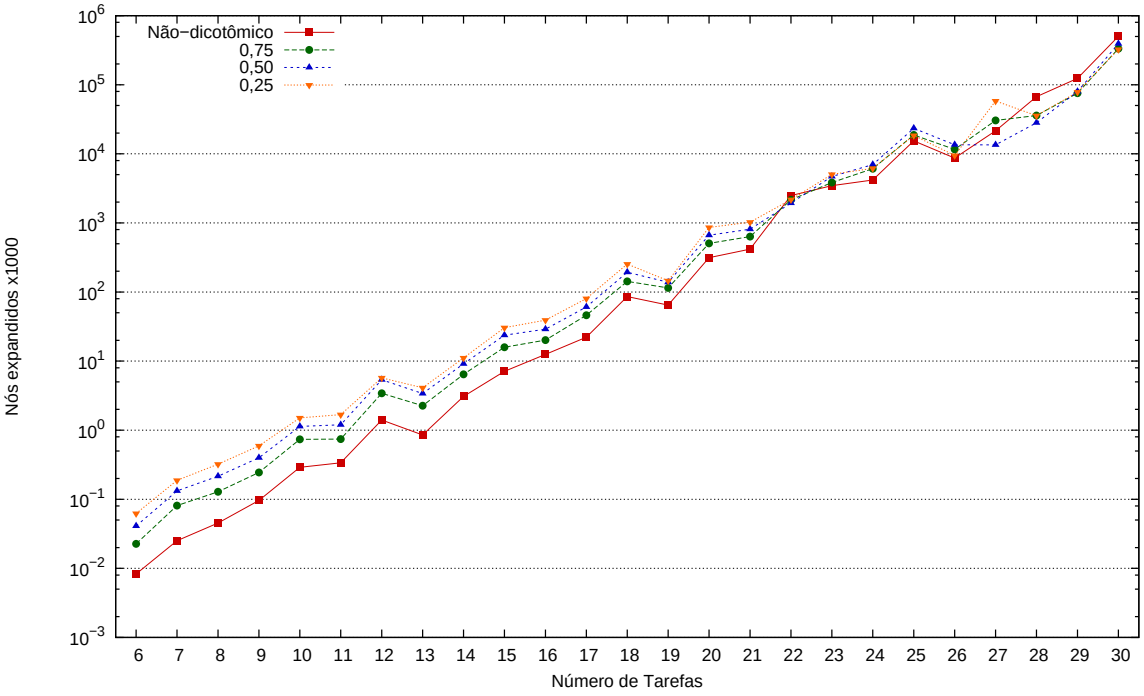


Figura 4.8: Número de nós expandidos variando o fator dicotômico do B&B

4.9 Instâncias de Grande Porte

Foi possível encontrar as soluções ótimas para problemas de até 30 tarefas com o B&B. Para problemas maiores, nós limitamos o tempo máximo de resolução do B&B para 7.200 segundos (duas horas), e medimos a melhora na solução provida pelo GRASP. Como o B&B não garante a otimalidade se o tempo limite for excedido, essa solução é comparada à solução encontrada pelo GRASP com 20.000 iterações (GRASP2) para avaliar se o GRASP com um número

Tarefas	Não-dicotômico	0,75	0,50	0,25
6	8,30 ± 12,29	22,60 ± 11,73	41,00 ± 11,20	62,20 ± 11,64
7	25,10 ± 08,57	80,90 ± 06,87	132,30 ± 07,11	187,80 ± 07,14
8	45,30 ± 09,62	128,50 ± 10,62	215,90 ± 09,57	321,80 ± 11,14
9	96,20 ± 08,80	243,90 ± 08,82	398,10 ± 09,29	591,80 ± 08,38
10	290,00 ± 16,02	735,70 ± 14,68	1.131,80 ± 14,01	1.504,00 ± 12,14
11	337,10 ± 12,90	743,00 ± 08,37	1.196,20 ± 07,31	1.678,30 ± 07,78
12	1.401,80 ± 27,02	3.420,70 ± 24,75	5.370,80 ± 25,23	5.730,00 ± 20,40
13	851,90 ± 24,26	2.263,00 ± 26,36	3.389,30 ± 25,97	4.115,00 ± 25,23
14	3.079,40 ± 11,43	6.438,20 ± 08,13	9.127,30 ± 08,88	11.134,70 ± 10,85
15	7.098,50 ± 12,79	15.880,50 ± 11,40	23.665,40 ± 12,80	30.438,90 ± 13,22
16	12.515,00 ± 13,04	20.169,80 ± 09,70	28.966,30 ± 12,15	39.108,60 ± 13,06
17	22.207,00 ± 25,60	45.978,80 ± 23,87	60.598,80 ± 21,46	80.193,30 ± 20,42
18	85.882,90 ± 13,36	142.398,00 ± 20,37	193.355,10 ± 21,83	253.212,80 ± 22,34
19	64.742,90 ± 21,77	114.820,60 ± 18,62	138.394,90 ± 14,27	145.363,50 ± 16,08
20	313.626,80 ± 18,77	504.798,40 ± 17,96	664.153,00 ± 17,65	859.574,60 ± 16,08
21	417.956,20 ± 09,49	633.374,10 ± 08,51	812.984,20 ± 09,46	1.027.000,50 ± 08,77
22	2.484.061,60 ± 37,60	2.155.730,70 ± 28,14	1.939.886,20 ± 22,91	2.180.202,30 ± 24,52
23	3.466.514,30 ± 16,45	3.842.650,20 ± 12,01	4.692.514,90 ± 14,42	5.049.259,20 ± 13,66
24	4.173.514,90 ± 24,58	6.097.963,30 ± 25,26	6.981.967,90 ± 26,68	6.175.971,30 ± 23,34
25	15.413.490,80 ± 19,07	18.720.138,10 ± 18,09	23.439.300,30 ± 16,63	18.442.353,60 ± 14,96
26	8.705.022,80 ± 20,38	11.578.822,90 ± 21,24	13.535.487,90 ± 21,55	9.481.133,00 ± 30,45
27	21.481.047,50 ± 15,11	30.497.553,10 ± 48,19	13.459.272,90 ± 20,34	58.214.709,00 ± 50,85
28	67.076.597,60 ± 18,59	35.794.212,50 ± 18,35	28.160.722,20 ± 21,77	35.694.334,10 ± 33,13
29	123.603.675,80 ± 22,78	75.424.120,70 ± 18,75	79.664.006,60 ± 22,25	78.351.108,70 ± 26,09
30	500.890.594,50 ± 14,66	332.407.397,50 ± 18,48	392.019.482,90 ± 23,63	325.298.502,20 ± 27,05

Tabela 4.8: Número de nós expandidos e desvio padrão variando o fator dicotômico do B&B

Tarefas	Improvement	Distance
30	32.80 %	-2.31 %
40	2.02 %	1.69 %
50	0.25 %	2.93 %
60	0.26 %	2.64 %
70	0.00 %	3.31 %
80	0.07 %	3.10 %
90	0.21 %	2.20 %
100	0.09 %	2.19 %

Tabela 4.9: Comparando o desempenho do B&B e do GRASP em instâncias maiores

maior de iterações é uma abordagem melhor. Para calcular a melhora obtida pelo B&B na solução do GRASP, utilizamos a Fórmula (4.1). A distância entre a solução fornecida pelo B&B e pelo GRASP2 é dada pela Fórmula (4.2). Os resultados são mostrados na Tabela 4.9.

$$\frac{\text{GRASP sol.} - \text{B\&B sol.}}{\text{GRASP sol.}} \tag{4.1}$$

$$\frac{\text{B\&B sol.} - \text{GRASP2 sol.}}{\text{B\&B sol.}} \tag{4.2}$$

Com 30 tarefas, o B&B melhora a solução encontrada pelo GRASP com 1.000 iterações a uma média de 9.8%, e a distância da solução do B&B para a do GRASP2 é $-2,31$ em média. Um número negativo significa que a solução do B&B é em geral melhor que a do GRASP2. Como o B&B encontra a solução ótima para as instâncias de 30 tarefas, uma distância negativa é esperada neste caso. Mas com o crescimento do número de tarefas, a solução do GRASP2 se torna mais interessante e a melhora proporcionada pelo B&B sobre a solução do GRASP diminui. Com 70 tarefas, o B&B é incapaz de melhorar a solução do GRASP em todas as 20 instâncias.

Quando o número de tarefas aumenta, o B&B perde em eficiência, já que ele deve resolver um caixeiro viajante para cada máquina durante a segunda fase do branching. Nestas situações, a solução provida pelo GRASP2 é mais interessante.

Capítulo 5

Conclusões

Neste trabalho, consideramos um problema de seqüenciamento com máquinas paralelas não-relacionadas, datas de entrega e tempos de seqüenciamento dependentes da seqüência e de máquina. Um B&B utilizando GRASP como procedimento de inicialização e dois modelos MIP são propostos e testados.

A principal contribuição deste trabalho é o B&B que foi desenvolvido para resolver instâncias de um problema de seqüenciamento em máquinas paralelas não relacionadas, tempos de preparação dependentes da seqüência e datas de entrega. O limite superior e a estratégia de *branching* para este B&B foram testados e se mostraram eficientes. A qualidade do *LS* afeta bastante a eficiência do B&B e um *LS* mais próximo do ótimo implica em descobrir a solução ótima mais rapidamente, embora seja mais custoso calcular o *LS* no início da execução.

Além disso, foram avaliados dois modelos baseados em estratégias clássicas da literatura. Também foi gerado um conjunto de instâncias com diferentes valores para as datas de entrega, tempos de processamento e de preparação.

Considerando o desempenho dos algoritmos e modelos, conseguimos resolver instâncias de até 30 tarefas e 6 máquinas com o B&B. Mostramos que o B&B tem um desempenho um tanto melhor que a solução pelos modelos, já que foi feito especificamente para este tipo de problema. Após uma análise cuidadosa, pode-se perceber que quando as soluções para uma instância podem assumir valores muito diferentes, se torna mais fácil para o B&B podar a árvore de soluções. É o caso quando se aumenta a variação dos tempos de processamento e se aumenta o número de máquinas diferentes.

No entanto, realizar alterações na instância que prejudiquem o cálculo do limite inferior pode piorar bastante a poda da árvore. É o caso quando se diminui as datas de entrega ou se aumenta o intervalo de variação do tempo de preparação.

Além destas conclusões apontadas, a abordagem dicotômica não obteve muito sucesso para instâncias pequenas, mas este quadro pode mudar para instâncias maiores. Mas com instâncias muito grande, a utilização do GRASP apenas com um grande número de iterações se mostrou mais eficiente.

Trabalhos futuros incluem alterações no método de resolução, como utilização de outras heurísticas para calcular o limite inferior para o atraso ponderado e para o makespan, e um

maior aproveitamento pelo B&B da solução provida pelo GRASP. Também incluem uma abordagem multi-objetivo do problema, explorando sua fronteira de Pareto, e fazendo associações entre a variação de cada objetivo. Uma maior cooperação entre os métodos de resolução e uma versão paralelizada do B&B também são temas que podem ser abordados.

O desenvolvimento de um modelo discretizando o tempo, comparando resultados com os dois modelos apresentados neste trabalho também pode ser um trabalho interessante.

Outra extensão é uma generalização maior no tipo de problema, que pode passar a considerar restrições de elegibilidade e tempos de disponibilidade.

Apêndice A

Instância Exemplo

O arquivo de entrada de uma instância contém:
- o número de tarefas;
- o número de tipos de máquina;
- a semente utilizada para gerar o arquivo;
- as datas de entrega de cada tarefa;
- as prioridades de cada tarefa;
- para cada tipo de máquina:
. os tempos de processamento de cada tarefa;
. os tempos de preparação para cada par de tarefas;

Um arquivo de exemplo para 15 tarefas e 6 máquinas em 3 grupos de máquinas idênticas é dado abaixo.

```
15  3  2321113466
  535  264  473   53  140  177  102  120  203   67  463  441  616  572   24
  3  3  1  2  3  1  2  1  1  1  3  3  2  2  2

  22   7 200 105 161  51  31 177  81  11 200 171 168  47 101

  37  36  50  47  29  38  41  47  27  39  43  29  25  45  42
  46  50  39  28  48  29  38  50  40  32  48  35  33  28  43
  25  40  35  40  49  34  34  44  25  40  29  29  29  46  29
  42  25  25  30  32  45  50  28  40  37  48  50  28  32  33
  37  37  31  32  35  34  38  30  36  40  36  43  42  50  44
  48  26  38  26  45  46  26  25  49  35  42  38  46  44  36
  50  41  40  25  44  31  39  27  31  48  47  39  34  34  37
  32  29  38  32  49  27  34  30  45  39  28  50  44  38  41
  42  47  34  32  50  49  45  48  43  41  34  44  25  28  36
  47  46  46  38  29  35  31  44  46  45  32  45  27  28  37
  29  34  43  32  39  39  44  45  45  47  40  28  42  31  43
```

```
 27  44  40  49  48  28  46  43  47  49  48  29  30  32  26
 30  46  40  27  29  35  30  28  43  38  38  29  30  31  40
 47  31  42  26  26  34  42  31  38  34  44  34  37  30  39
 48  31  42  47  40  32  34  27  36  32  50  41  42  41  35

182 117 129 153 147  31 126 105  80 126 164 138  59  49  10

 39  34  37  25  40  34  27  49  35  49  45  48  45  46  42
 39  44  39  27  33  42  26  40  50  41  25  30  45  45  39
 43  30  26  25  32  26  25  40  41  37  36  47  45  31  50
 27  50  42  29  37  27  50  36  41  34  47  46  30  32  47
 49  33  29  29  35  28  49  25  40  44  43  45  49  34  35
 41  25  50  43  27  36  40  37  30  28  38  27  32  26  46
 37  38  39  44  34  37  40  26  36  33  30  35  42  49  29
 36  29  32  32  31  49  30  47  44  42  40  44  27  41  42
 44  33  40  45  50  32  36  34  48  45  33  43  30  42  49
 50  34  36  46  41  44  42  50  39  47  41  47  49  41  36
 36  32  40  32  25  33  37  41  30  36  32  42  26  47  29
 37  25  43  42  36  39  26  25  40  44  27  42  37  29  35
 33  43  46  48  38  36  41  48  50  47  47  32  48  49  42
 45  25  30  31  42  33  48  36  41  37  47  37  42  44  33
 48  36  30  47  41  40  29  48  33  34  50  50  25  31  26

 94  26 128  11  48  27  25  60  91 109 167 184  46  54 180

 46  41  25  35  28  32  28  34  45  26  31  40  36  45  32
 50  26  49  29  41  36  31  38  27  28  40  47  46  47  37
 36  34  35  40  46  25  44  48  25  32  30  34  27  46  39
 50  36  29  26  31  46  40  25  29  32  31  42  28  34  49
 29  45  25  46  30  30  32  28  48  28  38  30  36  39  30
 31  31  36  32  49  38  25  50  33  44  32  25  35  32  36
 41  37  27  25  35  41  50  33  44  42  50  31  45  39  44
 34  36  34  41  45  50  36  29  30  48  38  34  25  50  42
 34  46  50  50  50  47  37  47  25  44  45  36  40  41  35
 39  29  48  38  43  37  29  40  26  25  50  27  29  28  37
 48  27  26  32  47  41  35  46  41  46  26  33  39  45  27
 48  35  35  31  44  38  34  36  37  30  32  28  34  40  37
 43  26  50  35  30  41  50  34  46  44  40  28  44  38  32
 38  27  48  33  31  45  46  28  33  28  42  26  44  30  49
 37  35  28  38  43  48  45  45  39  30  32  28  36  42  33
```

Referências Bibliográficas

Acero, R. D. e Delgado, J. F. T. (2000). Aplicación de una heurística de búsqueda tabú en un problema de programación de tareas en línea flexible de manufactura. *COLOMBIA*. Available on-line at http://hdl.handle.net/1992/570.

Błażewicz, J.; Ecker, K. H.; Pesch, E.; Schmidt, G. e Weglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*. Springer Verlag, Berlin.

Cormen, T.; Leiserson, C. e Rivest, R. (1990). *Introduction to Algorithms*, chapter 26: All-Pairs Shortest Paths. The MIT Press, Cambridge, MA.

Dachs, J.; Bayona, J. M.; Fowler, S. W.; Miquel, J.-C.; Albaiges, J.; Feo, T. A.; Sarathy, K. e McGahan, J. (1996). A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, 23(9):881–895.

Feo, T. A.; Venkatraman, K. e Bard, J. F. (1991). A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research*, 18(9):635–643.

Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co.

Glover, F. W. (1996). Tabu search and adaptive memory programming - Advances. *Interfaces in Computer Science and Operations Research*, pp. 1–75. R S Barr, R V Helgason, and J L Kennington, eds., Kluwer Academic Publishers.

Glover, F. W. (2000). *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Opertations Research*, chapter Multi-Start and Strategic Oscillation Methods - Principles to Exploit Adaptive Memory, pp. 1–24. Kluwer Academic Publishers.

Glover, F. W. e Kochenberger, G. A. (2003). *Handbook of Metaheuristics*. Springer.

Glover, F. W. e Laguna, M. (1997). *Tabu Search*. Springer.

Gómez Ravetti, M. (2003). Problemas de seqüenciamento com máquinas paralelas e tempos de preparação dependentes da seqüência. Master's thesis, DCC, Universidade Federal de Minas Gerais.

Gómez Ravetti, M. e Mateus, G. R. (2003). Seqüenciamento de tarefas com máquinas paralelas permitindo atrasos e com tempos de preparação de máquinas dependentes da sequência. In *XXXV Simpósio Brasileiro de Pesquisa Operacional*, Natal - Brazil.

Gómez Ravetti, M.; Mateus, G. R. e Rocha, P. L. (2004a). Programação de tarefas com máquinas paralelas não relacionadas, tempos de preparação de máquinas dependentes da seqüência, datas de entrega e restrições de elegibilidade. In *XXXVI SBPO - Simpósio Brasileiro de Pesquisa Operacional*, São João Del Rei (MG - Brazil).

Gómez Ravetti, M.; Mateus, G. R.; Rocha, P. L. e Pardalos, P. M. (2006). Scheduling problem with non-related parallel machines, sequence dependent setups, due dates and eligibility constraints. *International Journal of Operational Research (IJOR)*. Accepted for publication.

Gómez Ravetti, M.; Rocha, P. L. e Mateus, G. R. (2004b). Programación de tarefas con máquinas paralelas no relacionadas, puestas a punto dependientes de la secuencia y restricciones de elegibilidad. In *XII Congreso Latino-Iberoamericano de Investigación de Operaciones y Sistemas (CLAIO)*, Havana - Cuba.

Hans-Joachim, G. e John, U. (1996). Methods for solving pratical problems of job-shop scheduling modelled in CLP(FD). In *Conf. on Practical Application of Constraint Technology*, pp. 73–92.

Ho, J. C. e Chang, Y.-L. (2006). Minimizing the number of tardy jobs for m parallel machines. *European Journal of Operational Research*, 84:343–355.

Jackson, J. R. (1955). Scheduling a production line to minimize maximum tardiness. Technical report, Management Sci. Res. Project, UCLA, Research Report 43.

Jackson, J. R. (1956). An extension of johnson's on job lot scheduling. *Naval Research Logistics Quarterly*, 3:201–203.

Johnson, S. M. (1954). Optimal two and three stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68.

Kim, S. C. e Bobrowski, P. M. (1994). Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research*, 32(7):1503–1520.

Koulamas, C. e Kyparisis, G. J. (2004). Makespan minimization on uniform parallel machines with release times. *European Journal of Operational Research*, 157:262–266.

Laguna, M. e Martí, R. (1999). grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52.

Liu, C.-Y. e Liao, D.-Y. (Aug 2000). Scheduling flexible flowshops with sequence-dependent setup effects. In *IEEE Transactions on Robotics and Automation*, volume 16 of *4*, pp. 408–419.

Lopes, M. J. P. e de Carvalho, J. M. V. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, 176:1508–1527.

Luh, P. B.; Gou, L.; Zhang, Y.; Nagahora, T.; Tsuji, M.; Yoneda, K.; Hasegawa, T.; Kyoya, Y. e Kano, T. (1998). Job shop scheduling with group-dependent setups, finite buffers, and long time horizon. *Annals of Operations Research*, 76:233–259.

Lustig, I. J. e Puget, J.-F. (2001). Program != program: Constraint programming and its relationship to mathematical programming. *Interfaces*, 31:29–53.

Manne, A. S. (1960). On the job-shop scheduling problem. *Operations Research*, 8:219–223.

Meyr, H. (2000). Simultaneous lotsizing and scheduling by combining local search with dual reoptimization. *European Journal of Operational Research*, 120:311–326.

Meyr, H. (2002). Simultaneous lotsizing and scheduling on parallel machines. *European Journal of Operational Research*, 139:277–292.

Pinedo, M. (1995). *Scheduling - Theory, Algorithms and Systems*. Prentice Hall International Series in Industrial and Systems Engineering.

Pochet, Y. e Wosley, L. A. (2006). *Production Planning by Mixed Integer Programming*. Springer Series in Operations Research and Finacial Engeneering. Springer, Verlag, Berlin.

Rabadi, G.; Mollaghasemi, M. e Anagnostopoulos, G. C. (2004). A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. *Computers & Operations Research*, 31:1727–1751.

Resende, M. G. C.; Binato, S.; Hery, W. J. e Loewenstern, D. M. (2002). A greedy randomized search procedure for job shop scheduling. In *Essays and Surveys in Metaheuristics*, pp. 59–80. Celso C. Ribeiro and P. Hansen, editors.Kluwer Academic.

Resende, M. G. C. e Feo, T. A. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Rocha, P. L.; Gómez Ravetti, M. e Mateus, G. R. (2004). The metaheuristic GRASP as an upper bound for a branch and bound algorithm in a scheduling problem with sequence-dependent setup times. In *4th EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics*, Nottingham - England. Available on-line at http://webhost.ua.ac.be/eume/workshops/hybrid/A037Revised.pdf.

Rocha, P. L.; Gómez Ravetti, M.; Mateus, G. R. e Pardalos, P. M. (2006). Exact algorithms for a scheduling problem with nonrelated parallel machines and sequence and machine dependent setup times. *Computers and Operations Research (C&OR)*. Accepted for publication.

Roundy, R.; Cakanyldirim, M.; Chen, D.; Freimer, M.; Jackson, P. L. e Melkonian, V. (1999). Capacity-driven acceptance of customer orders for a multi-stage batch manufacturing system: Models and algorithms. Technical Report 1233, School of Operations Research and Industrial Engineering, Cornell University.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66.

Wagner, H. W. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistic Quarterly*, 6:131–140.

Wikimedia Foundation, I. (2005). Interquartile mean. Technical report, Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Interquartile_mean.