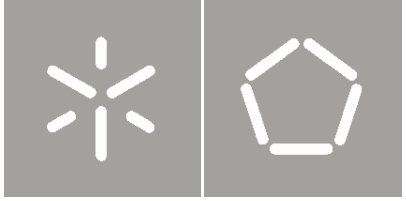


Universidade do Minho
Escola de Engenharia

António Jorge da Silva Trindade Duarte

Aplicação de Algoritmos de Partição e Geração de Colunas ao Agendamento de Máquinas Paralelas

Tese de Doutoramento
Ramo de Engenharia de Produção e Sistemas
Área de Investigação Operacional

Trabalho efectuado sob a orientação de
Professor Doutor José Manuel Vasconcelos Valério de Carvalho

Setembro de 2005

D E C L A R A Ç Ã O

É autorizada a reprodução integral desta tese apenas para efeitos de investigação, mediante declaração escrita do interessado, que a tal se compromete.

Universidade do Minho, 20 de Setembro de 2005.

O Autor,

Em memória do meu avô Luís.

Agradecimentos

Em primeiro lugar, ao Prof. Dr. Valério de Carvalho, com quem tive a honra e o privilégio de trabalhar nos últimos anos. Transcendendo em muito as funções de orientador científico, agradeço o seu trabalho, dedicação e disponibilidade, os seus conselhos sábios, as suas palavras de encorajamento e estímulo e, acima de tudo, a sua amizade.

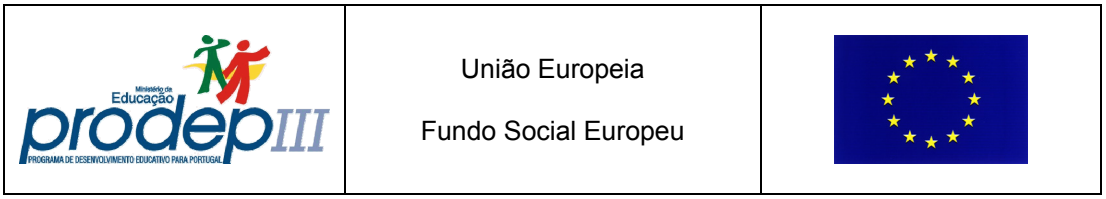
À Universidade do Minho e, em particular, à Escola de Engenharia e ao Departamento de Produção e Sistemas pela oportunidade de desenvolver este trabalho no seio da sua comunidade científica.

Ao Instituto Politécnico de Bragança e à Escola Superior de Tecnologia e Gestão pelo ambicioso programa de mestrados e doutoramentos que estão a levar a cabo e que me proporcionou excelentes condições para fazer este trabalho. Não posso deixar de incluir aqui o Departamento de Gestão Industrial e, em particular, o Prof. Dr. Armando Leitão pelo estímulo constante à conclusão deste trabalho.

À minha família, pelo apoio, compreensão e paciência.

A todos os que, directa ou indirectamente, contribuíram para este trabalho e que fariam uma longa lista.

Este trabalho foi financiado pelo Fundo Social Europeu através do concurso n.º 2/5.3/PRODEP/2001



Resumo

Aplicação de Algoritmos de Partição e Geração de Colunas ao Agendamento de Máquinas Paralelas

No presente trabalho é proposto um algoritmo para resolver de forma exacta um problema de programação de máquinas paralelas idênticas, com tarefas maleáveis sujeitas a datas de disponibilidade e datas de conclusão arbitrárias. O objectivo é minimizar uma função do trabalho atrasado e dos custos de preparação.

Considera-se que uma tarefa é maleável se o conjunto de máquinas onde é agendada puder ser escolhido e, eventualmente, modificado livremente ao longo do tempo. Evidentemente, a cada preempção realizada está associado um custo que é tido em conta na função objectivo.

Para este problema é proposta uma formulação de programação inteira sobre a qual será aplicada a decomposição de Dantzig-Wolfe, com vista a resolver o problema através da geração de colunas. No modelo decomposto, cada coluna representa a agenda de uma das máquinas e a função do subproblema é gerar agendas atractivas para incluir na solução de agendamento.

Para efectuar a partição foi desenvolvido um modelo de fluxo de custo mínimo equivalente e a partição é feita sobre as variáveis correspondentes a fluxos nos arcos desta formulação. Existe uma relação matemática entre as variáveis do modelo de fluxo de custo mínimo e as variáveis do modelo original.

Também foram desenvolvidas heurísticas de melhoria local de soluções válidas e uma heurística de arredondamento de quaisquer soluções fraccionárias. Para além disso, foram estudados dois casos particulares do problema: o problema com tarefas ordenáveis e o problema com janelas de agendamento comuns.

Finalmente, foi levado a cabo um largo conjunto de testes computacionais para verificar a eficiência dos vários algoritmos propostos e para determinar a

sensibilidade do modelo a parâmetros de dimensão da instância: o número de máquinas, o número de tarefas e o tamanho do horizonte de agendamento.

Summary

Application of Branch-and-Price to Parallel Machine Scheduling

This work presents an algorithm for solving exactly a scheduling problem with identical parallel machines and malleable tasks, subject to arbitrary release dates and due dates. The objective is to minimize a function of the late work and of the setup costs.

A malleable task is such that the set of processors allotted to its processing can be freely chosen and, possibly, modified over time. Obviously, for each preemption there is a corresponding setup cost to be considered in the objective function.

To approach this problem, we propose an integer programming formulation and we apply the Dantzig-Wolfe decomposition to it in order to solve the problem by column generation. On the decomposed model, each column represents the schedule of a single machine and the goal of the subproblem algorithm is to provide new valid and attractive schedules that can be included in the global schedule.

For the branching phase, we developed an equivalent network flow model and the branching is made on the variables corresponding to flows on arcs of this formulation. There is a mathematical relation between the variables of this network flow model and the variables of the original model.

We also developed heuristics for local improvement of valid solutions and a heuristic for rounding any non integral solution. Besides, we studied two special cases of the problem: the problem with an orderable task set and the problem with common time windows.

Finally, we carried out an extensive set of computational tests to verify the algorithm's efficiency and to determine the model's sensitivity to instance size parameters: the number of machines, the number of tasks and the size of the planning horizon.

Índice Geral

Índice de Figuras

Índice de Tabelas

Índice de Exemplos

Capítulo 1

Introdução

1.1 Âmbito da Tese

A programação inteira é uma técnica da Investigação Operacional que tem vindo a ser largamente aplicada ao longo das últimas décadas na resolução de problemas reais. A geração de colunas, sendo uma técnica com mais de 4 décadas, teve em anos recentes grandes desenvolvimentos e foi aplicada com sucesso a problemas maiores e mais complexos, podendo agora contemplar restrições dos problemas que antes eram frequentemente ignoradas.

É hoje possível resolver modelos que incluem os detalhes que surgem normalmente em problemas reais, tal como restrições de horários e restrições associadas à força de trabalho e que mesmo assim produzem soluções utilizá-

veis. Este trabalho teve grandes impactos na área da logística e distribuição ou no sector dos transportes aéreos. O planeamento de rotas de veículos com janelas temporais e o escalonamento de tripulações nos transportes aéreos são exemplos de problemas resolvidos hoje de forma rotineira e fornecendo soluções que representam poupanças anuais de grande significado.

A combinação entre a técnica da geração de colunas e as técnicas de *branch-and-bound* foi utilizada com sucesso para obter soluções óptimas para problemas de grande dimensão com variáveis inteiras e problemas de optimização combinatória. Embora ambos os métodos sejam conhecidos há muito, a sua utilização combinada é bastante recente. Estas técnicas são conhecidas na literatura anglo-saxónica como *branch-and-price* e são conhecidas entre nós como técnicas de *partição e avaliação*, numa tradução mais exacta, ou simplesmente como *geração de colunas*, fazendo uma tradução mais livre e modificando um pouco o significado original do termo. Utilizando esta metodologia tem sido possível resolver instâncias cada vez maiores de vários problemas de programação inteira.

Muitos dos modelos desenvolvidos possuem variáveis binárias. No entanto, há muitos problemas importantes que possuem variáveis inteiras genéricas, tal como o problema de corte (*cutting-stock problem*) e diversos problemas na área do agendamento de máquinas. Por exemplo, Valério de Carvalho [49] apresentou um algoritmo de partição e avaliação para resolver problemas de corte e problemas de empacotamento. O algoritmo referido resolveu de forma consistente todas as instâncias do problema de empacotamento da OR-Library, incluindo instâncias em aberto à altura. O esquema utilizado naquele trabalho provou ser uma metodologia que, com base no teorema de decomposição fluxos (ver [1]), pode ser utilizada para resolver vários problemas com variáveis inteiras genéricas pelas técnicas de partição e avaliação. A metodologia pode ser aplicada a problemas que possam ser formulados como problemas de fluxo em redes, com eventuais restrições adicionais.

A ideia básica é a seguinte: as soluções podem ser representadas como fluxos numa rede e procura-se uma decomposição de fluxos inteiros utilizando uma

estratégia de partição que impõe restrições sobre esses fluxos. É possível demonstrar que esta metodologia é equivalente à decomposição de Dantzig-Wolfe e, por esse facto, os modelos são igualmente fortes. No entanto, este esquema dá uma melhor perspectiva do problema, que pode ser usada para derivar regras de partição mais fortes e melhor balanceadas que as resultantes de hiperplanos e de cortes primais expressos em termos das variáveis originais do problema. É sabido que estas questões são de importância crucial neste tipo de métodos.

A principal razão para reformular os modelos utilizado a decomposição de Dantzig-Wolfe é a obtenção de modelos mais fortes. Quando o subproblema não tem integralidade, o limite inferior resultante da decomposição pode ser muito mais forte que o resultante da relaxação linear do modelo com variáveis originais (ver [45]). Na outra face da moeda, ficamos com modelos que têm um número exponencialmente grande de colunas e têm que ser resolvidos com recurso à geração de colunas. Para uma revisão sobre geração de colunas, ver [47].

Outras vantagens da geração de colunas são:

- É relativamente fácil modelar restrições complexas, uma vez que podem ser normalmente incluídas no subproblema, usualmente resolvido através de programação dinâmica;
- Embora os limites inferiores sejam iguais aos da relaxação lagrangeana, a geração de colunas parece guiar melhor os processos para o óptimo (ver [5]);
- A geração de colunas permite a utilização de cortes primais, quando expressos em termos de variáveis originais.

Na fase de partição e avaliação é necessário assegurar a compatibilidade entre o problema primal restrito e o subproblema. Este problema não é de todo simples:

- Primeiro, a regra de partição não pode induzir alterações intratáveis na estrutura do subproblema. Este deve permanecer o mesmo pro-

blema de optimização ao longo de todo o processo. É hoje pacífico que, para combinar geração de colunas e partição, esta deve ser feita sobre variáveis da formulação original do problema (ver [5]);

- Em segundo, há a questão da simetria: a mesma solução ou conjunto de soluções, não deve aparecer em mais do que um ramo da árvore de pesquisa, uma vez que isso atrasa o processo. As estratégias de partição devem ser desenhadas de forma a dividir o espaço de soluções em conjuntos mutuamente exclusivos, a explorar em diferentes nodos da árvore de pesquisa;

As técnicas de geração de colunas são conhecidas por terem uma convergência lenta e demorada. Em [24], du Merle et al. apresentaram um esquema de estabilização que combina a técnica da perturbação com a técnica das penalidades exactas, e que resulta numa redução muito significativa do número de colunas geradas quando aplicado a diversos problemas de optimização.

Em [50], Valério de Carvalho introduziu a ideia de utilizar cortes duais para acelerar a convergência. Nesse trabalho, foi derivada uma família de cortes duais para o problema de corte e foi demonstrado que a introdução de um número polinomial destes cortes reduz de forma apreciável o número de colunas geradas, o número de soluções degeneradas e o tempo gasto para resolver a relaxação linear do problema. Em problemas grandes e difíceis, o factor de aceleração foi de, aproximadamente, 4.5.

A ideia de restringir o espaço dual do problema num processo de geração de colunas (isto é, um algoritmo de planos de corte duais), mas preservando o valor da solução óptima, é concretizada adicionando colunas (cortes duais) que, em termos das variáveis originais, correspondem a ciclos em que alguns arcos são atravessados no sentido oposto ao da sua orientação.

Neste trabalho vai tentar desenvolver-se um algoritmo *state-of-the-art* para a resolução de problemas de agendamento, e que incorpore este conjunto de ideias.

1.2 Motivação

A motivação para este trabalho surgiu de um problema resolvido no âmbito da dissertação de mestrado [23]. Por sua vez, o problema surgiu de contactos havidos com uma empresa do sector têxtil, onde, no âmbito de uma remodelação do sistema de informação, se pretendia fazer o agendamento automático das encomendas num parque de teares. As encomendas podiam ser agendadas num conjunto arbitrário de processadores, naquilo que se designa por um problema com tarefas maleáveis[1].

Na altura o problema foi abordado através de heurísticas de melhoria local aplicadas a uma solução de agendamento inicial obtida através da resolução de um problema de fluxo de custo mínimo.

Como era nossa convicção que o agendamento automático de máquinas paralelas era um campo para a aplicação das técnicas de geração de colunas onde poderiam ser obtidos bons resultados, o problema foi novamente abordado.

Como se poderá ver ao longo deste trabalho, trata-se de um problema de programação inteira de grande dimensão e complexidade.

1.3 Contribuições da Tese

De todo o trabalho desenvolvido destaca-se:

- Apresentação de um modelo e correspondente algoritmo para resolução exacta de problemas de agendamento em máquinas paralelas idênticas de tarefas maleáveis com datas de disponibilidade arbitrárias, com vista a minimizar uma função do trabalho atrasado e do número de preparações;
- Apresentação de um modelo matemático de programação inteira e sua decomposição através da técnica de Dantzig-Wolfe, conduzindo a

[1] A definição exacta de tarefas maleáveis e toda a nomenclatura relacionada com o multiprocesamento são apresentadas em capítulos posteriores.

uma nova formulação que é resolvida através da geração diferida de colunas e de partição e avaliação;

- Reformulação do problema como um modelo de fluxo de custo mínimo que é utilizado para obter soluções válidas e variáveis sobre as quais será efectuada a partição;
- Definição de heurísticas rápidas para a melhoria local de soluções válidas e cálculo de limites inferiores;
- Definição de regras de redução de simetria nas colunas geradas no subproblema;
- Desenho de uma heurística de arredondamento rápido de soluções fraccionárias;
- Aplicação de cortes primais já conhecidos de outros problemas de programação inteira ao problema em estudo;
- Aplicação de *dual feasible functions* para a geração de cortes primais para o problema;
- Estudo detalhado de dois casos particulares (problema com tarefas ordenáveis e problema com janelas comuns) em que foi possível melhorar a eficiência do algoritmo genérico e resolver instâncias de maior dimensão.

1.4 Estrutura da Tese

A tese é composta de nove capítulos, sendo o primeiro a presente introdução. Nesta secção faz-se uma pequena descrição dos conteúdos de cada um dos capítulos.

No capítulo 2 é feita uma revisão sobre problemas de processamento paralelo, introduzindo a principal notação e descrevendo os vários tipos de problemas existentes.

No capítulo 3 é feita uma revisão sobre geração de colunas e métodos de partição e avaliação, onde estas técnicas são descritas de forma genérica. Também aqui é feita uma revisão sobre a decomposição de Dantzig-Wolfe.

No capítulo seguinte (capítulo 4) é feita uma revisão da principal literatura sobre problemas de agendamento e as metodologias correntemente utilizadas na sua abordagem. A atenção é focada sobre problemas que são semelhantes ou têm interesse para o problema em análise.

O capítulo 5 constitui o núcleo fundamental da tese e nele é descrito o problema em detalhe, é apresentado o modelo de programação matemática proposto para a sua resolução e é feita a decomposição desse modelo através da técnica de Dantzig-Wolfe. Também aqui é apresentado o algoritmo de programação dinâmica utilizado para resolver o subproblema e o esquema de partição utilizado durante a fase de partição e avaliação.

O capítulo 6 é outro dos capítulos fundamentais do trabalho, onde são descritos pormenores da implementação computacional efectuada. É aqui que são apresentadas as heurísticas de cálculo de limites inferiores, de melhoria local de soluções e de arredondamento de soluções fraccionárias. São igualmente apresentados cortes primais baseados em cortes conhecidos de outros problemas de programação inteira e em *dual feasible functions*.

No capítulo 7 são descritos dois casos particulares do problema: o problema com tarefas ordenáveis e o problema com janelas de execução comuns. É também descrita a forma de obter ganhos computacionais, aproveitando características específicas desses casos particulares.

O capítulo 8 é dedicado à descrição, apresentação e análise dos resultados dos testes computacionais realizados, com vista a testar os limites do algoritmo quanto ao tamanho e tipo de instâncias consistentemente resolvidas.

Finalmente, no capítulo 9 são tecidas algumas considerações globais ao trabalho realizado e são apresentadas perspectivas para trabalho futuro de melhoria do modelo e dos algoritmos propostos.

Capítulo 2

Processamento Paralelo

Neste capítulo começa-se por descrever genericamente os problemas de processamento paralelo e indicar algumas áreas de aplicação. Numa secção posterior é definida toda a simbologia matemática relacionada com problemas de processamento paralelo, que será depois utilizada ao longo de todo o trabalho. Finalmente é apresentado o esquema de classificação de Graham, com especial incidência na parte relevante para os problemas de processamento paralelo.

2.1 Introdução

Podem encontrar-se problemas de processamento paralelo sempre que, em determinada fase do processo produtivo, uma determinada operação produtiva possa ser executada em diferentes máquinas, iguais ou mais ou menos semelhantes, em alternativa ou, caso o trabalho seja divisível, simultaneamente.

Normalmente, o processamento paralelo é utilizado para aumentar a capacidade produtiva em determinada fase do processo ou para permitir redundância em processos críticos, não sendo estes dois objectivos incompatíveis entre si.

Isto leva a que na prática industrial surjam com alguma frequência problemas de processamento paralelo. Por exemplo, na indústria têxtil, as secções de tecelagem possuem, por norma, uma grande quantidade de teares onde as tarefas (encomendas) são processadas. Ainda na mesma indústria, as secções de costura são outro exemplo de capacidade produtiva instalada com o recurso ao processamento paralelo. Note-se que nestes exemplos, definindo uma tarefa como uma quantidade a produzir, esta pode ser executada em vários processadores diferentes, em alternativa ou simultaneamente.

2.2 Definições Matemáticas

2.2.1 Caracterização Genérica dos Problemas

Um problema de planeamento operacional é em larga medida caracterizado pelo conjunto de tarefas a executar e pelo conjunto de processadores que as vão executar. Designe-se por $T = \{T_1, T_2, \cdots, T_n\}$ o conjunto das n tarefas a processar e por $P = \{P_1, P_2, \cdots, P_m\}$ o conjunto dos m processadores do problema.

Se $m > 1$ e todas as máquinas executarem as mesmas funções, estamos perante um problema de processamento paralelo. Se os processadores execu-

tarem funções diferentes, são processadores dedicados, estando a sua caracterização fora do âmbito deste trabalho.

O problema consiste em encontrar uma solução que atribua às tarefas do conjunto T um ou mais processadores do conjunto P de modo a que aquelas possam ser completadas, respeitando a capacidade dos processadores e todas as eventuais restrições adicionais impostas ao sistema.

Há duas restrições que são normalmente impostas aos sistemas: a primeira impõe que, em determinado momento, um processador só pode executar uma única tarefa; a segunda impõe que uma tarefa, em determinado instante, apenas pode ser executada por um único processador. Esta segunda restrição não é imposta se o sistema permitir o processamento simultâneo das tarefas, também designado por multiprocessamento. Este é o caso dos problemas abordados neste trabalho.

Aparecem referidas na literatura várias formas de multiprocessamento. Usamos como referência o trabalho de Drozdowski [22] onde são sistematizados vários casos. O problema do agendamento de tarefas maleáveis em processadores paralelos é um desses casos. O trabalho de Blazewicz et al. [8] também se refere a este problema, pelo que se pode considerar que os conceitos de *tarefa maleável* e de *multiprocessamento* se aplicam a este trabalho. Mais à frente neste capítulo são detalhadas algumas questões relacionadas com estes conceitos.

2.2.2 Caracterização dos Processadores

Tal como já se disse, os processadores podem ser *paralelos*, no caso de executarem todos a mesma função, ou *dedicados*, no caso de executarem diferentes funções.

No caso dos processadores paralelos é comum distinguir três tipos de sistemas, em função da velocidade com que processam as tarefas. No caso em que os processadores têm uma velocidade de processamento comum, independente da tarefa executada, designam-se por processadores *idênticos*. Se cada proces-

sador tiver uma velocidade de processamento diferente mas independente da tarefa executada, designam-se por processadores *uniformes*. Neste caso associa-se uma constante b_i a cada processador que reflecte a sua velocidade relativa. Finalmente, se a velocidade do processamento for dependente da combinação processador/tarefa, os processadores designam-se por *não relacionados*.

Por norma, a complexidade dos problemas cresce quando se passa de processadores idênticos para processadores uniformes e destes para processadores não relacionados. Isto é lógico, uma vez que os processadores uniformes são um caso particular de processadores não relacionados e os processadores idênticos são um caso particular de processadores uniformes.

2.2.3 Caracterização das Tarefas

Cada tarefa $T_j \in T$ tem que ser caracterizada pelo seguinte conjunto de informações:

- Vector de tempos de processamento, $p_j = \left[p_{1j}, p_{2j}, \cdots, p_{mj}\right]^T$, em que p_{ij} corresponde ao tempo que o processador P_i demora a processar a tarefa T_j. No caso de os processadores serem uniformes, estando definida a constante b_i para os processadores, pode definir-se uma constante p_j (tempo de processamento padrão, normalmente medido no processador mais lento) tal que $p_{ij} = p_j / b_i$. No caso de processadores idênticos tudo fica mais simples, uma vez que para cada tarefa basta definir a constante p_j.
- Data de disponibilidade, r_j, que representa o momento a partir do qual a tarefa T_j pode começar a ser processada. Em muitos casos, esta constante não é definida, assumindo-se que todas as tarefas ficam disponíveis a partir do momento 0.
- Data de conclusão, d_j, que denota o momento em que a tarefa T_j deverá estar concluída. O agendamento pode não ser possível (ou desejável) sem haver violação destas datas.

- Datas limite (*deadlines*), $\tilde{d}_j$, que impõem datas de conclusão máximas para as tarefas, sem possibilidade de violação.
- Peso da tarefa, w_j, que pode reflectir a prioridade, importância ou urgência de umas tarefas em relação às restantes.

Por uma questão de simplicidade, e sem prejuízo de uma aplicação suficientemente genérica dos modelos, muitas vezes é assumido que os parâmetros que foram enumerados assumem valores inteiros ou racionais.

Uma característica importante dos modelos é o facto de haver ou não a possibilidade de interromper o processamento das tarefas uma vez este iniciado, com o objectivo de o retomar mais tarde. No caso de haver esta possibilidade estamos perante um modelo de tarefas preemptivas ou modelo com preempção.

Podem ainda ser definidas relações de dependência entre as tarefas, como por exemplo, poder ser necessário terminar a tarefa T_x antes de iniciar a tarefa T_y. Neste caso definir-se-ia a dependência $T_x \prec T_y$. É usual o conjunto de dependências de um modelo ser definido num grafo apropriado.

Quanto houver multiprocessamento também é necessário definir as características a ele associadas. No caso de processadores dedicados, a cada tarefa pode estar associado um conjunto fixo de processadores de que a tarefa necessita para ser executada ou podem existir vários conjuntos alternativos, variando ou não os tempos de execução. No caso de processadores paralelos, as tarefas podem necessitar de um determinado número de processadores simultaneamente ou de um número arbitrário de processadores (eventualmente com um limite máximo). Neste caso também é necessário definir como variam os tempos de execução em função do número de processadores utilizados. É ainda necessário definir se o perfil de processadores utilizados pode ser alterado ao longo do tempo ou se é fixo e, neste último caso, se é ou não permitida a preempção das tarefas.

Quando o perfil de processadores utilizados no processamento de determinada tarefa se pode alterar ao longo do tempo, as tarefas podem designar-se de maleáveis (ver Blazewicz et al. [8]).

2.2.4 Funções Objectivo

Uma solução de agendamento consiste na afectação de fatias de tempo dos processadores do conjunto P às tarefas do conjunto T, respeitando as seguintes condições:

- Em cada momento, nenhum processador está afecto a mais de uma tarefa e nenhuma tarefa está a ser processada em mais do que um processador, não existindo esta última condição em alguns modelos;
- Qualquer tarefa T_j é completada no intervalo $\left[r_j, \infty\right]$ (ou no intervalo $\left[r_j, \tilde{d}_j\right]$ se este tiver sido definido);
- Todas as relações de dependência (se existirem) são respeitadas;
- Se o modelo não permitir a preempção, nenhuma tarefa é interrompida antes de terminar o processamento. Se o modelo permitir a preempção, o número de preempções é finito[2];
- Todas as eventuais restrições adicionais, específicas dos modelos concretos, são respeitadas.

De uma solução calculada nestes termos, é trivial o cálculo dos seguintes valores:

- Instante de conclusão da tarefa, C_j;
- Tempo de permanência no sistema (*flow time*), $F_j = C_j - r_j$;
- Desvio da data de conclusão (*lateness*), $L_j = C_j - d_j$;
- Atraso (*tardiness*), $D_j = \max\left\{L_j, 0\right\}$

[2] Apenas por motivos práticos.

A partir destas quantidades podem desenvolver-se as principais medidas de eficiência ou critérios de óptimo que se podem encontrar na literatura:

- Tamanho do agendamento (*makespan*), $C_{max} = \max\{C_j\}$;
- Tempo médio de permanência no sistema (*mean flow time*), $\overline{F} = \frac{1}{n} \cdot \sum_{j=1}^{n} F_j$, ou na versão ponderada, $\overline{F}_w = \sum_{j=1}^{n}(w_j \cdot F_j) \Big/ \sum_{j=1}^{n} w_j$;
- Desvio da data de conclusão máximo (*maximum lateness*), $L_{max} = \max\{L_j\}$;
- Trabalho atrasado (*total tardiness*), $D = \sum_{j=1}^{n} D_j$, e trabalho atrasado ponderado (*total weighted tardiness*), $D_w = \sum_{j=1}^{n} w_j D_j$;
- Atraso médio, $\overline{D} = \frac{D}{n}$, e atraso médio ponderado, $\overline{D}_w = D_w \Big/ \sum_{j=1}^{n} w_j$;
- Número de tarefas com atraso, $U = \sum_{j=1}^{n} U_j$, em que $U_j = 1$ se $D_j > 0$ e $U_j = 0$ se $D_j \leq 0$, ou a número ponderado de tarefas com atraso, $U_w = \sum_{j=1}^{n} w_j \cdot U_j$.

De uma maneira geral, o critério do tamanho do agendamento conduz a altas taxas de ocupação dos processadores de modo a que estes sejam libertados no mais curto espaço de tempo. Os tempos médios de permanência no sistema conduzem por norma à minimização do trabalho em curso. Os critérios baseados nas datas de conclusão são de particular importância quando se produz por encomenda, uma vez que, normalmente, é pretendido o respeito ou a violação mínima daquelas datas.

Embora estas sejam as funções objectivo que, na literatura, se encontram com mais frequência, a situação concreta pode aconselhar alterações mais ou menos vultuosas às mesmas, ou até critérios completamente diferentes.

2.2.5 Outras Definições

Um problema de programação, Π, pode ser caracterizado por um conjunto de parâmetros, tais como os definidos nas secções anteriores, e por uma medida de eficiência. Uma instância, I, desse problema pode ser obtida atribuindo valores ao conjunto de parâmetros do problema.

Uma solução cuja medida de eficiência, γ, é mínima, é chamada uma solução óptima e o seu valor será denotado por γ^*.

Finalmente, um algoritmo de programação encontra soluções para o problema de programação Π. O ideal seria obter algoritmos de optimização. No entanto, devido à complexidade de muitos problemas, algoritmos de aproximação ao óptimo ou heurísticas são perfeitamente aceitáveis.

2.3 Notação de Graham

Uma metodologia para catalogar problemas de agendamento foi proposta por Graham em 1979 [33] e posteriormente complementada e estendida por outros autores (ver [9] e [22]). Neste secção far-se-á uma pequena descrição da notação e de algumas extensões posteriores, nomeadamente no que diz respeito ao multiprocessamento que não constava na notação inicial.

Utilizando a notação de Graham, qualquer problema pode ser descrito por um trinómio na forma $\alpha|\beta|\gamma$.

O primeiro campo, $\alpha = \alpha_1\alpha_2$, descreve os processadores. O parâmetro $\alpha_1 \in \{\varnothing, P, Q, R, O, F, J\}$ caracteriza o tipo de processadores e tem o seguinte significado:

$\alpha_1 = 1$: um único processador;

$\alpha_1 = P$: processadores idênticos;

$\alpha_1 = Q$: processadores uniformes;

$\alpha_1 = R$: processadores não relacionados;

$\alpha_1 = O$: processadores dedicados, modo *open shop*;

$\alpha_1 = F$: processadores dedicados, modo *flow shop*;

$\alpha_1 = J$: processadores dedicados, modo *job shop*.

O parâmetro $\alpha_2 \in \{\varnothing, k\}$ representa o número de processadores no sistema:

$\alpha_2 = \varnothing$: número variável de processadores[3];

$\alpha_2 = k$: o número de processadores é k (sendo k um inteiro positivo).

O segundo campo, $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7\beta_8$, descreve as características das tarefas e dos recursos. O parâmetro $\beta_1 \in \{\varnothing, pmtn\}$ indica a possibilidade de preempção:

$\beta_1 = \varnothing$: a preempção não é permitida;

$\beta_1 = pmtn$: a preempção é permitida.

O parâmetro $\beta_2 \in \{\varnothing, res\}$ caracteriza os recursos adicionais:

$\beta_2 = \varnothing$: não existem recursos adicionais;

$\beta_2 = res$: existem restrições relacionadas com recursos.

O parâmetro $\beta_3 \in \{\varnothing, prec, uan, tree, chains\}$ caracteriza as restrições de precedência:

$\beta_3 = \varnothing$: não existem relações de precedência – tarefas independentes;

$\beta_3 = prec$: restrições de precedência genéricas;

$\beta_3 = uan$: restrições de precedência do tipo *uniconnected activity networks*;

$\beta_3 = tree$: restrições de precedência em forma de árvore;

$\beta_3 = chains$: restrições de precedência que formam um conjunto de cadeias.

O parâmetro $\beta_4 \in \{\varnothing, r_j\}$ caracteriza as datas de disponibilidade das tarefas:

$\beta_4 = \varnothing$: todas as datas de disponibilidade são iguais a zero;

[3] O símbolo $\varnothing$ significa "vazio" e é omitido na apresentação dos problemas.

$\beta_4 = r_j$: as datas de disponibilidade variam com a tarefa.

O parâmetro $\beta_5 \in \left\{\varnothing, p_j = p, \underline{p} \leq p_j \leq \overline{p}\right\}$ caracteriza a duração das tarefas:

$\beta_5 = \varnothing$: as tarefas têm durações arbitrárias;

$\beta_5 = \left(p_j = p\right)$: as tarefas têm todas a duração p;

$\beta_5 = \left(\underline{p} \leq p_j \leq \overline{p}\right)$: a duração das tarefas está entre no intervalo $\left[\underline{p}, \overline{p}\right]$.

O parâmetro $\beta_6 \in \left\{\varnothing, \tilde{d}\right\}$ caracteriza as *deadlines*:

$\beta_6 = \varnothing$: não há *deadlines* no sistema, embora possa haver datas de conclusão;

$\beta_6 = \tilde{d}$: existem *deadlines* que devem ser obrigatoriamente respeitadas.

O parâmetro $\beta_7 \in \left\{\varnothing, n_j \leq k\right\}$ caracteriza o número de processadores necessários para completar uma tarefa no caso do sistema *job shop*:

$\beta_7 = \varnothing$: o número de processadores necessários é arbitrário, ou o sistema não é *job shop*;

$\beta_7 = \left(n_j \leq k\right)$: o número de processadores necessários a cada tarefa não é superior a k.

Finalmente, o parâmetro $\beta_8 = \left\{\varnothing, \textit{no–wait}\right\}$ descreve se são permitidas esperas ao programar processadores dedicados:

$\beta_8 = \varnothing$: o espaço de armazenamento entre processadores é ilimitado;

$\beta_8 = \textit{no–wait}$: não há espaço de armazenamento entre processadores, pelo que, sempre que uma tarefa terminar o processamento num processador, deve ser imediatamente transferida para o processador que lhe sucede e o processamento deve ser aí iniciado.

Como a notação original não incluía as questões relacionadas com o multiprocessamento, foi necessário expandir a notação. Drozdowski [22] propõe mais dois parâmetros que denotamos como β_1' e β_1'' e que substituem o campo β_1.

Assim, $\beta_1' \in \left\{\varnothing, spdp\text{ - }lin, spdp\text{ - }lin\text{ - }\delta_j, spdp\text{ - }any, size_j, cube_j, fix_j, set_j\right\}$, descreve o tipo de multiprocessamento:

- $\beta_1' = \varnothing$: o multiprocessamento não é permitido, não sendo possível processar as tarefas em vários processadores simultaneamente;
- $\beta_1' = spdp\text{ - }lin$: o tempo de processamento das tarefas é inversamente proporcional ao número de processadores utilizados ou, por outras palavras, a velocidade de processamento depende linearmente do número de processadores utilizados;
- $\beta_1' = spdp\text{ - }lin\text{ - }\delta_j$: situação idêntica à anterior mas com um limite máximo de processadores a utilizar simultaneamente igual a δ_j;
- $\beta_1' = spdp\text{ - }any$: o tempo de processamento das tarefas depende de uma função arbitrária do número de processadores utilizados;
- $\beta_1' = size_j$: o processamento de cada tarefa deve ser efectuado por um número constante de processadores igual a $size_j$;
- $\beta_1' = cube_j$: é um caso particular da situação anterior e implica que o processamento deve ser feito por um número de processadores igual a uma potência de 2 (isto é, 1, 2, 4, 8, ... processadores);
- $\beta_1' = fix_j$: denota uma situação em que as tarefas devem ser processadas por um conjunto fixo de processadores dedicados;
- $\beta_1' = set_j$: neste caso as tarefas também devem ser processadas por um conjunto predefinido de processadores, havendo vários conjuntos de processadores alternativos.

O parâmetro $\beta_1'' \in \{\varnothing, var, pmtn\}$ especifica se o perfil de processadores utilizados pode variar e se podem ocorrer preempções:

- $\beta_1'' = \varnothing$: não é permitida a preempção de tarefas e o perfil de processadores é fixo;
- $\beta_1'' = pmtn$: o perfil de processadores é fixo mas são permitidas preempções;

$\beta_1'' = var$: o perfil de processadores utilizados é variável. Esta situação é incompatível com, por exemplo, $\beta_1' = fix_j$.

Finalmente, o campo γ define a medida de eficiência do agendamento e pode ser igual a uma das medidas padrão (apresentadas em 2.2.4) ou outra qualquer medida, caso em que $\gamma = X$, tendo X que ser definido por não estar contemplado pela notação.

Esta é a notação vulgarmente utilizada para classificar problemas de agendamento e também será utilizada, sempre que necessário, ao longo do presente trabalho.

Capítulo 3

Partição e Geração de Colunas

Com vista a facilitar a leitura dos capítulos seguintes, neste capítulo é feita uma apresentação genérica dos conceitos e técnicas básicas dos métodos de partição e geração de colunas.

3.1 Introdução

A origem dos métodos de geração de colunas já não é recente. Os princípios básicos da geração de colunas [28], a decomposição de Dantzig-Wolfe [18], foram publicados há mais de quatro décadas. Embora os princípios sejam há muito conhecidos, só mais recentemente este método mostrou ser capaz de encontrar soluções para problemas reais. Para isso contribuiu em grande

medida a associação da decomposição ao método de partição e avaliação e o grande desenvolvimento da capacidade computacional. Desta associação nasceram os métodos de partição e geração de colunas (*branch-and-price*).

Uma das primeiras aplicações práticas com sucesso foi feita em 1961 e 1963 por Gilmore e Gomory sobre o problema de corte (*cutting stock*) [30]. Nesta aplicação a solução era encontrada por arredondamento. Só recentemente, combinando os dois métodos citados, foi possível encontrar soluções para os problemas. Destacam-se os trabalhos de Desrochers et al. (1992) [21] na programação de rotas de veículos (*vehicle routing*), de Vance et al. (1994) [51] no problema de corte ou de Valério de Carvalho (1999) [49] no problema de empacotamento (*bin-packing*). Em Barnhart et al. [5] podemos encontrar uma revisão de literatura sobre a utilização de métodos de partição e geração de colunas em problemas inteiros de grande dimensão.

3.2 Decomposição de Dantzig-Wolfe

Em problemas de programação inteira de grande dimensão aparecem com alguma frequência matrizes de coeficientes tecnológicos esparsas, sendo muito reduzido o número de coeficientes diferentes de zero. Por outro lado, também é frequente aparecerem conjuntos de restrições que apenas respeitam a algumas actividades e formam, dentro da matriz global, uma ou várias matrizes independentes (estrutura angular em blocos). Estes blocos normalmente são ligados por um conjunto de restrições que afectam várias actividades. Considere-se a seguinte figura que ilustra este tipo de estrutura:

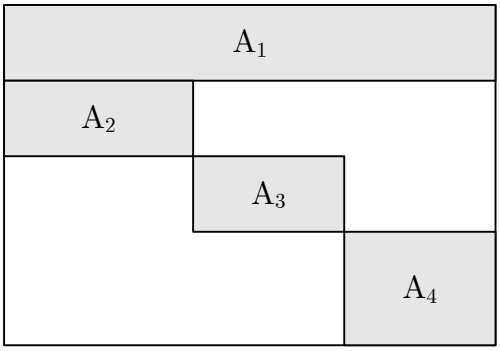


Figura 3.1 - Estrutura angular em blocos

A figura representa a matriz tecnológica de um problema em que apenas nas zonas sombreadas existem coeficientes não nulos. O conjunto de restrições A_1 afecta de uma maneira geral todas as actividades, enquanto que os conjuntos de restrições A_2, A_3 e A_4 apenas respeitam a algumas actividades.

Utilizando a decomposição de Dantzig-Wolfe, poderíamos dividir este problema em quatro partes distintas, obtendo-se um problema principal (*master problem*), onde apenas figurariam as restrições do bloco A_1, e três problemas acessórios (subproblemas), cada um com as restrições de um dos restantes blocos. A vantagem de fazer isto está em ficar com um problema principal mais simples e, eventualmente, aproveitar características da estrutura dos subproblemas para os resolver de forma mais eficiente.

Os vários problemas resultantes ainda são problemas de programação linear e são resolvidos alternadamente até se obter uma solução óptima igual à que seria obtida resolvendo o problema inicial.

Considere-se então a seguinte formulação de um problema de programação linear:

$$\begin{aligned} \max \quad & \sum_{k=1}^{K} c_k x_k \\ s.a \quad & \sum_{k=1}^{K} A_k x_k \leq b \\ & x_k \in X_k, \qquad \forall k \\ & x_k \geq 0, \qquad \forall k \end{aligned}$$

Considere-se ainda que K é o número de blocos que se pretende separar para tirar partido de eventuais características especiais (podendo assumir qualquer valor inteiro igual ou superior a 1), c_k é um vector de coeficientes da função objectivo, x_k um vector de variáveis, A_k uma matriz de coeficientes tecnológicos e que cada conjunto de restrições na forma $x_k \in X_k$ representa o conjunto de restrições que delimitam cada bloco.

De acordo com o teorema de Minkowski (ver [45]) qualquer poliedro X pode ser definido a partir do conjunto dos seus pontos extremos e do conjunto dos seus raios extremos, ambos conjuntos finitos. Designem-se os I pontos extremos por $X_1, X_2, \cdots, X_I$ e os J raios extremos por $R_1, R_2, \cdots, R_J$. Qualquer ponto $x \in X$ pode ser definido por uma combinação convexa de pontos extremos mais uma combinação não negativa de raios extremos.

$$X = \left\{ x \in \mathbb{R}^n_+ : \sum_{i=1}^{I} \lambda_i X_i + \sum_{j=1}^{J} \mu_j R_j, \sum_{i=1}^{I} \lambda_i = 1, \lambda_i \geq 0, \forall i, \mu_j \geq 0, \forall j \right\}$$

Quando o poliedro X é um conjunto fechado, qualquer ponto do poliedro pode ser definido por:

$$x = \sum_{i=1}^{I} \lambda_i X_i, \sum_{i=1}^{I} \lambda_i = 1, \lambda_i \geq 0, \forall i$$

Substituindo esta última expressão no problema original, após algum arranjo dos termos, obter-se-á o seguinte modelo:

$$\begin{aligned}
\max \quad & \sum_{k=1}^{K} \sum_{i=1}^{I_k} \left(c_k X_{k_i} \right) \lambda_{k_i} \\
s.a \quad & \sum_{k=1}^{K} \sum_{i=1}^{I_k} \left(A_k X_{k_i} \right) \lambda_{k_i} \leq b \\
& \sum_{i=1}^{I_k} \lambda_{k_i} = 1, && \forall k \\
& \lambda_{k_i} \geq 0, && \forall k, \forall i
\end{aligned}$$

O número de pontos extremos de um poliedro é exponencialmente grande, não sendo prático fazer a sua enumeração à partida. No entanto, como se vai mostrar, não é necessária a enumeração explícita de todos os pontos para determinar a solução óptima, podendo esta enumeração ser feita de forma

implícita. É neste ponto que se recorre aos vários subproblemas criados para determinar quais os pontos extremos cuja introdução no problema principal é mais vantajosa.

Começando com uma qualquer solução admissível, a avaliação dos pontos extremos que ainda não estão no problema principal (problema primal restrito) é feita com base na solução óptima deste problema.

Denotem-se os coeficientes da solução dual associados às primeiras m restrições do problema primal restrito por $\pi_1, \pi_2, \cdots, \pi_m$ e o coeficiente associado à restrição de convexidade de cada um dos blocos por υ_k, tal que $c_B B^{-1} = [\pi, \upsilon]$, sendo $\pi = [\pi_1, \pi_2, \cdots, \pi_m]$ e sendo $\upsilon = [\upsilon_1, \upsilon_2, \cdots, \upsilon_k]$.

É sabido que o custo reduzido de uma variável é dado por $c_B B^{-1} A_j - c_j$. Para as colunas que correspondam a pontos $x_k \in X_k$, o vector A_j tem a seguinte composição: nos primeiros m elementos são iguais a $A_k x_k$, sendo os restantes elementos nulos, à excepção do elemento correspondente à restrição de convexidade do domínio X_k, que é unitário.

Então, para uma coluna deste tipo, o custo reduzido será dado por:

$$\begin{aligned} c_B B^{-1} A_j - c_j &= \pi \cdot A_k x_k + 1 \cdot \upsilon_k - c_k x_k \\ &= (\pi A_k - c_k) x_k + \upsilon_k \end{aligned}$$

Para determinar qual o melhor ponto de X_k para eventual introdução no problema primal restrito basta resolver o seguinte problema:

$$\begin{aligned} \min \quad & (\pi A_k - c_k) x_k + \upsilon_k \\ s.a \quad & x_k \in X_k \end{aligned}$$

As soluções geradas nos subproblemas são pontos extremos dos poliedros e podem ser incluídas como novas variáveis no problema primal restrito. Enquanto os subproblemas conseguirem gerar soluções com óptimos negativos, as novas variáveis são candidatas a entrar na base do problema primal restrito. Quando os subproblemas não conseguirem gerar variáveis atractivas, estamos perante a solução óptima do problema principal.

3.3 Geração de Colunas

Uma vez efectuada a decomposição torna-se necessário iniciar e coordenar o processo de geração de novas colunas com vista à obtenção de uma solução óptima para o problema principal. O processo pode ser esquematicamente representado da seguinte forma:

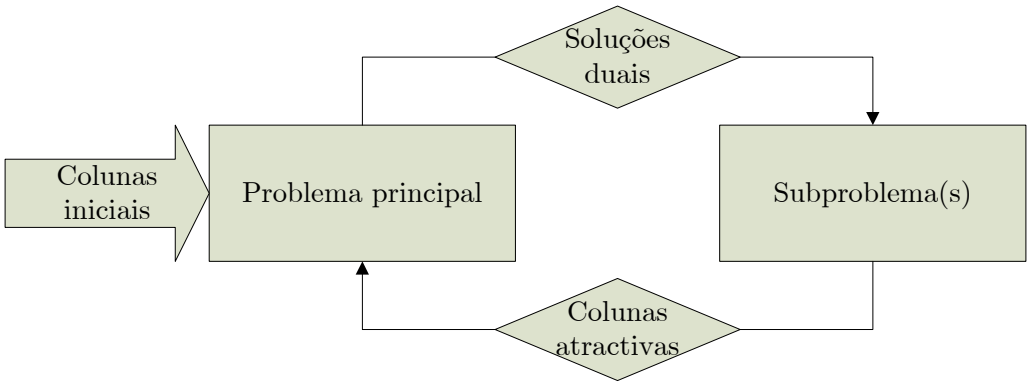


Figura 3.2 - Articulação do processo de geração de colunas

Para iniciar o processo de resolução é necessário introduzir no problema primal restrito uma base admissível. Normalmente é utilizada uma solução inicial de boa qualidade obtida através de um qualquer método heurístico. No entanto, na falta de melhor, pode ser utilizada uma matriz identidade, com penalizações adequadas nos coeficientes da função objectivo.

Seguidamente o problema principal é optimizado e a solução dual é passada para o subproblema. É então optimizado o subproblema correspondendo a solução obtida à coluna mais atractiva que é possível introduzir no problema principal. Se esta coluna for atractiva do ponto de vista do problema principal, isto é, se o seu custo reduzido for negativo, o problema é novamente optimizado e o ciclo repete-se. Quando o subproblema já não for capaz de identificar novas colunas atractivas, é porque a solução óptima do problema principal já foi encontrada.

3.4 Partição e Avaliação

Quando os problemas de programação linear incluem variáveis inteiras é comum relaxar as condições de integralidade numa fase inicial para as voltar a impor, se for necessário, numa fase posterior. O que normalmente acontece é que, após a optimização da chamada *relaxação linear*, é obtida uma solução inválida no que respeita às condições de integralidade.

A forma de resolver este problema consiste em impor sucessivamente restrições adicionais no problema primal restrito de modo a que as soluções que não respeitem as condições de integralidade sejam progressivamente eliminadas.

O método mais comum de proceder a essa eliminação de soluções inválidas é o método de partição e avaliação, conhecido na literatura anglo-saxónica por *branch-and-bound*. Neste método, a partir de um problema (nodo) cuja solução é inválida são criados dois novos problemas (ramos) em tudo iguais aos originais, aos quais se adicionam duas novas restrições (uma a cada problema) que dividem o espaço de soluções em duas partes que não se intersectam e, ao mesmo tempo, cortam um conjunto de soluções inválidas. Uma vez impostas as restrições, ambos os problemas são optimizados e é verificada a validade das soluções. O processo é repetido em cada um dos problemas criados até que todos os problemas sejam resolvidos ou se conclua que não é possível encontrar melhores soluções que a melhor existente.

Quando o processo se inicia pode já ser conhecido o valor de uma solução válida obtida por outro método. Esta solução é designada por *incumbente* e, em cada nodo, serve de termo de comparação para avaliar o interesse em prosseguir com a partição. De facto, se se puder concluir que, num determinado nodo, não é possível encontrar uma solução melhor que a incumbente, a exploração desse nodo pode ser abandonada.

A figura que se segue pretende ilustrar o que acontece quando se impõe uma restrição de partição:

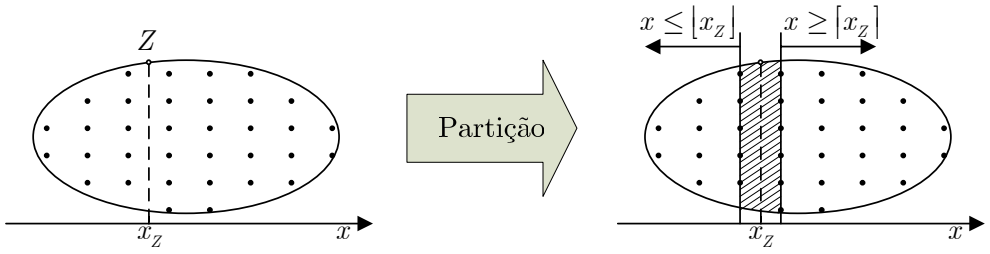


Figura 3.3 - Introdução de restrição de partição

Considere-se a relaxação de um problema em que o espaço de soluções é a área representada pela elipse. Os pequenos pontos representam a soluções que cumprem as condições de integralidade. Considere-se que a solução óptima da relaxação é o ponto Z (solução inválida). Nesse ponto o valor da variável x (que deve ser inteira) é um qualquer valor fraccionário x_Z. Uma forma de partição é criar dois novos problemas em que num deles se impõe a restrição $x \leq \lfloor x_Z \rfloor$ e no outro a restrição $x \geq \lceil x_Z \rceil$. Note-se que o resultado são dois problemas cujos espaços de soluções são mutuamente exclusivos, e que foi eliminado um conjunto de soluções inválidas (parte tracejada) sem eliminar nenhuma das válidas. Logo, a solução inteira óptima tem que estar contida num dos dois problemas criados. Se o processo de partição for aplicado sucessivamente é garantido que se irá encontrar a solução inteira óptima.

A árvore de pesquisa resultante da aplicação das restrições de partição pode ser percorrida segundo várias estratégias. As mais comuns são:

- pesquisa em profundidade, (*depth first search*) em que após cada partição se explora um dos novos nodos criados, aumentando sucessivamente a profundidade da pesquisa até ser encontrada uma solução admissível ou se concluir pelo abandono do nodo. Esta estratégia facilita a rápida obtenção de soluções válidas;
- pesquisa em largura, (*breadth first search*) em que se avaliam arbitrariamente todos os nodos de um determinado nível antes de se passar ao nível seguinte. Existe um senão computacional que se prende

com a necessidade de guardar informação sobre um grande número de nodos, mesmo para profundidades reduzidas;

– pesquisa pelo melhor valor, (*best first search*) em que se explora sempre em primeiro lugar o nodo que, entre todos os disponíveis nos vários níveis, tiver o melhor valor óptimo. Pode ter os mesmos inconvenientes computacionais que a pesquisa em largura.

Como é óbvio podem ser concebidas estratégias híbridas ou que incluam outros pormenores. A escolha da estratégia mais adequada para cada problema é normalmente feita recorrendo a testes de eficiência computacional.

3.5 Partição e Geração de Colunas

Quando de aplica a decomposição de Dantzig-Wolfe a problemas inteiros ou mistos em que as condições de integralidade são relaxadas, as soluções obtidas após a optimização, não respeitam normalmente aquelas condições. Neste caso, a aplicação da partição e avaliação não garante que a solução óptima seja encontrada porque, como as colunas foram enumeradas apenas implicitamente, colunas que não chegaram a ser introduzidas no problema primal restrito, podem ser atractivas à luz das novas restrições. Pode mesmo acontecer que colunas necessárias à solução óptima inteira não tenham sido geradas.

Isto implica que na determinação da solução óptima de cada nodo seja necessário verificar se há colunas ainda não enumeradas que sejam atractivas. Os métodos de partição e geração de colunas (*branch-and-price*) tratam precisamente destes processos. Na figura seguinte esquematiza-se o método:

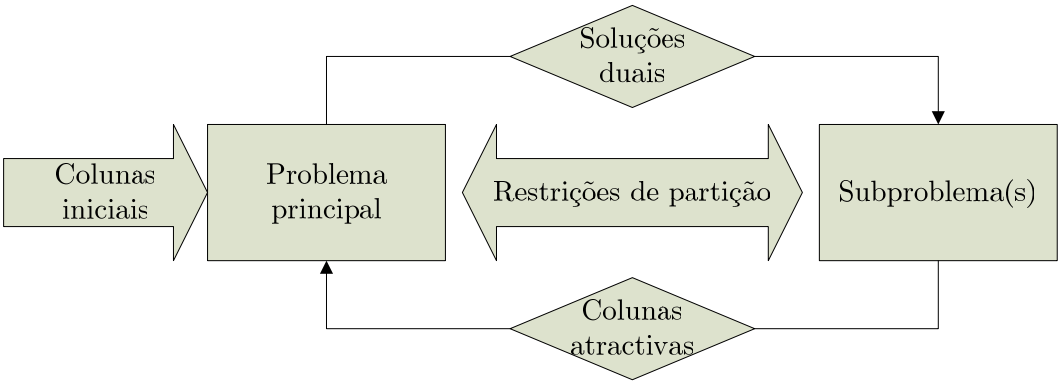


Figura 3.4 - Método de partição e geração de colunas

Após a obtenção de uma solução para a relaxação linear é necessário começar a acrescentar as restrições de partição. É aqui que residem as grandes dificuldades deste método: ao acrescentar uma nova restrição de partição, não só a estrutura do problema principal é modificada, mas também a estrutura dos subproblemas, que devem continuar a identificar correctamente as colunas atractivas.

Normalmente, a decomposição é feita de forma a que os subproblemas tenham determinadas características que os tornam de fácil resolução. Ao modificar a sua estrutura, o subproblema pode deixar de ter aquelas características e tornar-se num problema de programação inteira ou mista genérico, de difícil resolução.

3.6 Aceleração de Processos de Partição e Geração de Colunas

Os processos de geração de colunas são conhecidos por permitirem obter soluções de boa qualidade num curto espaço de tempo (soluções a distâncias de 1, 5 ou 10% do óptimo, dependendo da aplicação, podem ser perfeitamente aceitáveis). No entanto, para determinar a solução óptima do problema pode ser necessário muito tempo devido a uma fase final de convergência tipicamente longa.

Descrevem-se em seguida alguns métodos genéricos que costumam conduzir a resultados práticos interessantes no campo da aceleração dos processos de partição e geração de colunas.

3.6.1 Estabilização Dual

Foi notado, nos processos de geração de colunas, que o valor das variáveis duais vai variando de forma errática à medida que se vão acrescentando colunas ao problema primal restrito. Du Merle et al. [24] apresentaram uma estratégia de estabilização dos valores das variáveis duais que pode conduzir a acelerações significativas no processo de geração de colunas.

Basicamente, o processo consiste em impor penalizações na função objectivo quando as variáveis duais saem de um determinado intervalo. Os melhores valores para as penalizações e para os limites dos intervalos podem ser ajustados por via experimental.

3.6.2 Introdução de Cortes Duais

A utilização de um espaço dual mais limitado pode ajudar a encurtar os processos de geração de colunas. Apenas a título de exemplo, se for possível substituir restrições de igualdade por restrições de desigualdade (em alguns casos particulares é sempre possível transformar soluções com folga em soluções sem folga) está-se a proceder a uma redução do espaço dual.

Com base em ideias semelhantes Valério de Carvalho [50] conseguiu derivar uma família de cortes duais válidos para acelerar o processo de geração de colunas no problema de corte.

3.6.3 Outras Estratégias

Em [20] são descritas algumas estratégias de aceleração para problemas de planeamento de rotas e escalonamento de tripulações, algumas das quais podem ser utilizadas genericamente.

Antes de iniciar o processo de resolução de problemas há um conjunto de operações de simplificação que em certos casos podem ser feitas. As mais comuns são a obtenção de boas soluções primais e duais, redução do espaço de soluções do subproblema por recurso a regras de dominância, agregação, sequenciação de tarefas em problemas de programação ou divisão dos problemas em problemas independentes mais pequenos.

No subproblema pode ser possível restringir as redes utilizadas (na programação dinâmica, por exemplo) devido a questões de dominância ou ignorar temporariamente partes dessas redes enquanto estiverem a ser geradas colunas atractivas. No caso de o número de subproblemas ser grande, pode ser possível agregar subproblemas parecidos, reduzindo o número de colunas iguais geradas.

No problema principal, quando o número de colunas presente for grande, pode fazer-se a eliminação de algumas colunas para que o problema de torne computacionalmente mais leve. Igualmente, restrições com pouca probabilidade de serem violadas podem ser temporariamente relaxadas, sendo reintroduzidas se necessário. Devem também ser experimentados, se disponíveis, os vários algoritmos de resolução dos pacotes de programação linear (primal, dual, pontos interiores, etc.) para determinar qual o que produz melhores resultados.

Durante a partição e avaliação podem ser definidas tolerâncias para proceder ao corte precoce de partes da árvore de pesquisa. Num problema de minimização, um nodo seria abandonado se o valor do seu limite inferior fosse maior ou igual que o valor da melhor solução encontrada subtraída da tolerância. A estratégia não garante que seja encontrada a solução óptima, mas garante que a melhor solução encontrada está a uma distância admissível daquela.

Depois de encontrada a melhor solução na árvore de pesquisa, pode ser possível, através de heurísticas desenvolvidas para o efeito, proceder à procura de soluções melhores na vizinhança ou até a alterações de algumas características das soluções que possam causar problemas operacionais.

Não há garantias que uma dada estratégia de aceleração resulte em ganhos na totalidade dos problemas onde pode ser aplicada. Por isso, só a experimentação pode dizer que estratégias devem ser mantidas ou abandonadas num determinado caso concreto.

Capítulo 4

Revisão de Literatura

Apresenta-se neste capítulo uma síntese dos principais trabalhos publicados sobre agendamento de máquinas paralelas. Numa primeira fase são abordados alguns métodos heurísticos referidos correntemente. Numa outra parte são descritas metodologias de resolução exacta de determinados problemas, com destaque para as metodologias de partição e geração de colunas.

4.1 Introdução

Os problemas de máquinas paralelas ocorrem frequentemente em ambientes industriais, sendo grande o interesse da comunidade académica pelo seu estudo. Cheng e Sin [16] e Hoogeveen et al. [34] apresentaram revisões de litera-

tura sobre este tipo de problemas. Pinedo [46] apresenta vários casos concretos em que há resultados de regras de decisão e de heurísticas. Ao longo deste capítulo é referido um conjunto de problemas para os quais há trabalho científico e que pelas suas características estão de alguma forma relacionados com este trabalho.

4.2 Métodos Heurísticos

Este tipo de métodos tem grande aceitação porque resolve, embora não de forma exacta mas muitas vezes de forma satisfatória, problemas difíceis e/ou problemas de grande dimensão. Por norma, os métodos heurísticos baseiam-se em regras de decisão, pesquisa local ou enumeração.

Para o problema $Pm\Big|\Big|C_{\max}$ existe a regra LPT, *longest processing time first*, em que, sempre que uma máquina é libertada, se agenda a tarefa com o maior tempo de processamento. A regra não garante a solução óptima, mas demonstra-se que a seguinte desigualdade é verdadeira (ver [32]):

$$\frac{C_{\max}(LPT)}{C_{\max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$$

$C_{\max}(LPT)$ é o valor da solução conseguida utilizando a regra LPT, e $C_{\max}(OPT)$ é o valor da solução óptima. Como a desigualdade é verdadeira, é garantido que a regra LPT, sob nenhuma circunstância, produz uma solução com um valor superior em mais de 33% (aproximadamente) ao valor da solução óptima.

Se se alterar a medida de eficiência e se considerar o problema $Pm\Big|\Big|\sum C_j$, existe uma regra que produz soluções óptimas. Trata-se da regra SPT, *shortest processing time first* (ver [17]). Agendar de acordo com esta regra significa agendar primeiro as tarefas com menores tempos de processamento. Se o problema for $Pm\Big|\Big|\sum w_j C_j$ pode ser utilizada a regra WSPT, *weighted shortest processing time*. Segundo esta regra, deve ser agendada em primeiro

lugar a tarefa com o maior quociente w_j/p_j. A regra WSPT não garante uma solução óptima, mas é demonstrável a seguinte desigualdade (ver [37]):

$$\frac{\sum w_j C_j\,(WSPT)}{\sum w_j C_j\,(OPT)} < \frac{1}{2}\left(1+\sqrt{2}\right)$$

Este resultado garante que, mesmo no pior caso, o valor da solução WSPT não é superior em mais de 21% (aproximadamente) ao valor da solução óptima, o que torna a regra WSPT numa heurística interessante.

Se forem permitidas preempções, de uma maneira geral, os problemas tornam-se mais fáceis de resolver. Para o problema $Pm|pmtn|C_{\max}$, designando a tarefa mais longa por p_1, demonstra-se que

$$C^*_{\max} = \max\left(p_1, \frac{1}{m}\sum_{j=1}^{n} p_j\right)$$

Conhecendo à partida o valor da solução óptima, é relativamente fácil construir um algoritmo que determine o agendamento óptimo. Esse algoritmo, devido a McNaughton [44], é composto por 3 passos:

(i) Agendar as n tarefas numa única máquina e numa sequência arbitrária. O valor de $C_{\max}$ resultante deverá ser inferior ou igual a $mC^*_{\max}$;

(ii) Dividir este agendamento em m intervalos de tempo de tamanho igual a $C^*_{\max}$;

(iii) Cada intervalo de tempo conterá o agendamento de cada uma das máquinas.

Ainda para este problema há uma outra regra de agendamento que é óptima. Trata-se da regra LRPT, *longest remaining processing time first*. Esta regra é a versão preemptiva da regra LPT. A utilização desta regra, em geral, provoca um número de preempções infinito, o que, na prática, invalida a sua utilização. No entanto, este problema pode ser ultrapassado se se considerar que a máquina só pode ser interrompida em momentos de tempo discretos. O problema do número de preempções infinito é ultrapassado e demonstra-se que a regra continua a produzir soluções óptimas.

Generalizando ao ambiente $Qm|pmtn|C_{\max}$ há a regra LRPT-FM, *longest remaining processing time on the fastest machine*. A regra especifica que a tarefa cujo tempo de processamento restante seja maior, deve ser agendada na máquina mais rápida. No limite, a regra LRPT-FM igualiza todos os tempos de processamento restantes e faz com que as tarefas mudem constantemente de máquina, provocando um número infinito de preempções. Embora produza soluções óptimas a regra LRPT-FM também não produz resultados úteis do ponto de vista prático. Mais uma vez, impondo que as preempções só podem ocorrer em momentos discretos resolve o problema: o número de preempções deixa de ser infinito e as soluções produzidas são óptimas.

Se a regra LRPT-FM for aplicada às tarefas disponíveis a cada momento também se demonstra que a regra produz soluções óptimas no ambiente $Qm|r_j, pmtn|C_{\max}$.

Para uma discussão mais detalhada sobre as heurísticas LRPT e LRPT-FM ver Pinedo [46], onde são referidos os trabalhos de Horvath et al. [36] e de Gonzalez et al. [31].

Considere-se agora o problema $Qm|pmtn|\sum C_j$. Este problema é uma generalização do ambiente com máquinas paralelas idênticas. A análise deste problema levou à formulação da regra SRPT-FM, *shortest remaining processing time on the fastest machine*. De acordo com esta regra, a tarefa com menor tempo de processamento restante deve ser colocada na máquina mais rápida, a segunda menor na segunda máquina mais rápida, etc. As preempções ocorrem quando a máquina mais rápida acaba o processamento, momento em que todas as tarefas em processamento são transferidas para a próxima máquina mais rápida, iniciando-se uma tarefa (se existir) na máquina mais lenta. Ou seja, a aplicação da regra conduz a um número de preempções finito. Também se demonstra que a regra SRPT-FM conduz a soluções óptimas (ver [46], [40] e [31] para um tratamento mais detalhado).

Finalmente, os problemas com critérios de eficiência relacionados com as datas de conclusão. Considere-se o problema $Qm|pmtn|L_{\max}$. Este problema é uma generalização do ambiente de máquinas idênticas. Pretende-se verificar

se é possível agendar todas as tarefas com $L_{\max} = z$, ou seja, qualquer momento de conclusão C_j deve ser inferior ou igual a $d_j + z$, que funciona como uma *deadline*. A resposta a esta questão pode ser dada resolvendo o problema $Qm|r_j, pmtn|C_{\max}$. Imagine-se que a direcção do tempo foi invertida. Tomem-se as *deadlines* do problema original como datas de disponibilidade e aplique-se a regra LRPT-FM ao contrário, começando pela última *deadline*. Se todas as tarefas forem agendadas depois do momento zero, significa que é possível fazer o agendamento com $L_{\max} = z$. Para encontrar o valor óptimo de $L_{\max}$ basta uma pesquisa simples do melhor valor de z.

A técnica das *deadlines* também pode ser utilizada no problema $Qm|r_j, pmtn|L_{\max}$, ou seja, cada data de conclusão d_j ser substituída por uma *deadline* $d_j + z$. Neste caso, inverter o problema não traz qualquer vantagem, pois resulta num problema do mesmo tipo, com datas de disponibilidade e datas de conclusão.

Para mais detalhes sobre estes dois problemas ver Bruno et al. [10] e Labetoulle et al. [38].

O conjunto de exemplos apresentados nos parágrafos anteriores demonstra como a área do processamento paralelo é um campo de estudo vasto e aberto, uma vez que há uma grande variedade de situações que podem ocorrer: com / sem preempção, com / sem precedências, com / sem tempos de preparação de máquinas dependentes da sequência, processadores idênticos / uniformes / não relacionados, etc.

No entanto, apenas existem resultados para modelos que pouco têm a ver com o problema apresentado neste trabalho e que poderemos classificar como modelos simples, porque não prevêem, por exemplo, o multiprocessamento. No entanto, quase sempre, a partir da análise de modelos simples surgem ideias e princípios que podem ser aplicados em modelos mais complexos.

4.3 Métodos Exactos

Os métodos de resolução exactos são baseados normalmente em programação dinâmica, em partição e avaliação ou em partição e geração de colunas. Embora sobre os dois primeiros métodos haja alguns resultados (por exemplo, Amico e Martello apresentam em [19] um algoritmo de partição e avaliação para minimizar o tamanho do agendamento) aplicam-se sobretudo a problemas de pequena dimensão ou de complexidade limitada, uma vez que os aspectos combinatórios deste tipo de problemas não permitem a resolução de problemas de maior dimensão ou de maior complexidade. Devido a este facto, esta secção será dedicada a trabalhos onde tenham sido aplicados métodos de partição e geração de colunas ao agendamento de máquinas paralelas, sem prejuízo de poder ser referido um ou outro trabalho com interesse que não tenha sido resolvido por um destes métodos.

4.3.1 Problemas com Funções Objectivo Aditivas

Akker et al. em [2] concluem que os problemas de programação de máquinas paralelas com funções objectivo aditivas (em que cada máquina tem a sua contribuição para o custo total e esta é independente das agendas das outras máquinas) colocam um grande problema computacional devido à grande dificuldade em calcular bons limites inferiores. Por exemplo o problema $P||\sum w_j C_j$ (minimização do tamanho do agendamento ponderado) recebe atenção dos investigadores desde 1964 e só em 1997 foi proposta no trabalho referido acima uma metodologia baseada em geração de colunas capaz de resolver instâncias razoáveis (100 trabalhos em 10 máquinas).

4.3.1.1 Problema $P||\sum w_j C_j$

O esquema proposto por Akker et al. em [2] para a resolução deste problema é baseado no seguinte teorema:

Teorema 4.1 (Elmaghraby e Park, [25])

Existe uma sequência óptima com as seguintes propriedades:

(i) As tarefas são processadas continuamente, com início no instante zero e nenhuma máquina fica livre antes de todos os trabalhos se terem iniciado.

(ii) A última tarefa de qualquer máquina é completada entre H_{min} e H_{max}, com

$$H_{min} = \frac{\sum_{j=1}^{n} p_j - (m-1)\cdot \max\left(p_j\right)}{m} \text{ e}$$

$$H_{max} = \frac{\sum_{j=1}^{n} p_j + (m-1)\cdot \max\left(p_j\right)}{m}.$$

(iii) Em cada máquina as tarefas são sequenciadas por ordem não crescente dos rácios w_j/p_j.

(iv) Se $w_j \geq w_k$ e se $p_j \leq p_k$, então existe uma sequência óptima na qual a tarefa T_j nunca é iniciada depois da tarefa T_k. ■

Com base nas três primeiras propriedades, é possível resolver o problema através de programação dinâmica com um algoritmo que corre em $O\left(n\left(\sum p_j\right)^{m-1}\right)$ quer em termos de tempo, quer em termos de espaço, o que não permite um número de máquinas muito alto nem tempos de processamento muito longos.

No esquema proposto para a geração de colunas, o problema ficou com a seguinte definição matemática:

$$\begin{aligned} min \quad & \sum_{s\in S} c_s x_s \\ s.t. \quad & \sum_{s\in S} x_s = m \\ & \sum_{s\in S} a_{js} x_s = 1, \quad j = 1,2,\cdots,n \\ & x_s \in \{0,1\}, \quad \forall s \in S \end{aligned}$$

Nesta formulação, s refere-se à agenda para uma máquina. Essa agenda é definida por um vector $\left[a_{1s}, a_{2s}, \cdots, a_{js}\right]$, em que cada elemento do vector é igual a 1 se a tarefa T_j estiver na agenda s e igual a 0 caso contrário. O valor de c_s representa a contribuição dessa agenda para o custo total e, como a função objectivo é aditiva, pode ser calculado independentemente de todas as outras agendas. Existe uma restrição que garante que todas as máquinas são utilizadas (para eliminar trivialidades pressupõe-se que $n > m$) e uma outra que garante que cada tarefa é agendada apenas uma única vez no conjunto de todas as máquinas. Finalmente a restrição binária garante que determinada agenda ou pertence ($x_s = 1$) ou não pertence ($x_s = 0$) a determinada solução. Numa fase inicial, esta última restrição é relaxada para a forma $0 \leq x_s \leq 1$.

O subproblema utilizado para gerar as novas agendas consiste em calcular um novo vector de valores a_{js} e é utilizado um esquema de programação dinâmica para o resolver. Esse algoritmo aproveita as propriedades descritas no Teorema 4.1 e corre em $O\left(n\sum p_j\right)$, quer em termos de tempo, quer em termos de espaço.

Se a solução resultante da resolução da relaxação linear não for inteira ou não puder ser convertida para inteira (em certas condições identificadas no trabalho, tal era possível) tem que se aplicar um esquema de partição para obter a solução óptima.

O esquema consiste em identificar, dentro das soluções fraccionárias, a tarefa fraccionária (é definida uma condição que qualifica as tarefas como fraccionárias) de menor índice w_j/p_j. Sobre estas tarefas é imposta um partição binária, em que num dos ramos é imposta a condição $C_j \leq \min\left(C_j^*\right)$ e no outro ramo a condição $C_j \geq \min\left(C_j^*\right) + 1$.

O esquema de partição é aplicado recorrentemente até que não haja qualquer tarefa fraccionária, caso em que é garantido que a solução ou é inteira ou se pode converter numa solução inteira.

Este esquema de partição tem a vantagem de não implicar alterações de vulto no subproblema, podendo este continuar a ser resolvido pelo algoritmo de programação dinâmica proposto.

Também Chen e Powell [15] propuseram um esquema semelhante para resolver este problema. Estes autores obtiveram um problema primal restrito (igual ao apresentado atrás) através da decomposição de Dantzig-Wolfe aplicada a um modelo que também apresentam. A partição por eles sugerida incide sobre as variáveis originais desse modelo. As alterações induzidas no subproblema implicam a utilização de um algoritmo de resolução mais complexo durante a fase de partição.

Em termos computacionais, os resultados são semelhantes, uma vez que ambos os métodos conseguem resolver em tempos aceitáveis instâncias com 100 tarefas e 10 processadores.

No entanto, a metodologia proposta por Chen e Powell tem como grande vantagem o facto de ser mais genérica e no mesmo trabalho ([15]) eles também apresentam resultados para os problemas $Q||\sum w_j C_j$, $R||\sum w_j C_j$, $P||\sum w_j U_j$, $Q||\sum w_j U_j$ e $R||\sum w_j U_j$, afirmando ainda que a metodologia é extensível a quaisquer problemas do mesmo género (máquinas idênticas, uniformes ou não relacionadas) que tenham funções objectivo aditivas.

4.3.1.2 Problema $P\left|d_j = d\right|\sum\left(u_j E_j + v_j T_j\right)$

O interesse neste problema aparece com a cada vez maior aceitação e implementação de sistemas de gestão da produção do tipo *just-in-time*. Neste problema existe uma data de conclusão, d, comum a todas as tarefas e não restritiva do agendamento e penalidades associadas aos adiantamentos, $E_j = \max\left(0, d_j - C_j\right)$, e aos atrasos, $T_j = \max\left(0, C_j - d_j\right)$, do agendamento.

Chen e Powell [13] propõem um algoritmo baseado em geração de colunas para a resolução exacta de problemas deste tipo. Neste trabalho é apresentada uma formulação de programação inteira que é depois tratada através da decomposição de Dantzig-Wolfe. O problema principal e subproblemas resul-

tantes são resolvidos pela metodologia básica da geração de colunas. Desta forma, conseguiram resolver em tempo aceitável problemas com 6 processadores e 60 tarefas.

Em 2002, Chen e Lee [11] generalizaram a abordagem anterior para o caso em que, em vez de uma data de conclusão comum, existe uma janela de conclusão comum (sendo o problema apresentado anteriormente um caso particular deste). Também aqui foi aplicada a decomposição de Dantzig-Wolfe a uma formulação original e o problema foi resolvido utilizando a geração de colunas. Através do algoritmo proposto conseguiram resolver problemas com 6 máquinas e 40 tarefas.

4.3.2 Problemas com Tempos de Preparação

A introdução de tempos de preparação nas mudanças de tarefa representa uma complicação adicional nos problemas de programação. Resoluções exactas de problemas deste tipo são extremamente escassas na literatura sobre agendamento de máquinas paralelas.

Os tempos de preparação podem ser de dois tipos: dependentes da sequência de tarefas processadas ou independentes daquela sequência. Chen e Powell [14] apresentaram um modelo baseado em decomposição e geração de colunas para lidar com os dois tipos de problema. No entanto, as tarefas não foram consideradas individualmente, mas sim em famílias, incorrendo-se em tempos de preparação quando se muda de uma tarefa de determinada família para uma tarefa de outra família. Note-se porém que cada família pode conter apenas uma tarefa, o que faz dos problemas em que as tarefas são consideradas individualmente um caso particular do problema com tarefas agrupadas em famílias.

No trabalho que apresentaram, Chen e Powell conseguiram resultados para os problemas $P|s_u|\sum w_j C_j$, $P|s_{uv}|\sum w_j C_j$, $P|s_u|\sum w_j U_j$ e para $P|s_{uv}|\sum w_j U_j$. Note-se que s_u se refere a tempos de preparação independentes da sequência e que s_{uv} se refere a tempos de preparação dependentes da sequência. Nesta

notação o índice u refere-se a determinada família, variando entre 1 e b, sendo este último o número de famílias do problema.

Relativamente aos tempos de preparação dependentes da sequência é assumido, como é corrente neste tipo de problemas, que os tempos de preparação verificam a chamada "desigualdade triangular". Isto é, para quaisquer valores de u, v e z, verifica-se a desigualdade $s_{uv} + s_{vz} \geq s_{uz}$.

Os problemas foram resolvidos com recurso à metodologia padrão da geração de colunas, aproveitando propriedades existentes na estrutura dos subproblemas para os resolver de forma computacionalmente eficiente. Assim, apresentaram resultados computacionais aceitáveis para instâncias até 40 tarefas, 6 processadores e 8 famílias.

Em 2004, Lopes [42] resolveu através de geração de colunas um problema de programação de máquinas paralelas não idênticas com tempos de preparação dependentes da sequência e datas de disponibilidade para as tarefas e para os processadores de forma a minimizar a soma ponderada dos atrasos ou, matematicamente, $R\left|a_i, r_j, s_{ij}\right|\sum w_j D_j$.

Neste trabalho é proposta uma formulação de programação linear à qual é aplicada a decomposição de Dantzig-Wolfe. Cada variável do problema reformulado consiste numa sequência de afectação de tarefas a uma máquina, ou seja, consiste na agenda de uma máquina. A resolução dos subproblemas resulta na geração de novas agendas para os diferentes tipos de máquinas e o algoritmo utilizado é baseado em programação dinâmica.

Para efectuar a partição é utilizada uma formulação de fluxos numa rede e as variáveis de partição correspondem a fluxos nos arcos dessa rede.

Em termos computacionais são apresentados resultados para instâncias de dimensão considerável que foram resolvidas em tempo aceitável: instâncias difíceis com 50 máquinas e 180 tarefas e instâncias mais fáceis com 50 máquinas e 220 tarefas.

4.3.3 Problemas com Multiprocessamento

Um pressuposto comum na literatura sobre planeamento e agendamento é o de que uma tarefa, em qualquer instante, só pode ser processada numa única máquina (ver por exemplo [7]). No entanto, há um grande número de aplicações onde este pressuposto não é adequado. Por exemplo, se houver um conjunto de máquinas iguais e as tarefas forem divisíveis, não há qualquer problema em processar determinada tarefa em mais que um processador simultaneamente.

Drozdowsky [22] identifica vários tipos de multiprocessamento. Em alguns modelos as tarefas podem requerer um conjunto específico de processadores. Noutros modelos pode haver conjuntos de processadores alternativos ou mesmo total liberdade de escolha dos processadores utilizados e do seu número. Também aqui pode ser ou não admitida preempção e o perfil de processadores utilizado pode ser fixo ou variável. Em resumo, dentro desta classe de problemas existem vários tipos de problemas com diferenças entre si muito assinaláveis.

Devido a este facto, as referências científicas a esta classe de problemas não são muitas e as que aparecem são maioritariamente dedicadas a esquemas de aproximação ao óptimo e a heurísticas e, por norma, apenas se aplicam a casos muito específicos. Quanto a esquemas de resolução exacta referem-se alguns que apenas são aplicáveis quando o número de processadores é conhecido e pequeno (normalmente 2 ou 3 processadores).

Este é o caso do trabalho apresentado por Bianco et al. em [6], em que são propostos algoritmos para a resolução em tempo polinomial de problemas com tarefas sujeitas a datas de disponibilidade e com máquinas disponíveis apenas durante determinadas janelas de tempo. Adicionalmente, as tarefas teriam de ser agendadas num conjunto predefinido de processadores dedicados. Mesmo nestas condições severamente restritivas, apenas são apresentados algoritmos polinomiais para minimizar o tamanho do agendamento (C_{max}) e o atraso máximo (L_{max}) para as situações de 2 e 3 processadores. É também apresentada uma formulação de programação linear genérica para

quando o número de processadores é conhecido à partida, embora não sejam fornecidos quaisquer resultados computacionais.

Chen e Lee [12] apresentam um esquema de aproximação ao óptimo para o problema da minimização de C_{max} num ambiente em que as tarefas devem ser executadas num conjunto predeterminado de processadores, havendo vários conjuntos alternativos. O algoritmo que propõem tem duas etapas: numa primeira etapa é escolhida a combinação de processadores que executará a tarefa e, numa segunda fase, é determinado o agendamento propriamente dito. Os autores demonstram que o algoritmo proposto fornece resultados exactos para os casos de 2 e 3 processadores.

Mais recentemente, Blazewicz et al. em [8] resolveram o problema de agendar um conjunto de tarefas maleáveis em $m \geq n$ processadores, definindo "tarefa maleável" como uma tarefa que a qualquer momento pode ser processada por um número arbitrário de processadores. O tempo de processamento dependia de uma função não linear do número de processadores utilizados, tendo sido experimentados vários tipos de funções. O objectivo do trabalho era encontrar a melhor forma (minimização de C_{max}) de agendar trabalhos de processamento computacional que podiam ser executados em paralelo em várias máquinas.

Capítulo 5

Formulação do Problema

Neste capítulo é feita uma descrição detalhada do problema, é proposta uma formulação matemática do mesmo e é apresentado um esquema de resolução genérico. Assim, na secção 5.1 é feita uma descrição pormenorizada do problema. Na secção seguinte (5.2) é apresentada uma formulação do problema baseada no problema de fluxo de custo mínimo. Depois, na secção 5.3 é proposto um modelo de programação inteira para a resolução exacta do problema. Como este modelo de programação inteira tem um número exponencialmente grande de variáveis, é proposta nas secções 5.4 e 5.5 uma metodologia de resolução com base num esquema de partição e geração de colunas.

5.1 Definição do Problema

No problema em estudo o objectivo é agendar um conjunto de tarefas num grupo de processadores paralelos idênticos, com o objectivo de minimizar uma função objectivo dependente do trabalho atrasado e da quantidade de preparações. As tarefas podem ser agendadas segundo um esquema de multiprocessamento de perfil variável, isto é, a qualquer momento é possível ter um número arbitrário de processadores dedicados a uma tarefa e a qualquer momento é possível reorientar processadores para outras tarefas. A figura seguinte esquematiza esta situação:

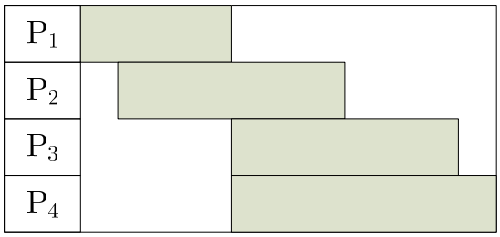


Figura 5.1 – Multiprocessamento de perfil variável

Na figura está representado o agendamento num sistema de 4 processadores de uma determinada tarefa representada a sombreado. Repare-se que a tarefa começa por ser agendada numa só máquina. Depois o número de máquinas dedicadas à tarefa vai variando ao longo da agenda.

5.1.1 Processadores

Considere-se então um conjunto de m processadores idênticos designado por P, tal que, $P = \{P_1, P_2, \cdots, P_m\}$. Os processadores só podem processar uma tarefa de cada vez mas podem ser interrompidos em instantes arbitrários. Por uma questão de redução da complexidade computacional, vamos admitir que as interrupções devem ser efectuadas em instantes inteiros. Isto não implica perda de generalidade uma vez que a unidade pode ser definida livremente considerando o nível de detalhe requerido pela aplicação concreta e a capacidade computacional disponível.

5.1.2 Tarefas

Considere-se agora o conjunto das n tarefas a processar, designado por T, tal que $T = \{T_1, T_2, \cdots, T_n\}$. Associado a cada tarefa existe um conjunto de parâmetros que definem os requisitos a ela associados:

- Tempo de processamento, p_j, que define a quantidade de processamento de que determinada tarefa necessita;
- Data de disponibilidade, r_j, que indica o instante de tempo a partir do qual a tarefa passa a estar disponível para processamento;
- Data de conclusão, d_j, que representa o instante em que a tarefa deve estar terminada. O processamento posterior é possível mas incorre-se numa penalização por atraso. Faz todo o sentido que $d_j > r_j$;
- Penalização por atraso, w_j, que define a penalização associada ao processamento da tarefa para além do instante d_j. Os valores absolutos dos vários w_j são em larga medida arbitrários, interessando mais a relação entre eles.

Embora os valores de p_j, r_j e d_j possam ser quaisquer valores positivos, neste problema vão ser limitados a valores inteiros para que o sistema seja coerente com as unidades de tempo definidas para os processadores. Tal como já se afirmou, isto não representa grande perda de generalidade.

Devido a esta última definição, e como as tarefas são perfeitamente divisíveis, podemos olhar para uma agenda como uma sequência de unidades agendadas. Se duas unidades consecutivas pertencerem à mesma tarefa, não se incorre em qualquer preparação. Se, pelo contrário, em duas unidades de tempo consecutivas de um processador estiverem agendadas duas tarefas diferentes, incorre-se numa preparação.

5.1.3 Função Objectivo

Pretende-se encontrar soluções que minimizem uma função do trabalho atrasado e do número de preparações. A utilização de uma função objectivo padrão (como as da secção 2.2.4) coloca alguns problemas porque para as calcular é necessário conhecer a totalidade das agendas dos vários processadores. Como no esquema de resolução é necessário calcular a contribuição da agenda de cada processador para o custo global do agendamento (a função objectivo deve ser aditiva), a utilização de muitas daquelas funções é inviável. Existe ainda a complicação adicional relacionada com a contabilização do número de preparações.

Assim, definiu-se a função objectivo como a soma das contribuições dos vários processadores para o custo global da solução de agendamento. Seja c_k a contribuição que uma agenda de uma máquina para a função de custo global. Defina-se então c_k:

$$c_k = L'_k + P_k$$

Em que L'_k corresponde ao custo pelos atrasos incorridos e P_k corresponde ao custo pela quantidade de preparações efectuadas.

Para definir P_k é necessário definir uma constante p que represente o custo de uma preparação e que deve ser conjugada com os valores dos vários w_j para que o algoritmo de resolução seleccione correctamente as melhores agendas. Esta constante p é multiplicada pelo número de preparações em que se incorre, N^P, para calcular P_k, tal que:

$$P_k = pN^P$$

Para calcular L'_k é necessário precisar como se calcula o custo do trabalho atrasado. Neste caso, optou-se por uma função onde os custos do atraso aumentam de forma linear com o afastamento da data de conclusão. Matematicamente fica:

$$L'_k = \sum_{j=1}^{n} \sum_{l \in L'_j} \left(C_l - d_j \right) w_j$$

Nesta equação, L'_j é o conjunto de unidades da tarefa T_j que estão agendadas para lá do instante d_j. O valor C_l é o instante de conclusão do processamento de uma unidade atrasada. Com este esquema pretende-se que o trabalho atrasado seja agendado antes de iniciar o agendamento de novas tarefas não atrasadas. Note-se que se a penalização pelo trabalho atrasado fosse constante, independentemente do período onde a unidade atrasada fosse agendada, o trabalho atrasado tenderia a ser todo agendado no final do agenda, para que o número de unidades a agendar com atraso fosse mínimo (assumindo pesos idênticos para as tarefas). Num ambiente industrial, este efeito teria muitos inconvenientes, tais como atrasos inaceitáveis na conclusão das tarefas e uma grande quantidade de trabalho em curso.

Considere-se o seguinte exemplo de cálculo das penalizações associadas a trabalho atrasado:

Exemplo 5.1

Considere-se a seguinte agenda de uma máquina onde são agendadas as tarefas T_1 e T_2:

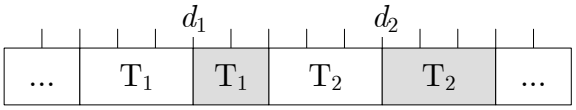


Figura 5.2 – Cálculo do custo do trabalho atrasado

Nesta agenda a tarefa T_1 tem 2 unidades agendadas com atraso. A primeira dessas unidades está agendada logo depois de d_1, finalizando em $d_1 + 1$, implicando uma penalização igual a w_1. Como a segunda das unidades finaliza em $d_1 + 2$, a penalização associada é de $2w_1$. Logo a penalização associada aos atrasos da tarefa T_1 e de $3w_1$. Para a tarefa T_2 a situação repete-se e existe ainda mais uma unidade que finaliza em $d_2 + 3$ a que está associada uma penalização de $3w_2$, elevando a penalização total devido à tarefa T_2 para $6w_2$. A penalização total por ambas as tarefas será então de $3w_1 + 6w_2$. ■

5.1.4 Classificação do Problema

Utilizando a notação de Graham, apresentada na secção 2.3 este problema pode ser referenciado como

$$P\left|r_j, spdp\text{ - }lin, var\right|X\,,$$

que designa um problema de máquinas paralelas idênticas com datas de disponibilidade arbitrárias e com multiprocessamento, de modo a minimizar uma função objectivo não padrão.

5.2 Modelo de Fluxo de Custo Mínimo

O problema descrito na secção anterior pode ser modelado numa rede e resolvido através de um algoritmo para problemas de fluxo de custo mínimo [1]. Embora este modelo forneça sempre soluções válidas ele é insensível aos custos de preparação, não sendo possível obter a solução óptima para a função objectivo desenvolvida em 5.1.3. No entanto, as soluções fornecidas podem ser melhoradas por heurísticas e podem servir como limites superiores em algoritmos de pesquisa.

5.2.1 Descrição do Modelo

Para uma mais fácil compreensão, o modelo pode ser visto como um problema de transportes [1]. Como origens do problema temos as n tarefas a agendar em que há uma oferta igual a p_j. Designem-se estas origens por T_j. Como destinos temos as unidades de tempo do agendamento. Cada destino, designado por I_t (t poderá variar entre 1 e t_{max}, sendo este último definido como o instante limite do horizonte de planeamento) corresponde a agendar no intervalo de tempo $[t-1,t]$. A procura em cada destino é igual ao número de máquinas disponíveis, ou seja, m. Para equilibrar a oferta com a procura é necessário inserir mais uma oferta, T_L, correspondente ao tempo livre que se deve distribuir pelas várias máquinas. O valor da oferta deverá ser igual a

$$mt_{max} - \sum_{j=1}^{n} p_j$$

Se o valor calculado para esta oferta for negativo significa que o agendamento é impossível, pois a quantidade de trabalho a agendar excede a capacidade dos processadores no horizonte de planeamento considerado.

A rede que se apresenta na figura seguinte ilustra esta formulação:

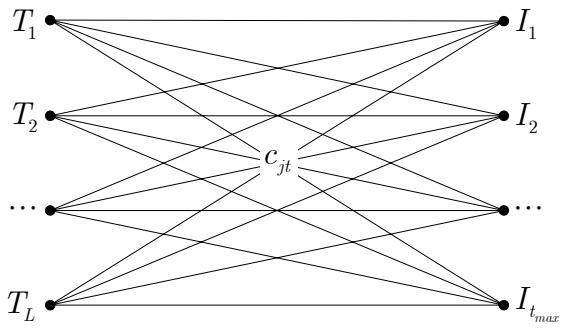


Figura 5.3 – Formulação de fluxo de custo mínimo

Da figura anterior há a notar que o arco $\left(T_j, I_t\right)$ apenas existe se a tarefa T_j estiver disponível para processamento no instante $t-1$, ou seja, se $r_j \leq t-1$. Quanto aos custos, c_{jt}, devem ser calculados da seguinte forma:

$$c_{jt} = \begin{cases} 0 & se\ d_j \geq t \\ \left(t - d_j\right) w_j & se\ d_j < t \end{cases}$$

Este cálculo está de acordo com o que foi definido em 5.1.3 para a função objectivo, pelo que esta formulação minimiza a parte dos custos relativos a trabalho atrasado. Quanto aos custos devidos às preparações, tal como já se afirmou, o modelo é totalmente insensível.

Este modelo é semelhante ao modelo de Horn [35], no entanto, aquele não permitia o multiprocessamento (aplicava-se a problemas do tipo $P\left|pmtn, r_j, \tilde{d}_j\right|$). Os modelos têm apenas em comum a transformação de um problema de agendamento num problema de fluxo de custo mínimo.

Há no entanto um caso onde o modelo agora proposto serve para encontrar a solução óptima:

Proposição 5.1

Se os tempos de processamento forem unitários, isto é, $p_j = 1$ para todo o j, o modelo de fluxo de custo mínimo apresentado fornece a solução óptima.

Prova

Se os tempos de processamento forem unitários, todas as tarefas serão agendadas apenas uma única vez, incorrendo numa preparação. Qualquer que seja a solução de agendamento o número de preparações será constante e igual a ao número de tarefas do problema considerado. ■

Na próxima secção apresenta-se um algoritmo para transformar a solução do problema de fluxo de custo mínimo numa solução de agendamento válida.

5.2.2 Obtenção de uma solução

Uma vez resolvido o problema de fluxo de custo mínimo apresentado na secção anterior, é necessário transformar os fluxos obtidos numa solução de agendamento. Embora a passagem seja trivial, nesta secção é proposto um método que, para um dado conjunto de fluxos, minimiza o número de preparações.

Designe-se então o fluxo no arco $\left(T_j, I_t\right)$ por S_{jt}. Uma solução fica então completamente definida pela matriz de fluxos resultantes da resolução. Designe-se essa matriz por S, tal que:

$$S = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1t_{max}} \\ S_{21} & S_{22} & \cdots & S_{2t_{max}} \\ \cdots & \cdots & \cdots & \cdots \\ S_{n1} & S_{n2} & \cdots & S_{nt_{max}} \end{bmatrix}$$

Como os valores das ofertas e das procuras que foram introduzidas no problema de transportes são todos inteiros, é garantido que a matriz S apenas contém valores inteiros. Da formulação do problema resulta que $\sum_{j=1}^{n} S_{jt} \leq m$

para todo o t. Logo, na primeira unidade de tempo vão ser agendados S_{j1} processadores com cada uma das n tarefas, sendo necessário incorrer numa preparação em todas as máquinas agendadas.

Na segunda unidade de tempo não será necessário efectuar preparação se a tarefa a agendar no processador for igual à tarefa agendada no período anterior. Para cada tarefa poder-se-ão "poupar" $\min\left(S_{j1}, S_{j2}\right)$ preparações. Em geral, no instante t podem ser evitadas, no máximo, $\sum_{j=1}^{n} \min\left(S_{jt}, S_{jt+1}\right)$ preparações. O algoritmo que se propõe consegue efectuar o agendamento evitando em todos os períodos o maior número possível de preparações.

O algoritmo consiste em iniciar o agendamento dos fluxos da matriz S por ordem cronológica, ou seja, começando pelos fluxos relativos a $t = 1$ e prosseguindo até $t = t_{max}$. Em cada período de tempo o vector S_t é decomposto em dois vectores, S_t^C e S_t^U que se calculam da seguinte forma:

$$S_{jt}^C = \min\left(S_{jt-1}, S_{jt}\right)$$

$$S_{jt}^U = S_{jt} - S_{jt}^C$$

Como facilmente se conclui, o vector S_t^C corresponde às quantidades que podem ser agendadas sem incorrer em preparações e o vector S_t^U comporta as restantes quantidades. Para se iniciar o agendamento deve admitir-se que $S_{j0} = 0$ para todo o j, implicando que o vector S_1^C tenha apenas elementos nulos, o que corresponde a dizer que, no período $t = 1$ se incorre numa preparação em todas as máquinas agendadas.

Para fazer o agendamento, em primeiro lugar distribuem-se as quantidades S_{jt}^C por máquinas onde estejam agendadas as tarefas j no período anterior. Seguidamente agendam-se as quantidades S_{jt}^U de forma arbitrária nas máquinas onde ainda nada esteja agendado. Eis o pseudo código:

```
begin
    for t:=1 to t_max do
    begin
        for j:=1 to n do
        begin
            S^C_jt =min( S_jt-1 , S_jt );
```

```
            S^U_jt = S_jt - S^C_jt ;
        end;
        for j:=1 to n do
        begin
            while S^C_jt >0 do
            begin
                for i:=1 to m do
                begin
                    if agenda(i, t-1)= j then
                    begin
                        agenda(i, t ):= j
                        S^C_jt := S^C_jt -1;
                        break;
                    end;
                end;
            end;
        end;
        for j:=1 to n do
        begin
            while S^U_jt >0 do
            begin
                for i:=1 to m do
                begin
                    if agenda(i, t )= ∅ then
                    begin
                        agenda(i, t ):= j
                        S^U_jt := S^U_jt -1;
                        break;
                    end;
                end;
            end;
        end;
    end;
end;
```

A complexidade computacional deste algoritmo é polinomial em relação ao tamanho da instância e é da ordem de $O\left(nm^2t_{max}\right)$, o que significa tempos de execução perfeitamente aceitáveis, mesmo para instâncias de grande dimensão.

Proposição 5.2

Da aplicação do algoritmo apresentado a um determinado conjunto de fluxos resulta um número mínimo de preparações.

Prova

No instante $t = 0$ é necessário efectuar preparações em todas as máquinas agendadas, sendo esse número igual a

$$\sum_{j=1}^{n} S_{j1} .$$

Nos instantes $t = 1,2,\cdots,t_{max} - 1$, para qualquer tarefa, não são necessárias preparações se $S_{jt} \geq S_{jt+1}$, uma vez que o algoritmo agenda S_{jt+1} máquinas sem efectuar preparações. Caso contrário, será necessário efectuar, no mínimo, $S_{jt+1} - S_{jt}$ preparações. Como o algoritmo consegue agendar S_{jt} máquinas sem efectuar preparações, apenas é efectuado o número mínimo de preparações para cada tarefa. ■

Exemplo 5.2

Considere-se agora um exemplo em que se dispõe de 3 máquinas e se pretende agendar 3 tarefas no intervalo de tempo $[0,5]$. Os dados relativos às tarefas estão na tabela que se segue:

j	p	r	d	w
1	5	0	4	10
2	5	0	4	10
3	5	1	4	10

Tabela 5.1 – Dados do Exemplo 5.2

Admita-se que o custo de uma preparação é $p = 1$. A rede do problema de fluxo de custo mínimo resultante destes dados é a seguinte:

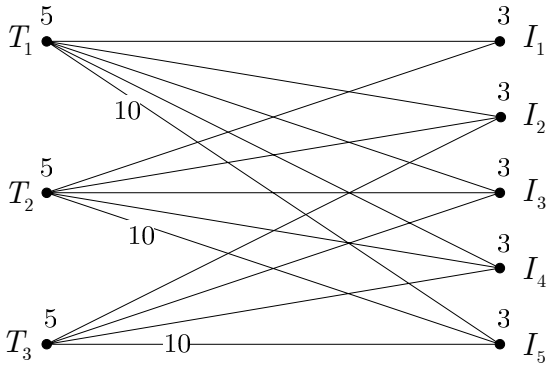


Figura 5.4 – Rede do Exemplo 5.2

Junto às origens é denotada a oferta nessa origem e junto aos destinos a respectiva procura. Todos os arcos são orientados da esquerda para a direita e os valores inscritos sobre eles referem-se aos custos de atravessamento definidos segundo as expressões anteriormente apresentadas. Quando o valor do custo não é explícito significa que é nulo.

Uma das soluções possíveis para este problema de fluxo de custo mínimo é a que resulta na seguinte matriz de fluxos:

$$S = \begin{bmatrix} 2 & 2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \end{bmatrix}$$

Aplique-se agora o algoritmo de agendamento, cuja solução será representada na Figura 5.5. No período 1, correspondente ao intervalo de tempo $[0,1]$, não há qualquer dificuldade, devendo duas máquinas ser agendadas com a tarefa 1 e uma máquina com a tarefa 2. Designem-se por 1 e 2 as máquinas agendadas com a tarefa 1 e por 3 a máquina agendada com a tarefa 2.

No segundo período de agendamento é necessário agendar duas máquinas com a tarefa 1 e 1 máquina com a tarefa 3. Neste caso, as duas unidades da tarefa 1 seriam as primeiras a ser agendadas nas máquinas que estão preparadas para aquela tarefa. Utilizando a simbologia do algoritmo, temos $S_{12}^C = 2$, que seriam as primeiras duas unidades a ser agendadas.

Olhando agora para a terceira unidade de tempo, temos $S_{13}^C = 1$ e $S_{33}^C = 1$, o que significa que devemos agendar a unidade de tarefa 1 numa das máquinas

que está preparada para aquela tarefa e a unidade de tarefa 3 na máquina preparada para a tarefa 3.

Da aplicação sucessiva do algoritmo resulta a seguinte agenda:

	0				5
P_1	1	1	1	2	2
P_2	1	1	2	2	3
P_3	2	3	3	3	3

Figura 5.5 – Agendas finais

Na figura acima representa-se a sombreado o trabalho atrasado e assinalam-se as preparações com uma divisória mais carregada. O custo global desta solução é de 37 unidades. ■

Embora o número de preparações seja mínimo para os fluxos da matriz S dados, pode facilmente constatar-se que era possível obter uma agenda válida com menos preparações apenas procedendo a algumas trocas simples. Por exemplo, a solução apresentada na Figura 5.6 tem um custo de apenas 35 unidades.

	0				5
P_1	1	1	1	1	1
P_2	2	2	2	2	3
P_3	2	3	3	3	3

Figura 5.6 – Agenda alternativa

5.3 Modelo de Programação Inteira

Uma vez que o modelo de fluxo de custo mínimo introduzido na secção anterior não é capaz de fazer a optimização global do modelo, pois ignora os custos de preparação dos processadores, torna-se necessário encontrar um modelo

que entre em linha de conta com esses custos. O modelo de programação inteira que se apresenta nesta secção consegue precisamente esse objectivo.

5.3.1 Modelo

A formulação que se propõe é baseada em variáveis indexadas ao tempo. Este tipo de variáveis foram pela primeira vez utilizadas em problemas de planeamento de operações por Sousa e Wolsey [48]. No presente caso, uma variável binária deste tipo designa-se por x_{ijt} e toma o valor 1 se a tarefa j estiver agendada na máquina i na unidade de tempo t, que corresponde ao intervalo de tempo $[t-1,t]$.

A primeiras restrições do modelo são as seguintes:

$$\sum_i \sum_t x_{ijt} = p_j \qquad ,\forall j$$

Este conjunto de restrições garante que todas as tarefas são agendadas na totalidade, enquanto que o conjunto de restrições que se segue garante que em cada máquina e em cada unidade de tempo apenas é agendada uma tarefa:

$$\sum_j x_{ijt} \leq 1 \qquad ,\forall i,\forall t$$

Note-se que, como i varia entre 1 e m, este conjunto de restrições também garante que, em cada unidade de tempo, não são utilizadas mais do que m máquinas.

Para tratar a questão das preparações, é necessário introduzir um novo tipo de variável binária que designamos por y_{ijt}. Pretende-se que, sempre que o facto de fazer $x_{ijt} = 1$ constitua uma preparação, a correspondente variável, y_{ijt}, tome o valor de 1, sendo nula nos restantes casos. Para que isto aconteça, é necessário impor os seguintes conjuntos de restrições:

$$y_{ij1} = x_{ij1} \qquad ,\forall i,\forall j$$

que garante que é contabilizada uma preparação sempre que, para $t = 1$, seja agendada a tarefa j na máquina i. Para os restantes períodos de tempo,

$$y_{ijt} \geq x_{ijt} - x_{ijt-1} \qquad ,\forall i, \forall j, t = 2, 3, \cdots, T_{max}\,,$$

que garante que é contabilizada uma preparação sempre que houver o agendamento de uma nova tarefa numa máquina. Na tabela que se segue apresentam-se as várias combinações de valores possíveis para x_{ijt} e x_{ijt-1}, bem como as restrições resultantes:

x_{ijt-1}	x_{ijt}	Restrição
0	0	$y_{ijt} \geq 0$
0	1	$y_{ijt} \geq 1$
1	0	$y_{ijt} \geq -1$ (redundante)
1	1	$y_{ijt} \geq 0$

Tabela 5.2 - Cálculo das variáveis y_{ijt}

Como se pode verificar, na única situação em que há uma preparação, a variável y_{ijt} correspondente é obrigada a assumir o valor 1.

Com base nestas variáveis e nos seus significados, apresenta-se agora a função objectivo:

$$min \sum_i \sum_j \sum_t p y_{ijt} + \sum_i \sum_j \sum_{t=d_j+1}^{T_{max}} w_j \left(t - d_j\right) x_{ijt}$$

A primeira parcela da soma representa o custo das operações de preparação, enquanto que a segunda parcela representa os custos relacionados com o trabalho atrasado.

Sintetiza-se agora o modelo completo:

$$
\begin{aligned}
min \quad & \sum_i \sum_j \sum_t p y_{ijt} + \sum_i \sum_j \sum_{t=d_j+1}^{T_{max}} w_j \left(t - d_j\right) x_{ijt} \\
s.t. \quad & \sum_i \sum_t x_{ijt} = p_j \qquad ,\forall j \\
& \sum_j x_{ijt} \leq 1 \qquad ,\forall i, \forall t \\
& y_{ij1} = x_{ij1} \qquad ,\forall i, \forall j \\
& y_{ijt} \geq x_{ijt} - x_{ijt-1} \qquad ,\forall i, \forall j, t = 2, 3, \cdots, T_{max} \\
& x_{ijt} = 0{,}1 \qquad ,\forall i, \forall j, \forall t \\
& y_{ijt} = 0{,}1 \qquad ,\forall i, \forall j, \forall t
\end{aligned}
$$

Embora este modelo tenha toda a validade, pela sua formulação baseada em variáveis binárias, pelo facto de se tratar de um problema com muita simetria (soluções diferentes da formulação correspondem à mesma solução física), a sua resolução através de um *package* comercial apenas é possível para instâncias muito pequenas. Por exemplo, para um problema de 10 máquinas, existem 10! = 3,628,800 soluções óptimas simétricas. Por outro lado, adicionando 10 tarefas e 10 períodos de planeamento, haveria 2,000 variáveis e 1,110 restrições.

5.3.2 Decomposição de Dantzig-Wolfe

Proceda-se agora à decomposição deste modelo pela técnica de Dantzig-Wolfe, deixando o primeiro conjunto de restrições no problema principal. As restantes restrições, depois de reorganizadas, formam uma estrutura angular em blocos, em que cada bloco corresponde às restrições que afectam uma única máquina. Reescrevendo o modelo sob a forma vectorial, de modo a evidenciar esta estrutura, fica:

$$
\begin{aligned}
min \quad & \sum_i \left[p, c_i^x\right] \cdot \left[y_i, x_i\right] \\
s.t. \quad & \sum_i A_i^j x_i = p_j \qquad ,\forall j \\
& \left[y_i, x_i\right] \in P_i \qquad ,\forall i
\end{aligned}
$$

Em que $c_i^x = w_j\left(t - d_j\right)$ para $t \geq d_j + 1$ e $c_i^x = 0$ nos restantes casos, e em que os elementos de A_i^j tomam o valor 1 nas posições que afectam a restrição p_j e 0 nos restantes casos.

Como se pode ver, a função objectivo apenas foi arranjada, enquanto que os últimos cinco conjuntos de restrições foram substituídos por m restrições, onde se garante que o vector $\left[y_i, x_i\right]$ pertence ao poliedro P_i, definido pelas restrições associadas às variáveis respeitantes à máquina i que foram removidas. Cada um destes m blocos de restrições dará origem a um subproblema.

O poliedro P_i, é um conjunto convexo, podendo qualquer ponto ser representado por uma combinação convexa de pontos extremos, P_k^i, tal que:

$$\left[y_i, x_i\right] = \sum_k \lambda_k^i P_k^i,\ \sum_k \lambda_k^i = 1,\ \lambda_k^i \geq 0,\ \forall k$$

Aplicando a decomposição de Dantzig-Wolfe ao problema ficamos com o seguinte problema principal:

$$\begin{aligned} min \quad & \sum_i \sum_k \left[p, c_i^x\right] \cdot P_k^i \cdot \lambda_k^i \\ s.t. \quad & \sum_i \sum_k A_i^j P_k^{ix} \lambda_k^i = p_j \qquad ,\forall j \\ & \sum_k \lambda_k^i = 1 \qquad ,\forall i \end{aligned}$$

Para simplificar fez-se $P_k^i = \left[P_k^{iy}, P_k^{ix}\right]$.

Como os processadores são idênticos, todos os subproblemas são iguais, sendo o conjunto de pontos extremos comum. Logo, as referências ao processador podem ser retiradas da formulação e as restrições de convexidade podem ser substituídas pela sua soma, ficando:

$$\begin{aligned} min \quad & \sum_k \left[p, c^x\right] \cdot P_k \cdot \lambda_k \\ s.t. \quad & \sum_k A^j P_k^x \lambda_k = p_j \qquad ,\forall j \\ & \sum_k \lambda_k = m \end{aligned}$$

Por uma questão de simplicidade, efectuem-se também as substituições de símbolos $\lambda_k = x_k$, $\left\{p, c_{jt}^x\right\} \cdot P_k = c_k$ e $A^j P_k^x = a_{jk}$.

O modelo final fica:

$$\begin{aligned} min \quad & \sum_k c_k x_k \\ s.t. \quad & \sum_k a_{jk} x_k = p_j, \quad j = 1,2,\cdots,n \\ & \sum_k x_k = m \end{aligned}$$

Neste modelo, cada variável pode ser entendida como a agenda de uma máquina, sendo c_k a contribuição dessa agenda, em preparações e em atrasos, para o custo global da solução de agendamento e representando a_{jk} a quantidade de tarefa j que está incluída na agenda x_k. Com base nesta interpretação, que deriva da decomposição efectuada, pode retirar-se a seguinte conclusão:

Proposição 5.3

É condição suficiente para que uma solução cumpra os requisitos de integralidade que as variáveis x_k serem inteiras.

Prova

Como o valor de qualquer variável corresponde ao número de máquinas que executam uma determinada agenda, se estes valores forem inteiros, a solução pode ser traduzida num plano de operações válido. ■

Salienta-se, no entanto, que esta condição não é necessária, porque, em determinadas condições, pode ser possível obter um plano de operações válido a partir de uma solução fraccionária. Este assunto é tratado em detalhe na secção 5.5.

É sabido que a substituição de igualdades por desigualdades na formulação de problemas de programação linear traz vantagens computacionais, devido ao facto de se obter uma maior restrição do espaço dual de soluções. Nas duas proposições que se seguem vai provar-se que as restrições de igualdade que aparecem na formulação podem ser substituídas por restrições de desigualdade.

Proposição 5.4

Na formulação resultante, as restrições do tipo $\sum_k a_{jk}x_k = p_j$ podem ser substituídas por restrições do tipo $\sum_k a_{jk}x_k \geq p_j$.

Prova

Ao fazer esta substituição pode acontecer, para um conjunto de tarefas, o agendamento de uma quantidade superior aos respectivos p_j. Tal facto não pode contribuir para um aumento do valor do óptimo, pois caso contrário, o algoritmo encontraria uma solução de menor custo sem aquelas quantidades adicionais. As quantidades em excesso podem ser arbitrariamente removidas do final da agenda, pois remover quantidades agendadas nunca contribui para o aumento do número de preparações ou para um aumento do trabalho atrasado. ■

Proposição 5.5

A restrição $\sum_k x_k = m$ pode ser substituída por $\sum_k x_k \leq m$.

Prova

Se as quantidades a agendar envolvidas o permitirem, não é obrigatório agendar todas as máquinas, podendo apenas ser agendadas as máquinas necessárias. De qualquer maneira, as máquinas em excesso seriam sempre agendadas com agendas nulas. ■

Podemos então formular o modelo que vai ser utilizado da seguinte forma:

$$\begin{aligned} min \quad & \sum_k c_k x_k \\ s.t. \quad & \sum_k a_{jk}x_k \geq p_j, \quad j = 1,2,\cdots,n \\ & \sum_k x_k \leq m \\ & x_k \geq 0 \text{ e } inteiro, \quad \forall k \end{aligned}$$

À luz desta formulação, torna-se agora mais claro porque foi necessário encontrar uma função objectivo onde a contribuição de uma agenda de um processador para essa função pudesse ser calculada independentemente de todas as outras agendas (função objectivo aditiva). A maioria das funções objectivo padrão, como L_{max} ou C_{max}, não têm esta propriedade.

Como facilmente se depreende, o conjunto de todas as agendas possíveis é exponencialmente grande. Para um pequeno problema como o do Exemplo 5.2, com 3 tarefas e um horizonte de agendamento de 5 unidades, o número de agendas possíveis é $4^5 = 1024$ (admitindo que também pode ser introduzido tempo livre na agenda). Com as mesmas 3 tarefas e um horizonte de agendamento de 10 unidades de tempo já seriam possíveis 1,048,576 agendas diferentes! Esta constatação implica a impraticabilidade de enumerar à partida todas as agendas possíveis (pontos extremos do poliedro do subproblema) e resolver o problema, pelo que o problema terá que ser resolvido por um método que não necessite desta enumeração, tal como o método de geração de colunas anteriormente apresentado.

Torna-se então necessário criar um subproblema onde sejam identificadas as variáveis que em determinado momento sejam mais atractivas no problema principal. A apresentação deste subproblema será o objecto da próxima secção.

Para iniciar o processo é de toda a conveniência ter um conjunto de colunas de onde possa ser retirada uma solução inteira válida. Embora em termos matemáticos o processo pudesse ser iniciado substituindo a matriz de coeficientes tecnológicos por uma matriz identidade e substituindo os vector da função objectivo por um vector de valores arbitrariamente grandes, a existência de uma solução inicial válida fornece desde logo um limite superior para o valor da solução.

Como a formulação de fluxo de custo mínimo apresentada na secção anterior fornece sempre (desde que o problema não seja impossível) uma solução de agendamento válida, essa solução pode ser utilizada para calcular as colunas a colocar inicialmente no problema primal restrito.

A operação de transformação da solução de agendamento num conjunto de coeficientes para introduzir no problema primal restrito resume-se a simples operações de contagem do trabalho agendado e dos custos em que se incorre.

Exemplo 5.2 (continuação)

Recupere-se a solução de agendamento que foi calculada anteriormente:

	0				5
P_1	1	1	1	2	2
P_2	1	1	2	2	3
P_3	2	3	3	3	3

Figura 5.7 – Agenda final do Exemplo 5.2

A transformação desta agenda no quadro inicial da formulação de programação inteira que se propõe não passa de uma operação de contagem:

	x_1	x_2	x_3	
T_1	3	2	0	$= 5$
T_2	2	2	1	$= 5$
T_3	0	1	4	$= 5$
M	1	1	1	$\leqslant 3$
min	12	13	12	

Figura 5.8 – Quadro resultante da agenda inicial

As variáveis x_1, x_2 e x_3 resultam, respectivamente, da transformação das agendas associadas a P_1, P_2 e P_3. Os valores de c_k são calculados pela soma das penalizações por trabalho atrasado com as penalizações pelo número de preparações. Os valores de a_{jk} resultam da contagem do trabalho pertencente a cada tarefa que determinada agenda contenha. Como se pode constatar, a operação é trivial. ■

A este problema primal restrito é necessário acrescentar agora as variáveis (colunas) que conduzam à solução óptima. Este é o objecto da próxima secção.

5.4 Geração de Colunas

Para gerar as novas colunas a incluir no problema primal restrito, é necessário desenhar um subproblema que, a partir da informação dual fornecida pelo problema primal restrito, identifique correctamente colunas atractivas que ainda não tenham sido introduzidas no problema. As colunas atractivas vão sendo introduzidas, uma a uma, no problema primal restrito, até que o subproblema já não consiga identificar mais colunas atractivas, caso em que estaremos perante a solução óptima do problema primal restrito.

Refira-se que nesta fase do processo, as condições de integralidade das variáveis do problema primal restrito devem ser relaxadas, sendo mais tarde introduzidas por um esquema de partição e avaliação. Deste modo, quando nesta fase se referem soluções óptimas, pretende-se significar soluções óptimas para a relaxação linear do problema.

5.4.1 Subproblema Resultante da Decomposição

Relembre-se a formulação inicial do problema principal, depois de simplificada:

$$\begin{aligned} min \quad & \sum_i [p, c_i^x] \cdot [y_i, x_i] \\ s.t. \quad & \sum_i A_i^j x_i = p_j \qquad ,\forall j \\ & [y_i, x_i] \in P_i \qquad ,\forall i \end{aligned}$$

Tal como foi referido na secção 3.2, o custo reduzido de uma coluna (indexada com l) é dado por $c_B B^{-1} A_l - c_l$. Sendo $c_B B^{-1}$ o vector dual, no caso concreto, este é igual a $\pi = [\pi_T, \pi_M]$, com $\pi_T = [\pi_1, \pi_2, \cdots, \pi_n]$, em que π_j é a variável dual correspondente à restrição associada a p_j e π_M a variável dual

associada à restrição de convexidade que limita o número de máquinas utilizadas.

Como uma nova coluna corresponde a um ponto P_k do conjunto de pontos extremos do poliedro P, os primeiros n elementos de A_l são iguais a $A^j x$ e o último elemento (correspondente à restrição de convexidade) é unitário. Logo, o custo reduzido é dado por:

$$c_B B^{-1} A_l - c_l = \pi_T A^j x + 1\pi_M - [p, c^x] \cdot [y, x]$$

Passando da notação vectorial para somatórios, fica:

$$\sum_j \sum_t \pi_j x_{jt} + \pi_M - \sum_j \sum_t p y_{jt} - \sum_j \sum_{t=d_j+1}^{T_{max}} w_j \left(t - d_j\right) x_{jt}$$

Logo, o subproblema a resolver será:

$$\begin{aligned}
max \quad & \sum_j \sum_t \pi_j x_{jt} + \pi_M - \sum_j \sum_t p y_{jt} - \sum_j \sum_{t=d_j+1}^{T_{max}} w_j \left(t - d_j\right) x_{jt} \\
s.t. \quad & \sum_j x_{jt} \leq 1 \qquad ,\forall t \\
& y_{j1} = x_{j1} \qquad ,\forall j \\
& y_{jt} \geq x_{jt} - x_{jt-1} \qquad ,\forall j, t = 2, 3, \cdots, T_{max} \\
& x_{jt} = 0,1 \qquad ,\forall j, \forall t \\
& y_{jt} = 0,1 \qquad ,\forall j, \forall t
\end{aligned}$$

Note-se que este subproblema, formulado desta forma, continua a ser computacionalmente muito exigente, mesmo para instâncias muito pequenas. No entanto, com base na interpretação que dele pode ser feita, é possível pensar em outras formas de o resolver.

Na função objectivo, a primeira parcela premeia em π_j cada unidade da tarefa j. A segunda parcela, π_M, é o preço a pagar pela utilização do recurso "máquina" e é constante, independentemente da agenda. A terceira parcela inclui na função objectivo as penalizações devidas pelas preparações. Por fim, é acrescentada uma penalização por cada unidade de tarefa agendada com atraso.

Raciocinando sobre esta interpretação, apresenta-se na próxima secção um modelo de programação dinâmica que pode ser utilizado para resolver instâncias do subproblema de grande dimensão.

5.4.2 Modelo de Programação Dinâmica

Para identificar novas colunas foi desenvolvido um algoritmo de programação dinâmica. O algoritmo utiliza $n+1$ estados que reflectem a preparação existente na máquina (não preparada ou preparada para uma das n tarefas) e t_{max} estágios, correspondentes ao agendamento de cada uma das unidades de tempo que compõem o horizonte de agendamento mais um estágio inicial.

A partir das datas de disponibilidade das tarefas é elaborada uma rede de transição entre estados de forma a que seja impossível colocar a máquina num estado correspondente a uma tarefa que ainda não esteja disponível naquele estágio (intervalo de tempo). Os custos de transição de estado são determinados pelos prémios ou penalidades de agendar determinada tarefa (obtidos da informação dual) e pelos custos de preparação e do trabalho atrasado.

Seja $\pi = \left[\pi_1, \pi_2, \cdots, \pi_n, \pi_M\right]$ a solução dual do problema primal restrito, em que os primeiros n coeficientes correspondem à restrição relativa a cada uma das tarefas (restrições T_j) e o último coeficiente à restrição relativa ao número de máquinas (restrição M). Seja c_k o coeficiente da função objectivo da nova variável e $A_k = \left[a_{1k}, a_{2k}, \cdots, a_{jk}, 1\right]^T$ o respectivo vector na matriz tecnológica. A nova variável é atractiva se $\pi A_k - c_k > 0$.

Assim, a contribuição de uma transição de estado para a função objectivo do subproblema deve ser o prémio dual associado à tarefa que se está a agendar (π_j) subtraído de p no caso de se incorrer numa preparação e subtraído de $w_j\left(t - d_j\right)$ no caso de agendamento para lá de d_j. De facto, ao agendar uma unidade de determinada tarefa, a única alteração em A_k é o incremento do coeficiente a_{jk} respectivo em 1 unidade. Se, simultaneamente, se incorrer numa preparação ou no agendamento atrasado, c_k é incrementado em p e

$w_j\left(t-d_j\right)$, respectivamente. Agendar uma unidade de tempo sem qualquer tarefa não produz qualquer alteração na função objectivo do subproblema.

Considere-se agora a rede de transição entre estados:

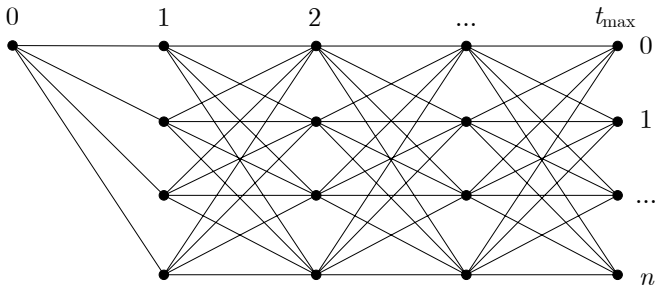


Figura 5.9 - Rede de transição de estado

Denote-se por $F_t(j)$ o valor da função objectivo no estado j do estágio t. Este estado só está disponível e pode ser alcançado se $r_j < t$, ou seja, se a tarefa j já estiver disponível para agendamento no período $[t-1,t]$. A solução óptima do problema de programação dinâmica pode ser calculada aplicando sucessivamente, do estado 0 para o estado t_{max} as seguintes equações de recorrência:

$$F_0(0) = \pi_M$$

$$F_t(j) = \max\left(F_{t-1}(l) + \pi_j^a - w_j^t - p_{lj}^a\right)$$

com $t = 1,2,\cdots,t_{max}$, com $j = 0,\left\{j : r_j < t\right\}$ e com $l = 0,\left\{j : r_j < t-1\right\}$.

Os valores de π_j^a, w_j^t e p_{lj}^a são as contribuições da passagem de estágio e o seu cálculo já foi explicado acima. Sistematizam-se agora as expressões para o cálculo:

$$\pi_j^a = \begin{cases} 0, & j = 0 \\ \pi_j, & j > 0 \end{cases}$$

$$w_j^t = \begin{cases} 0, & j = 0 \vee t \le d_j \\ w_j\left(t-d_j\right), & j > 0 \wedge t > d_j \end{cases}$$

$$p_{lj}^{a} = \begin{cases} 0, & j = 0 \vee l = j \\ p, & j > 0 \wedge l \neq j \end{cases}$$

O processo de geração decorre como está descrito na Figura 3.2 (página 26) procedendo-se à resolução alternada do problema primal restrito e do subproblema, até que o subproblema não consiga identificar mais colunas atractivas, momento em que foi encontrada a solução óptima da relaxação linear.

Exemplo 5.2 (continuação)

Retome-se o exemplo que tem vindo a ser tratado ao longo do capítulo. A resolução do quadro inicial da relaxação linear fornece uma solução primal $x = [1,1,1]$ e uma solução dual $\pi = [0.8, 4.8, 1.8, 0]$, sendo o valor da função objectivo igual a 37.

Aplicando agora as equações de recorrência obtém-se:

$F_0(0) = 0$
$F_1(0) = 0\,,\ F_1(1) = -0.2\,,\ F_1(2) = 3.8$ e $F_1(3) = 0.8$
$F_2(0) = 3.8\,,\ F_2(1) = 3.6\,,\ F_2(2) = 8.6$ e $F_2(3) = 4.6$
$F_3(0) = 8.6\,,\ F_3(1) = 8.4\,,\ F_3(2) = 13.4$ e $F_3(3) = 9.4$
$F_4(0) = 13.4\,,\ F_4(1) = 13.2\,,\ F_4(2) = 18.2$ e $F_4(3) = 14.2$
$F_5(0) = 18.2\,,\ F_4(1) = 8.0\,,\ F_5(2) = 13.0$ e $F_4(3) = 9.0$

O óptimo do subproblema tem um valor de 18.2 e corresponde à agenda 2-2-2-2-0. Associe-se a esta agenda uma variável x_4 em que $A_4 = [0,4,0,1]$ e $c_4 = 1$ (apenas uma preparação). Como se pode constatar, $\pi A_4 - c_4 = 18.2$ que é um valor positivo, sendo a coluna atractiva. Introduza-se esta nova coluna no problema principal e proceda-se a nova optimização:

	x_1	x_2	x_3	x_4		π
T_1	3	2	0	0	$= 5$	7
T_2	2	2	1	4	$= 5$	11
T_3	0	1	4	0	$= 5$	8
M	1	1	1	1	$\leqslant 3$	-31
min	12	13	12	1		
x	1	1	1	0		37

Figura 5.10 - Quadro após introdução de x_4

Como se pode ver no quadro, embora a solução primal não se altere, a solução dual sofreu alterações, devendo ser novamente resolvido o subproblema com os novos dados. Essa resolução vai resultar na solução 2-2-2-2-2 e na variável x_5 em que $\pi A_5 - c_5 = 13$, resultando num novo quadro óptimo:

	x_1	x_2	x_3	x_4	x_5		π
T_1	3	2	0	0	0	$= 5$	10.[3]
T_2	2	2	1	4	5	$= 5$	10
T_3	0	1	4	0	0	$= 5$	10.25
M	1	1	1	1	1	$\leqslant 3$	-39
min	12	13	12	1	11		
x	1.6[7]	0	1.25	0	0.08[3]		35.92

Figura 5.11 - Quadro após introdução de x_5

Desta vez o subproblema gera a agenda 1-1-1-1-1 que ao ser introduzida no problema principal como a variável x_6 resulta no seguinte quadro:

	x_1	x_2	x_3	x_4	x_5	x_6		π
T_1	3	2	0	0	0	5	$= 5$	10
T_2	2	2	1	4	5	0	$= 5$	10
T_3	0	1	4	0	0	0	$= 5$	10.25
M	1	1	1	1	1	1	$\leqslant 3$	-39
min	12	13	12	1	11	11		
x	0	0	1.25	0	0.75	1		34.25

Figura 5.12 – Quadro após introdução de x_6

A partir desta solução dual não é possível obter mais nenhuma coluna atractiva, pelo que estamos perante a solução óptima. Note-se que não é possível gerar a agenda 3-3-3-3-3 porque a tarefa 3 só está disponível a partir de $t = 1$. ■

Este algoritmo de programação dinâmica tem t_{max} estágios e $n+1$ estados por estágio. Em cada estágio é necessário analisar um máximo de $(n+1)^2$ transições de estado, pelo que a complexidade computacional é $O\left((n+1)^2 t_{max}\right)$.

Designe-se agora o óptimo da relaxação linear por z_{RL}. Como todos os coeficientes da função objectivo são inteiros e as variáveis também devem ser todas inteiras, é fácil de concluir que o valor óptimo da função objectivo vai ser um inteiro. Podemos também concluir que o valor do óptimo do problema será igual ou superior ao inteiro imediatamente superior a z_{RL}, isto é, $z_{OPT} \geq \lceil z_{RL} \rceil$. Temos assim calculado um limite inferior para a solução óptima.

Este limite inferior pode ser incorporado no problema primal restrito sob a forma da seguinte restrição:

$$\sum_{\forall k} c_k x_k \geq \lceil z_{RL} \rceil$$

Este limite inferior será tratado em mais detalhe na secção 6.2.1 e, para já, não será introduzido no problema primal restrito.

No final desta fase da resolução dispõe-se de uma solução válida que serve de limite superior, que é a solução inicial retirada do problema de fluxo de custo mínimo, e o valor do limite inferior da solução óptima. No entanto, a solução resultante da relaxação linear, por norma, não cumpre os requisitos de integralidade das variáveis. É pois necessário aplicar um esquema de partição e avaliação que se expõe na secção seguinte.

5.5 Esquema de Partição e Avaliação

Para encontrar a solução óptima do problema que tem vindo a ser tratado é então necessário utilizar um esquema de partição e avaliação para alterar a relaxação linear do problema, de modo a que a solução passe a cumprir os requisitos de integralidade das variáveis.

É um facto conhecido que a utilização das variáveis do problema primal restrito no esquema de partição conduz geralmente a fenómenos de regeneração de variáveis [5]. Por exemplo, se para uma variável qualquer x_k cujo valor fosse igual a 0.5 fosse imposta uma restrição de partição do tipo $x_k \leq 0$, o que sucederia na próxima vez que fosse chamado o subproblema era a criação de uma variável exactamente igual à que foi colocada a zero, e o processo de resolução entraria num ciclo.

Este facto leva a que seja necessário encontrar uma formulação alternativa para o problema e fazer a partição sobre as variáveis dessa formulação. No caso deste problema essa formulação já existe, sendo a formulação como modelo de fluxo de custo mínimo. Recorde-se a rede do problema de fluxo de custo mínimo apresentado em 5.2:

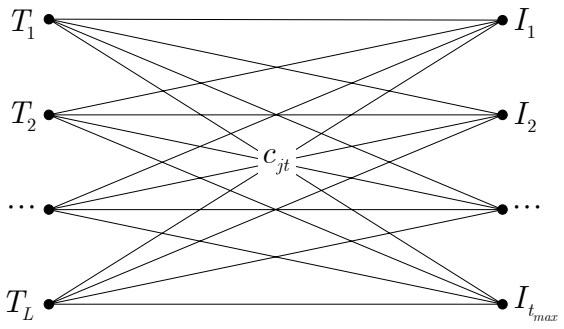


Figura 5.13 - Formulação de fluxo de custo mínimo

Já se descreveu a forma de transformar uma solução do problema de custo mínimo em agendas para as diversas máquinas. A operação inversa, ou seja, transformar um conjunto de agendas num conjunto de fluxos também não apresenta qualquer dificuldade, resumindo-se a somar os níveis das variáveis que contribuem para determinado fluxo.

Designe-se o fluxo no arco $\left(T_j, I_t\right)$ por S_{jt}. Defina-se ainda δ_k^{jt} da seguinte forma:

$$\delta_k^{jt} = \begin{cases} 0 & \textit{a tarefa } j \textit{ não está agendada em } [t-1,t] \\ 1 & \textit{caso contrário} \end{cases}$$

Desta forma podemos definir a correspondência entre as variáveis x (formulação de programação inteira) e as variáveis S (formulação de fluxo de custo mínimo):

$$S_{jt} = \sum_k \delta_k^{jt} x_k$$

Note-se que para calcular δ_k^{jt} a partir das variáveis x_k é necessário conhecer as agendas associadas às variáveis x_k, pelo que estas devem ser guardadas aquando da sua geração, pois não há uma relação unívoca entre os coeficientes tecnológicos das variáveis e as respectivas agendas.

O exemplo que se segue ilustra o cálculo de fluxos a partir de uma solução do problema primal restrito.

Exemplo 5.2 (continuação)

Retome-se a solução da relaxação linear do exemplo que tem vindo a ser tratado. As variáveis não nulas são $x_3 = 1.25$, $x_5 = 0.75$ e $x_6 = 1$. As respectivas agendas estão na figura que segue:

x_3	2	3	3	3	3
x_5	2	2	2	2	2
x_6	1	1	1	1	1

Figura 5.14 - Agendas das variáveis não nulas

Calculem-se a título de exemplo os fluxos para o primeiro período de tempo:

$$S_{11} = \delta_3^{11} x_3 + \delta_5^{11} x_5 + \delta_6^{11} x_6 = 0 + 0 + 1 = 1$$

$$S_{21} = \delta_3^{21} x_3 + \delta_5^{21} x_5 + \delta_6^{21} x_6 = 1.25 + 0.75 + 0 = 2$$

$$S_{31} = \delta_3^{31} x_3 + \delta_5^{31} x_5 + \delta_6^{31} x_6 = 0 + 0 + 0 = 0$$

Atente-se agora na tabela onde estão calculados todos os fluxos:

S_{jt}		t					S_j
		1	2	3	4	5	
j	1	1	1	1	1	1	5
	2	2	0.75	0.75	0.75	0.75	5
	3	0	1.25	1.25	1.25	1.25	5
S_t		3	3	3	3	3	15

Tabela 5.3 – Cálculo dos fluxos

Na tabela acima, além dos fluxos S_{jt} foram ainda calculados totais por período ($S_t = \sum_j S_{jt}$) e por tarefa ($S_j = \sum_t S_{jt}$) e o total global dos fluxos ($S = \sum_j \sum_t S_{jt}$). Como se pode ver, a integralidade deste totais, mesmo conjunta, não garante a integralidade da solução. No entanto, se houver validação computacional no sentido da sua utilidade na aceleração do processo de partição, também se podem impor restrições de partição sobre estes fluxos. ■

Atente-se agora na seguinte proposição:

Proposição 5.6

É condição necessária e suficiente para a obtenção de uma agenda válida que todos os fluxos S_{jt} de uma solução sejam inteiros.

Prova

É condição suficiente porque a partir de um conjunto de fluxos inteiros é sempre possível aplicar o algoritmo descrito em 5.2.2 para obter uma solução de agendamento válida. É condição necessária porque um determinado fluxo S_{jt} corresponde ao número de máquinas que estão agendadas com a tarefa j no período de tempo t, valor este que tem de ser necessariamente inteiro. ■

É por este motivo que não é condição necessária para a obtenção de uma solução válida que todas as variáveis do problema primal restrito tenham valores inteiros. É possível obter, a partir de uma solução fraccionária do problema primal restrito, um conjunto de fluxos inteiros a partir dos quais seja possível calcular um plano de operações válido. O exemplo que se segue ilustra esta situação.

Exemplo 5.3

Considere-se a solução fraccionária apresentada na Figura 5.15 com 4 variáveis e respectivas agendas:

x_1	1	1	2	2
x_2	2	2	2	2
x_3	1	1	3	3
x_4	2	2	3	3

Figura 5.15 – Agendas das variáveis

Considere que todas as variáveis estão a um nível de 0.5. Qualquer que seja o fluxo que se escolha, o seu valor é 0 ou 1, sendo todos inteiros. De facto, esta solução tem exactamente os mesmos fluxos e o mesmo valor da solução $x = [0,1,1,0]$, que é inteira. Embora esta situação específica seja artificial, em instâncias grandes, ocorrem frequentemente situações semelhantes. ■

O esquema de partição que se propõe pode ser sintetizado no seguinte diagrama:

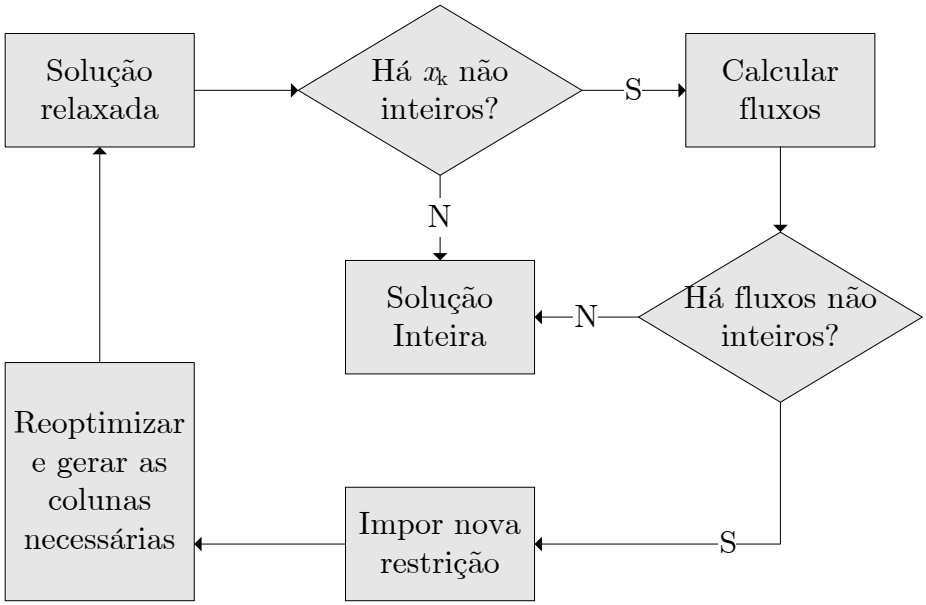


Figura 5.16 – Esquema de partição e geração de colunas

Partindo de uma qualquer solução relaxada, verifica-se se essa solução tem alguma variável x_k fraccionária. Se não houver variáveis fraccionárias está encontrada uma solução inteira, caso contrário, é necessário calcular todos os fluxos S_{jk}. Neste ponto, pode acontecer (e nos testes computacionais realizados acontecia com frequência) que todos os fluxos sejam inteiros. Caso isto se verifique basta aplicar o algoritmo apresentado em 5.2.2 aos fluxos obtidos para se calcular uma solução inteira com o mesmo valor da solução fraccionária que lhe esteve na origem.

Caso nem todos os fluxos sejam inteiros, escolhe-se um fluxo sobre o qual se fará incidir uma restrição de partição. Denote-se esse fluxo por S^*_{jt}. Serão

então criados dois problemas cujos espaços de soluções são mutuamente exclusivos. Os problemas são em tudo iguais ao original excepto na nova restrição de partição, em que um deles recebe uma restrição do tipo $S_{jt} \leq \left\lfloor S^*_{jt} \right\rfloor$ e o outro uma restrição do tipo $S_{jt} \geq \left\lceil S^*_{jt} \right\rceil$. Desta forma há um conjunto de soluções fraccionárias que são eliminadas do processo de resolução.

O impor sucessivo de restrições de partição e criação de novos problemas vai então criar uma árvore de pesquisa que pode ser explorada segundo várias estratégias (ver secção 3.4).

5.5.1 Implicações no Subproblema

Ao acrescentar uma nova restrição a estrutura do problema principal modificou-se. Para que o subproblema continue a identificar as colunas mais atractivas para introduzir neste novo problema principal é necessário que o primeiro seja modificado. Fundamentalmente, a modificação deve acomodar no processo de resolução do subproblema a informação dual resultante das novas restrições introduzidas.

Neste caso, as restrições que vão sendo introduzidas apenas dizem respeito às variáveis que agendam determinada tarefa, j, em determinado período de agendamento, t. Logo, o valor da variável dual associada a determinada restrição pode ser visto como um prémio / penalidade adicional ao agendamento da tarefa T_j no período de tempo I_t. Esta alteração é bastante fácil de acomodar no subproblema pois apenas modifica o cálculo dos custos de mudança de estado na formulação de programação dinâmica.

Considere-se que em determinado nodo da árvore de pesquisa há impostos dois conjunto de restrições de partição, G e H, tais que:

$$\sum_k \delta_k^{jt} x_k \leq \left\lfloor S^l_{jt} \right\rfloor \quad \forall l \in G$$

$$\sum_k \delta_k^{jt} x_k \geq \left\lceil S^l_{jt} \right\rceil \quad \forall l \in H$$

O coeficiente δ_k^{jt} assume o valor 1 se, na coluna k, a tarefa j estiver agendada no período t, e 0 caso contrário. Sejam ainda μ e ν dois vectores de

variáveis duais associadas, respectivamente, às restrições do conjunto G e às restrições do conjunto H.

A nova equação de recorrência do modelo de programação dinâmica pode ser definida da seguinte forma:

$$F_t(j) = \max\left(F_{t-1}(l) + \pi^a_{jt} - w^t_j - p^a_{lj}\right)$$

Onde anteriormente figurava π^a_j agora figura π^a_{jt} que pode ser calculado pela seguinte expressão:

$$\pi^a_{jt} = \begin{cases} 0, & j = 0 \\ \pi_j + \sum_{l \in G_{jt}} \mu_l + \sum_{l \in H_{jt}} \nu_l, & j > 0 \end{cases}$$

Em que G_{jt} e H_{jt} são subconjuntos de G e H que apenas contêm restrições que incidam sobre a tarefa T_j e sobre o período I_t. Desta forma, aos prémios e penalidades ditadas pelas restrições do problema original são somados os prémios e penalidades por agendar determinada tarefa em determinado período de tempo resultantes das restrições de partição.

Retome-se agora o exemplo que tem vindo a ser tratado para ilustrar a aplicação do método de partição apresentado nesta secção:

Exemplo 5.2 (continuação)

Atente-se agora nos fluxos calculados na Tabela 5.3. Para iniciar o processo de partição escolha-se um qualquer fluxo de forma arbitrária. Neste caso vamos escolher o fluxo com menor j e, em caso de empate, o de menor t. Utilizando este critério, o fluxo a partir será o fluxo $S_{22} = 0.75$. A partir do nodo raiz da árvore de pesquisa (relaxação linear) vão ser criados dois novos ramos (problemas), cópias exactas do problema que lhes deu origem (o problema do nodo pai). A um dos problemas será acrescentada a restrição $S_{22} \leq 0$ ($\lfloor 0.75 \rfloor$) e ao outro a restrição $S_{22} \geq 1$ ($\lceil 0.75 \rceil$). Deste modo, o espaço de soluções dos dois problemas é mutuamente exclusivo e é eliminado o conjunto de soluções fraccionárias em que $0 < S_{22} < 1$.

A resolução do primeiro problema resulta no seguinte quadro final:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7		π
T_1	3	2	0	0	0	5	0	$= 5$	9.5[3]
T_2	2	2	1	4	5	0	4	$= 5$	10
T_3	0	1	4	0	0	0	1	$= 5$	9.[6]
M	1	1	1	1	1	1	1	$\leqslant 3$	-36.[6]
$S_{22} \leqslant 0$	0	0	0	1	1	0	0	$\leqslant 0$	-2.[3]
min	12	13	12	1	11	11	13		
x	0	0	1	0	0	1	1		36

Figura 5.17 – Quadro após introdução da restrição $S_{22} \leq 0$

Na resolução procedeu-se à geração de colunas que resultou na adição da variável x_7 com a agenda 2-3-2-2-2. Foi encontrada uma solução inteira de valor 36.

Por sua vez, a resolução do outro problema resulta no quadro final que se segue:

	x_1	x_2	x_3	x_4	x_5	x_6	x_8		π
T_1	3	2	0	0	0	5	0	$= 5$	10.0[6]
T_2	2	2	1	4	5	0	2	$= 5$	10
T_3	0	1	4	0	0	0	3	$= 5$	10.[3]
M	1	1	1	1	1	1	1	$\leqslant 3$	-39.[3]
$S_{22} \geqslant 1$	0	0	0	1	1	0	1	$\geqslant 0$	0.[3]
min	12	13	12	1	11	11	12		
x	0	0	1	0	0.[6]	1	0.[3]		34.[3]

Figura 5.18 – Quadro após introdução da restrição $S_{22} \geq 1$

Neste caso, foi adicionada a variável x_8 cuja agenda é 2-2-3-3-3. Como se pode verificar, esta solução continua a ser fraccionária, no entanto o seu valor passou de 34.25 para 34.[3].

Nesta iteração foi possível baixar o valor do limite superior de 37 (solução inicial) para 36 (solução dada pelo primeiro problema da partição). O limite inferior continua em 35 ($\lceil 34.[3] \rceil$). Logo, pode concluir-se que a solução óptima terá um valor de 35 ou 36.

Na figura que se segue representa-se graficamente a informação obtida na resolução dos dois problemas resultantes da partição:

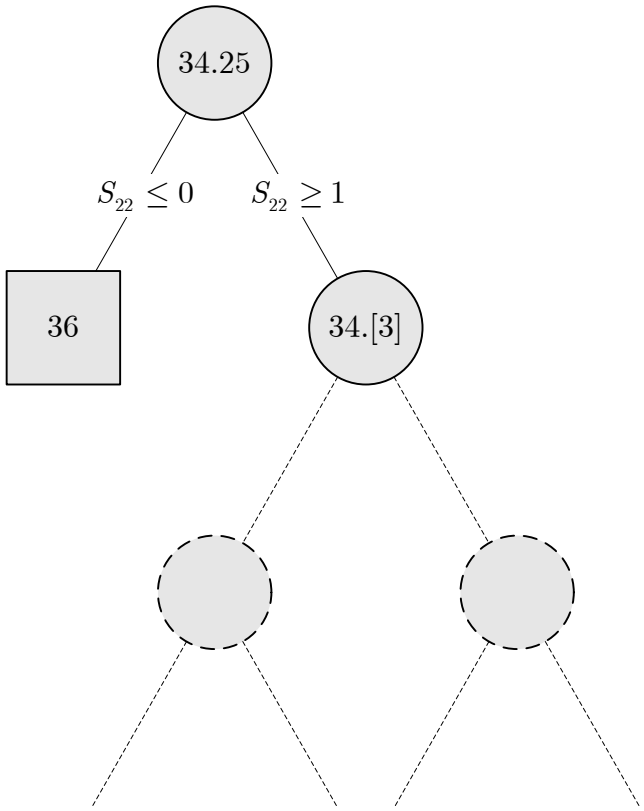


Figura 5.19 – Árvore de pesquisa

As soluções fraccionárias são representadas por círculos e as soluções inteiras por quadrados. A parte a tracejado representa a parte da árvore que ainda falta explorar. Note-se que não é necessário explorar todos os ramos da árvore de pesquisa até à obtenção de soluções inteiras ou problemas impossíveis: como temos um limite superior, neste caso, qualquer nodo com solução fraccionária com valor superior a 35 poderia ser imediatamente abandonado pois, na melhor das hipóteses, apenas se encontraria uma solução de valor igual à

existente (36). Em termos práticos, apenas interessa encontrar soluções de valor inferior a 36 ou provar que elas não existem. ■

5.6 Conclusão

Neste capítulo foi apresentada a metodologia básica desenvolvida neste trabalho. Esta metodologia pode ser usada para resolver uma categoria de problemas genéricos descritos em 5.1.

Como frequentemente acontece nos processos de geração de colunas, estes são bastantes rápidos a encontrar soluções de boa qualidade, mas podem ser extremamente lentos a chegar à solução óptima.

Nos próximos capítulos são apresentadas algumas técnicas utilizadas para acelerar o processo de partição e avaliação e são descritos alguns casos particulares do problema onde foi possível fazer ainda mais melhorias, tirando vantagem de características específicas desses casos particulares.

Capítulo 6

Algoritmo de Partição e Geração de Colunas

6.1 Introdução

Tal como se referiu no final do capítulo anterior, por norma, os algoritmos de geração de colunas chegam rapidamente a soluções de boa qualidade mas são muito lentos a encontrar a solução óptima ou a provar que determinada solução em mão é óptima. Para resolver este problema também são conhecidas várias técnicas. Neste capítulo descrevem-se um conjunto de técnicas que foram experimentadas ou implementadas na resolução do problema em questão.

6.2 Cálculo de Limites Inferiores

Uma das técnicas que pode ser utilizada para acelerar o processo de partição e avaliação é o cálculo de limites inferiores, com o objectivo de abandonar, precocemente e com segurança, ramos não promissores da árvore de pesquisa. Nesta secção são descritos alguns métodos que podem ser utilizados no problema em estudo.

6.2.1 Limite Inferior da Relaxação Linear

Se todos os coeficientes da função objectivo forem inteiros (o que acontece se as penalidades por preparações (p) e as penalidades por atrasos (w_j) forem inteiras) há garantia (uma vez que as variáveis de decisão também são inteiras) de que o valor da função objectivo de qualquer solução válida seja um valor inteiro. Isto permite derivar um limite inferior trivial para o valor da solução num dado nodo da árvore de pesquisa.

Seja z_{RL} o valor da solução da relaxação linear num determinado nodo. O valor de qualquer solução válida, z_{INT}, nesse ramo da árvore de pesquisa obedece à desigualdade

$$z_{INT} \geq \lceil z_{RL} \rceil$$

Como já foi referido na secção 5.4, este limite pode ser introduzido na formulação do problema primal restrito sob a forma da seguinte restrição:

$$\sum_k c_k x_k \geq \lceil z_{RL} \rceil$$

Embora este tipo de restrições tenham sido utilizadas na resolução de outros problemas (só a título de exemplo ver [49]), testes computacionais preliminares demonstraram que a introdução desta restrição no caso concreto atrasa o processo de convergência. Embora não tendo conhecimento de um caso paralelo na literatura, pensamos que, pelo facto desta restrição ser muito forte e tornar os preços duais de praticamente todas as outras restrições nulos, a qualidade da informação dual que é fornecida ao subproblema seja prejudica-

da, atrasando deste modo a identificação das melhores colunas para formar a solução óptima.

Sem prejuízo do que foi dito no parágrafo anterior, este limite inferior continua a ser útil para decidir o abandono precoce de ramos da árvore de pesquisa quando a diferença entre o valor da melhor solução inteira (z_{INC}) e o valor da relaxação linear for inferior a uma unidade, ou seja, se $\lceil z_{RL} \rceil = z_{INC}$ o nodo em questão pode ser abandonado.

6.2.2 Limite Inferior Derivado do Modelo de Fluxos

O limite inferior que se descreve de seguida é válido quando, para qualquer tarefa, $w_j \gg p$, isto é, quando o custo em que se incorre por uma unidade processada com atraso é muito maior que o custo de uma preparação. Esta restrição é comum em problemas reais, uma vez que, por norma, em ambientes de máquinas paralelas os tempos de preparação tendem a ser relativamente baixos em relação às unidades de tempo utilizadas para o planeamento, resultando daqui que atrasar o trabalho em uma unidade de planeamento será sempre pior que incorrer em várias preparações.

Este limite inferior baseia-se na solução fornecida pelo problema de fluxo de custo mínimo descrito em 5.2. Para o valor de qualquer solução há duas contribuições: a quantidade de trabalho atrasado e a quantidade de preparações. No problema de fluxo de custo mínimo é obtido um mínimo para a primeira dessas contribuições, a quantidade e o custo do trabalho atrasado por tarefa. Designe-se o custo do trabalho atrasado (óptimo do problema de fluxos) por L^C e a quantidade de trabalho atrasado (número de unidades agendadas com atraso) por L^Q. Logo, o valor da solução óptima do problema não pode ser inferior a L^C adicionado do custo das preparações. Para que se obtenha um limite inferior falta apenas calcular o número mínimo de preparações necessárias.

Para cada tarefa deve ser agendada uma quantidade p_j. Divida-se essa quantidade em duas partes: uma parte igual a L^Q que pode ser agendada com

atraso, e uma parte O_j^Q, tal que $p_j = O_j^Q + L^Q$, que, impreterivelmente, tem que ser agendada na janela de tempo $\left[r_j, d_j\right]$. Assim, o número mínimo de preparações necessárias para agendar O_j^Q na janela de tempo admissível pode ser calculado pela expressão que se segue:

$$N_j^P = \left\lceil \frac{O_j^Q}{d_j - r_j} \right\rceil$$

Então, o limite inferior para a solução óptima pode ser calculado da seguinte forma:

$$z_{LI} = L^C + p\sum_j N_j^P$$

Os resultados computacionais indicaram que este limite inferior é melhor que o limite da relaxação linear e funciona tanto melhor quanto menor for o valor de L^Q, pois no caso em que há menos atrasos é possível calcular limites inferiores para o número de preparações mais robustos.

No entanto, pelos motivos já apresentados na secção anterior, a inclusão da restrição

$$\sum_k c_k x_k \geq z_{LI}$$

que reforça a formulação do problema primal restrito, não conduziu a bons resultados computacionais.

6.2.3 Extensão do Modelo de Fluxos à Fase de Partição

6.2.3.1 Introdução

Os limites inferiores calculados em 6.2.2 são normalmente melhores que os dados pela relaxação linear, mas tornam-se menos efectivos à medida que se acrescentam restrições de partição e o valor da relaxação linear se vai aproximando daqueles. Isto acontece porque o espaço de soluções da relaxação linear vai sendo restringido sem que a informação sobre as novas restrições seja utilizada para melhorar os limites inferiores dados pelo modelo de fluxos.

O que se propõe nesta secção é incorporar as restrições de partição no cálculo de limites inferiores utilizando o modelo de fluxos.

Impor uma restrição de partição no problema primal restrito corresponde a impor restrições sobre um fluxo no modelo de fluxo de custo mínimo. Logo, em cada nodo da árvore de pesquisa, o modelo de fluxo de custo mínimo pode ser utilizado para recalcular os valores de L^C e de L^Q.

Adicionalmente, é possível introduzir mais alguma informação sobre as restrições de partição no processo de cálculo do número mínimo de preparações. Aqui, a ideia é explorar os efeitos da imposição de uma restrição do tipo $S_{jt} \leq a$. Ao impor uma restrição deste tipo, limita-se o número de máquinas a utilizar no período t para a tarefa j ao inteiro a. Ora o cálculo de N_j^P apresentado na secção 6.2.2 tem implícito que em qualquer período da janela de tempo $\left[r_j, d_j\right]$ se pode utilizar um qualquer número de máquinas. Se dentro daquela janela houver um ou vários períodos em que o número de máquinas disponíveis esteja limitado, o número mínimo de preparações necessárias pode ter que ser revisto para cima. Considere-se o seguinte exemplo:

Exemplo 6.1

Considere-se a necessidade de agendar 10 unidades da tarefa j na janela $[0,5]$. Não havendo restrições, o número de preparações necessárias é 2. Imagine-se agora que no período $[2,3]$ se impõe a restrição $S_{j3} \leq 1$, isto é, no referido período apenas é possível utilizar uma máquina. Neste caso, a única forma de agendar as 10 unidades dentro da janela admissível é incorrendo em 4 preparações. A figura que se segue ilustra uma das soluções possíveis:

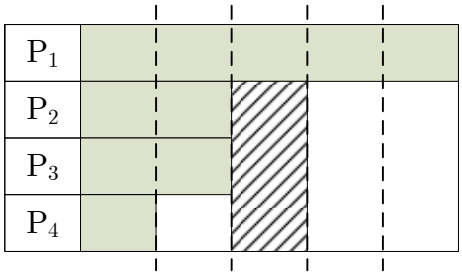


Figura 6.1 – Agendamento com uma restrição

Apenas estão representados 4 processadores, mas os resultados são extensíveis a qualquer número. A área a tracejado representa a impossibilidade de agendar naqueles processadores devido à imposição de uma restrição. As áreas a sombreado representam as 10 unidades agendadas. Como é facilmente verificável, havia outras possibilidades de efectuar o agendamento. ■

No caso de haver uma restrição a limitar o número de máquinas em determinado período, as coisas são mais ou menos óbvias e é fácil calcular o número mínimo de preparações. Sempre que se impõe uma restrição, há um conjunto de máquinas (eventualmente todas) cujas janelas válidas para o agendamento são reduzidas ou divididas em duas janelas mais pequenas (como no exemplo). Havendo um conjunto de janelas de tamanhos diferentes onde é possível agendar, o número mínimo de preparações é obtido agendando em primeiro em primeiro lugar as maiores janelas.

No entanto, à medida que se vão acrescentando mais restrições, haverá eventualmente um momento em que é necessário impor sobre uma determinada tarefa uma segunda restrição mas num período de tempo diferente. Nesta situação passa a haver desde logo várias possibilidades de construção das janelas, com influência no número mínimo de preparações. Atente-se no exemplo:

Exemplo 6.2

Adicione-se aos dados do Exemplo 6.1 uma nova restrição a incidir sobre o período $[1,2]$, tal que, $S_{j2} \leq 2$.

Como facilmente se pode verificar a inclusão desta restrição é incompatível com a solução apresentada na Figura 6.1, uma vez que no período 2 são ocupadas 3 máquinas.

Como também facilmente se constata, o número mínimo de preparações continua a ser 4, bastando passar as quantidades agendadas nas máquinas 2, 3 e 4 nos períodos 1 e 2 para os períodos 4 e 5. e dizendo que, por exemplo, a

restrição não permitirá agendar as máquinas 3 e 4 no período 2. A solução pode ser visualizada na figura que se segue:

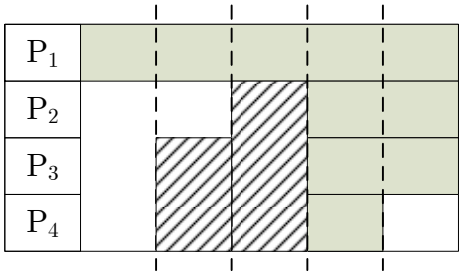


Figura 6.2 – Agendamento óptimo após nova restrição

No entanto também poderíamos ter optado por construir as janelas como na figura que se segue:

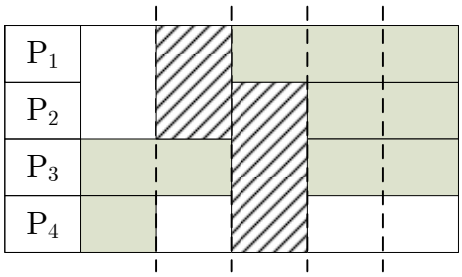


Figura 6.3 – Outro agendamento possível

Neste caso, como se pode verificar, o número mínimo de preparações é igual a 5, permitindo demonstrar que a construção das janelas não é irrelevante. ■

Uma solução óbvia seria o estudo de todas as possibilidades de construção das janelas, mas como facilmente se conclui, mesmo com poucas restrições a incidir sobre uma tarefa, teríamos um número bastante grande de combinações e o cálculo tem que ser repetido em todos os nodos da árvore e para todas as tarefas, gerando uma carga computacional incomportável.

No entanto, foi possível desenvolver uma heurística para efectuar estes cálculos em tempo polinomial. Trata-se de uma heurística do tipo "ganancioso" que consegue encontrar o número mínimo de preparações. A heurística é apresentada de seguida.

6.2.3.2 Heurística de Arranjo de Janelas

Considere-se então um nodo onde estão impostas um conjunto G composto por l restrições do tipo $S_{j_l t_l} \leq a_l$. O conjunto G pode ser subdividido em n conjuntos mutuamente exclusivos, $G_j = \{l : j_l = j\}$ que correspondem aos subconjuntos de restrições que incidem sobre cada uma das tarefas. Para calcular o número mínimo de preparações para cada tarefa é necessário retirar do problema de fluxos os valores de O_j^Q. Para cada período de agendamento é necessário calcular o número de máquinas em que se vai bloquear o agendamento, b_t, que pode ser calculado por

$$b_t = \max\{0, m - a_l : t_l = t\}$$

Agora, para cada período, é necessário colocar os blocos que proíbem o agendamento nas máquinas mais adequadas. O algoritmo consiste em, para qualquer período de tempo, colocar o blocos necessários nas b_t máquinas de índice mais baixo (poder-se-ia utilizar outra ordem qualquer). Ao proceder desta forma assegura-se que as máquinas de índice mais alto ficam com janelas de agendamento maiores onde será depois possível agendar as tarefas com o mínimo de interrupções. O procedimento deve repetir-se para cada tarefa. A figura que se segue ilustra o resultado da aplicação desta heurística para uma tarefa:

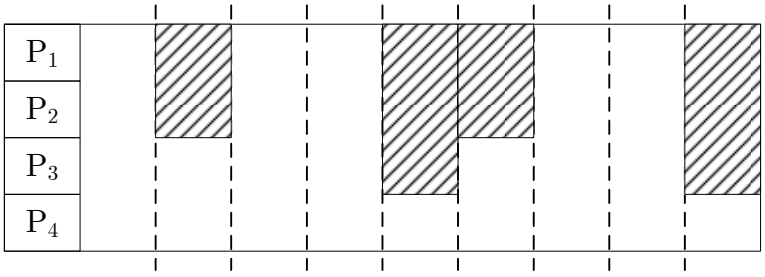


Figura 6.4 – Heurística de colocação de blocos

Nesta ilustração há 4 períodos de tempo com restrições impostas sobre a tarefa considerada. O algoritmo faz com que os blocos, sempre que possível, fiquem em máquinas onde há outros blocos relativos a outros períodos. Como

se pode ver as máquinas de índice mais elevado tendem a ficar com janelas de agendamento maiores.

Proposição 6.1

As janelas de agendamento resultantes deste procedimento conseguem agendar qualquer tarefa com o mínimo de preparações.

Prova

Considere-se uma solução produzida pela heurística proposta. A essa solução podem ser introduzidas alterações relevantes ou irrelevantes. Uma alteração irrelevante é quando a mudança de um bloco cria uma nova solução mas cujas janelas resultantes são exactamente iguais às da solução original, apenas em máquinas diferentes. Uma alteração relevante cria um conjunto de janelas diferentes das que existiam originalmente.

Afirmar que a solução produzida pela heurística não é óptima equivale a dizer que existe uma outra solução que pode ser obtida a partir desta, através de uma ou mais alterações relevantes, onde é possível agendar com menos preparações.

Quando se faz uma alteração relevante a uma solução produzida pela heurística, move-se um determinado bloco de uma janela de tamanho W_0 para uma janela de tamanho W_1, com $W_1 > W_0$. A figura que se segue esquematiza a alteração:

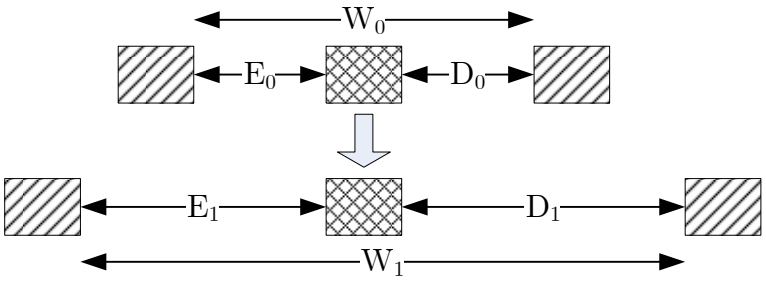


Figura 6.5 – Esquema de movimentação de blocos

Ao transferir o bloco que está na janela W_0 para a janela W_1, as janelas E_0 e D_0 desaparecem, sendo substituídas por uma janela de tamanho $W_0 = E_0 + D_0 + 1$. A janela W_1 também desaparece, sendo substituída por duas janelas, E_1 e D_1, sendo $W_1 = E_1 + D_1 + 1$. Na figura que se segue são comparados graficamente os tamanhos das várias janelas:

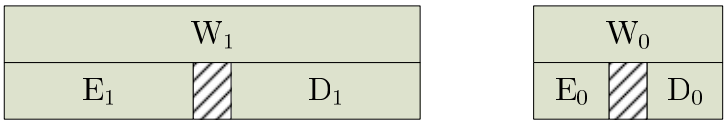


Figura 6.6 – Comparação das várias janelas

Ordenem-se agora por valor decrescente do tamanho as janelas que existem numa e noutra solução. Na solução original há janelas de tamanho W_1, W_0, E_0 e D_0 (sem perda de generalidade, admita-se que $E_0 \geq D_0$). Na solução alterada, há janelas de tamanho E_1, D_1, W_0 (a ordem de tamanho destas janelas poderia ser qualquer uma. Para melhor visualização, representem-se graficamente estas janelas, ordenadas pelo tamanho:

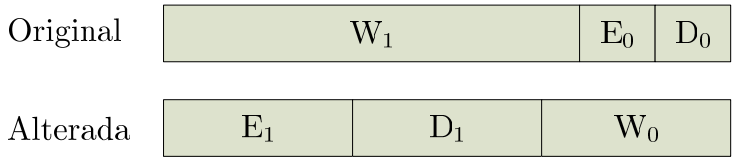


Figura 6.7 – Comparação das soluções

Havendo uma determinada quantidade para agendar nas janelas, quanto maiores forem as janelas, menor será o número de preparações. Ao olhar para a figura é imediata a conclusão que, qualquer que seja a quantidade a agendar naquelas janelas, o número de preparações será sempre menor, ou quando muito igual, na solução original.

Embora no problema haja depois outras janelas, elas são comuns às duas soluções, de modo que, qualquer que seja o conjunto de janelas comuns que se acrescentem às duas soluções, a conclusão é invariável. ■

Ficando demonstrado que a efectivação de uma alteração relevante à solução fornecida pela heurística nunca contribui para uma diminuição do número de preparações, por maioria de razão, efectivar várias alterações, também não pode contribuir para essa diminuição.

A utilização deste procedimento para calcular o número mínimo de preparações a efectuar para o agendamento de cada tarefa, juntamente com os resultados obtidos do problema de fluxo de custo mínimo modificado, à semelhança do que se fez na secção anterior, permite ir elevando o valor do limite inferior à medida que se acrescentam restrições.

6.2.3.3 Variáveis com Valor Nulo

Ainda nos casos em que $w_j \gg p$, é possível utilizar o modelo de fluxos para implementar computacionalmente outras melhorias.

Quando são geradas novas colunas, no subproblema, qualquer tarefa pode ser agendada com um atraso máximo igual a $T_{max} - d_j$, ou seja, a diferença entre o horizonte de agendamento e a data de conclusão. No entanto, como o valor de L^C (custo global do trabalho atrasado) é conhecido e a solução óptima não o pode ultrapassar, o custo com o trabalho atrasado em cada coluna também não pode ultrapassar o valor de L^C, ficando assim imposto um limite ao trabalho atrasado de cada tarefa, que pode ser menor que o limite genérico apresentado acima.

Por exemplo, se for concluído no problema de fluxos que não há necessidade de trabalho atrasado, nenhuma coluna gerada deverá incluir trabalho atrasado.

A implementação do que foi descrito conduz à não geração de colunas que nunca iriam fazer parte da solução óptima, mas que, transitoriamente, poderiam fazer parte da solução.

6.2.3.4 Limites Superiores para as Variáveis

O valor de L^C, quando positivo, também pode ser utilizado para impor limites superiores às variáveis de decisão, embora isto não possa ser feito directamente devido ao problema da regeneração de colunas.

Seja L_k^C o custo com o trabalho atrasado associado a uma qualquer coluna k. Um limite superior trivial para a variável x_k é dado por $\left\lfloor L^C / L_k^C \right\rfloor$. No entanto, a simples imposição deste limite recorrendo ao software de optimização utilizado, sem dar conhecimento da restrição ao subproblema, resultaria na geração de uma coluna igual à que foi limitada, sempre que necessário. A este efeito é dado normalmente o nome de regeneração de colunas, e, a não ser evitado, provoca o aparecimento de ciclos viciosos no algoritmo de geração de colunas. É também por este motivo que a partição nunca se faz directamente sobre as variáveis do problema principal.

6.3 Soluções Iniciais

Ao resolver problemas através de técnicas de partição e avaliação, como é o presente caso, é de grande importância ter desde a primeira hora um intervalo de pesquisa pequeno, isto é, ter limites superiores e inferiores o mais próximos possível. Isto diminui a necessidade de explorar partes consideráveis da árvore de pesquisa. Enquanto na secção 6.2 se tratou de aumentar os limites inferiores, nesta secção e na seguinte, fala-se em diminuir os limites superiores.

Na abordagem adoptada para a resolução deste problema o limite superior é sempre o valor da melhor solução existente em determinado momento. Logo, é importante obter rapidamente boas soluções.

A primeira solução válida é obtida através da resolução de um problema de fluxo de custo mínimo que não tem em conta os custos associados às preparações. Tal como se afirmou na secção 5.2, isto resulta numa solução inicial com um número arbitrariamente grande de preparações.

Expõe-se de seguida uma técnica de perturbação da matriz de custos do problema de fluxo de custo mínimo que visa a obtenção de soluções com um número mais reduzido de preparações. Considere-se a seguinte figura que representa parte da rede do problema de fluxo de custo mínimo utilizado para obter as soluções iniciais:

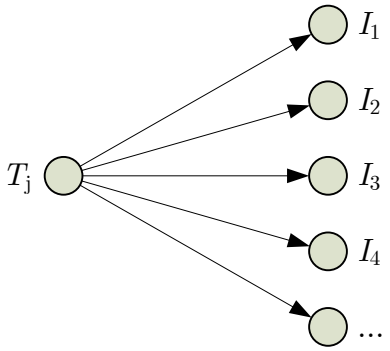


Figura 6.8 – Rede parcial

Para os intervalos I_t que se encontram dentro da janela de agendamento $\left[r_j, d_j\right]$, os valores de custo inscritos no arco $\left(T_j, I_t\right)$ são sempre iguais a 0. Pela natureza dos algoritmos utilizados pelos pacotes de programação linear para problemas de rede, a resolução do problema atribui quase sempre uma quantidade tão grande quanto possível de fluxo, a um número reduzido de arcos.

Ao transformar a solução devolvida pelo problema de fluxo de custo mínimo numa solução para o problema de agendamento, sempre que determinados arcos têm grandes quantidades de fluxo, comparativamente com os arcos correspondentes aos períodos vizinhos, isso implica um grande número de preparações (ver a secção 5.2.2).

Por outro lado, se o valor de fluxo for mais ou menos constante em todos os intervalos de tempo, o número de preparações é bastante mais reduzido, ficando no limite, igual ao valor desse fluxo constante.

Para forçar o equilíbrio dos fluxos nos arcos de determinada tarefa, acrescentou-se à função objectivo do problema de fluxo de custo mínimo uma função

que penaliza a quantidade de fluxo nesse arco de forma quadrática. A função objectivo original do problema era:

$$\min \sum_{j} \sum_{t} c_{jt} S_{jt}$$

Passando a ser:

$$\min \sum_{j} \sum_{t} \left(c_{jt} S_{jt} + \varepsilon S_{jt}^{2} \right)$$

Como o valor da perturbação cresce de forma quadrática com o aumento de S_{jt}, a solução tenderá a nivelar os valores de S_{jt}.

A escolha de um termo quadrático para impor a perturbação prende-se principalmente com o facto de o problema resultante ser um problema de programação separável com uma sub matriz de problema de fluxo de custo mínimo, problema este que é resolvido eficientemente pelo pacote de *software* de optimização utilizado (e nem que não fosse, só seria resolvido uma vez, no início do processo de geração de colunas). Por outro lado, dado a simplicidade do objectivo que se pretende atingir, não há necessidade de recorrer a funções mais complexas.

6.4 Melhoria Local de Soluções Válidas

Outra forma de melhorar o limite superior do processo de partição e avaliação consiste em melhorar localmente as solução válidas encontradas. Esta técnica pode aplicar-se tanto à solução inicial, como às sucessivas soluções encontradas durante a pesquisa. Propõe-se uma heurística bastante simples que, embora veja o potencial de melhoria significativamente reduzido à medida que se encontram melhores soluções, pode dar uma contribuição para a redução dos limites superiores numa fase inicial.

Quando as soluções são de pior qualidade, como é o caso das soluções iniciais derivadas do problema de fluxo de custo mínimo e, eventualmente, das primeiras soluções encontradas na árvore de pesquisa, é normal haver nos processadores tarefas que são interrompidas e retomadas mais vezes que o necessário. Um simples agregar dos vários fragmentos espalhados pelo horizonte

temporal do processador conduz por vezes a significativas economias no número de preparações.

Dada uma solução válida, a heurística analisa o trabalho agendado em cada processador individualmente. O objectivo é reorganizar as tarefas que estão afectas a cada um dos processadores.

Para isso, é percorrido o horizonte temporal de cada processador, agendando em cada período a tarefa que, de entre as disponíveis, tenha a data de conclusão mais próxima, sendo o desempate feito por um critério arbitrário. Para evitar complicações relacionadas com o trabalho atrasado e para manter a heurística computacionalmente simples, esta não move o trabalho agendado originalmente como atrasado.

Como há o risco (no caso de tarefas com janelas de execução encapsuladas[4], por exemplo) de a heurística piorar a solução original, só se deve implementar a nova solução se for melhor que a original. A figura que se segue ilustra uma dessas situações:

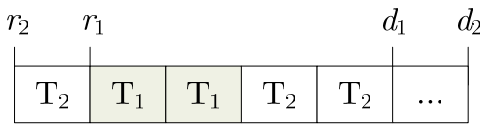


Figura 6.9 – Aplicação da heurística de melhoria local

No exemplo proposto, a janela de execução de T_1 está encapsulada na janela de execução de T_2. Facilmente se conclui que as 2 unidades de T_1 poderiam ser agendadas depois das 3 unidades de T_2, poupando 1 preparação. No entanto, a aplicação da heurística 1 conduz à solução apresentada na figura, podendo desta forma piorar uma solução que tivesse o arranjo óptimo.

Como se pode verificar, a heurística é de grande simplicidade computacional, podendo ser usada sem reservas, mesmo sabendo da baixa probabilidade de obter melhorias em soluções de melhor qualidade.

[4] A janela de execução de uma tarefa T_i diz-se encapsulada na janela de execução de outra tarefa T_j se $r_i > r_j$ e $d_i < d_j$ forem ambas verdadeiras.

6.5 Arredondamento de Soluções Fraccionárias

Uma outra forma que se encontrou para obter soluções inteiras foi através do arredondamento de soluções fraccionárias. Nesta secção é apresentada uma heurística capaz de arredondar em tempo polinomial qualquer solução fraccionária retirada da relaxação do problema primal restrito.

Relembre-se da secção 5.5 que numa solução fraccionária há um conjunto de fluxos, S_{jt} que representam a quantidade de tarefa T_j agendada no período t na totalidade das máquinas. Verificou-se na altura que, para que uma solução seja inteira, é condição necessária e suficiente que todos os fluxos S_{jt} sejam inteiros. Isto implica que, numa solução fraccionária, nem todos os fluxos S_{jt} são inteiros. A heurística que agora se apresenta faz o arredondamento de todos os fluxos S_{jt}.

A heurística procede ao arredondamento período a período, iniciando no período $t=1$ e prosseguindo sequencialmente até ao período $t=t_{max}-1$ (se os fluxos dos períodos anteriores forem todos inteiros, os fluxos do período $t=t_{max}$ também o serão, uma vez que são a última parcela de uma soma que deve ser inteira e igual a p_j). Em cada período a primeira operação é determinar a quantidade total a agendar nesse período, que deve ser inteira uma vez que será a soma de fluxos S_{jt} inteiros. Seja $S_t=\sum_j S_{jt}$ a quantidade total inicialmente agendada no período t. É trivial concluir que S_t é uma quantidade entre 0 e m, podendo ser fraccionária ou inteira. Seja S'_t a quantidade a agendar na solução arredondada. Faça-se o valor de S'_t igual ao menor inteiro igual ou superior a S_t, ou seja, $S'_t=\lceil S_t \rceil$. Note-se que, desta forma, nunca poderá ser calculado um valor superior a m para S'_t.

O motivo pelo qual se arredonda para cima a quantidade total a agendar tem a ver com o evitar que se chegue a períodos posteriores e se necessite de arredondar uma quantidade superior a m, que seria impossível de agendar.

Uma vez estabelecida a quantidade a agendar no período, é necessário arredondar todos os S_{jt} relativos a esse período, garantindo que a sua soma seja igual a S'_t.

O procedimento que se adoptou é baseado na manutenção de um indicador dos arredondamentos efectuados em valores o mais próximos possível de 0. Designe-se por Δ esse indicador.

No início do arredondamento de cada período, $\Delta = 0$. Depois de calcular S'_t o valor de deve ser actualizado, fazendo $\Delta = S_t - S'_t$. Desta forma, Δ tomará um valor no intervalo $]-1,0]$.

Sempre que no período se proceder ao arredondamento de um qualquer valor S_{jt} o valor de Δ deve ser actualizado, bem como os valores dos fluxos dos períodos seguintes. Em primeiro lugar é necessário calcular a diferença no arredondamento específico, Δ_{jt}. Se o arredondamento for para cima, $S'_{jt} = \lceil S_{jt} \rceil$, enquanto que se o arredondamento for para baixo, $S'_{jt} = \lfloor S_{jt} \rfloor$. Logo a diferença pode ser calculada fazendo $\Delta_{jt} = S'_{jt} - S_{jt}$. Depois do arredondamento, $\Delta = \Delta + \Delta_{jt}$. Como a quantidade arredondada necessita de ser tomada em consideração nos períodos seguintes, é necessário fazer $S_{j(t+1)} = S_{j(t+1)} - \Delta_{jt}$. Se resultarem deste último cálculo fluxos $S_{j(t+1)}$ negativos, no período seguinte esses fluxos serão imediatamente arredondados para 0, de acordo com o procedimento descrito.

Para efectuar os arredondamentos é necessário escolher qual a tarefa a arredondar (das que ainda não têm fluxos inteiros) e determinar se o arredondamento deve ser efectuado para cima ou para baixo.

O critério utilizado na heurística para efectuar esta escolha consiste em escolher o arredondamento que mais aproximar de 0 o valor do indicador dos arredondamentos efectuados, Δ.

Para cada tarefa a necessitar de arredondamento, calcule-se a diferença de arredondamento no caso do arrendamento ser efectuado para cima e no caso deste ser efectuado para baixo, respectivamente, $\Delta^{\uparrow}_{jt}$ e $\Delta^{\downarrow}_{jt}$. A tarefa e o tipo de arredondamento a efectuar são escolhidos de acordo com o mínimo valor de $\left|\Delta + \Delta^{\uparrow}_{ij}\right|$ ou de $\left|\Delta + \Delta^{\downarrow}_{ij}\right|$.

Ao proceder desta forma, o valor de Δ anda sempre em torno de 0 e, no último arredondamento do período é sempre colocado a 0, significando que a

soma dos arredondamentos efectuados em cada tarefa é igual ao arredondamento global do período.

Observe-se agora na prática, através de um exemplo, como o arredondamento é efectuado.

Exemplo 6.3

Considere-se um ambiente de 3 máquinas e 3 tarefas, para um horizonte temporal de 6 períodos. A solução fraccionária que se deseja arredondar é composta por duas variáveis, x_1 e x_2, fixadas a um nível de 1.5 e 1.2 unidades, respectivamente. A variável x_1 corresponde à agenda 1-1-2-2-2-2 e a variável x_2 corresponde à agenda -3-3-3-3-3. Nestas condições a matriz de fluxos será:

$$S = \begin{bmatrix} 1.5 & 1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 1.5 & 1.5 & 1.5 \\ 0 & 1.2 & 1.2 & 1.2 & 1.2 & 1.2 \end{bmatrix}$$

Iniciando o procedimento no período $t = 1$, temos $S_{t=1} = 1.5$ pelo que, neste período, devem ser agendadas $S'_{t=1} = \lceil 1.5 \rceil = 2$ unidades. Assim,

$$\Delta = S_{t=1} - S'_{t=1} = 1.5 - 2 = -0.5$$

Como só uma tarefa apresenta um valor fraccionário, resta escolher se o arredondamento será feito para cima ou para baixo. Calcule-se então $\Delta_{11}^{\uparrow}$ e $\Delta_{11}^{\downarrow}$:

$$\Delta_{11}^{\uparrow} = \lceil 1.5 \rceil - 1.5 = 0.5 \qquad \Delta_{11}^{\downarrow} = \lfloor 1.5 \rfloor - 1.5 = -0.5$$

Para escolher a direcção do arredondamento devem calcular-se as grandezas:

$$\left|\Delta + \Delta_{11}^{\uparrow}\right| = |-0.5 + 0.5| = 0 \qquad \left|\Delta + \Delta_{11}^{\downarrow}\right| = \left|-0.5 + (-0.5)\right| = |-1| = 1$$

Como se pode ver, o arredondamento para cima da tarefa T_1 é o que minimiza o indicador utilizado para seleccionar o arredondamento a efectuar. Na matriz de fluxos, é necessário fazer as seguintes alterações:

$$S'_{11} = S_{11} + \Delta_{11} = 1.5 + 0.5 = 2 \qquad S_{12} = S_{12} - \Delta_{11} = 1.5 - 0.5 = 1$$

A matriz de fluxos actualizada ficará:

$$S = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 1.5 & 1.5 & 1.5 \\ 0 & 1.2 & 1.2 & 1.2 & 1.2 & 1.2 \end{bmatrix}$$

Procedendo agora ao arredondamento do período $t = 2$, temos $S_{t=2} = 1 + 1.2 = 2.2$, $S'_{t=2} = \lceil 2.2 \rceil = 3$ e $\Delta = S_{t=2} - S'_{t=2} = 2.2 - 3 = -0.8$. Mais uma vez, apenas há uma tarefa a necessitar de arredondamento, restando escolher a direcção em que se efectuará:

$\Delta^{\uparrow}_{32} = \lceil 1.2 \rceil - 1.2 = 0.8$, e $\left|\Delta + \Delta^{\uparrow}_{32}\right| = |-0.8 + 0.8| = 0$

$\Delta^{\downarrow}_{32} = \lfloor 1.2 \rfloor - 1.2 = -0.2$, e $\left|\Delta + \Delta^{\downarrow}_{32}\right| = |-0.8 + (-0.2)| = |-1| = 1$

Olhando aos resultados, a tarefa T_3 deve ser arredondada para cima. Procedendo ao arredondamento e actualizando a matriz de fluxos, fica:

$$S = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 1.5 & 1.5 & 1.5 \\ 0 & 2 & 0.4 & 1.2 & 1.2 & 1.2 \end{bmatrix}$$

Para ilustrar o procedimento sempre que haja várias tarefas a arredondar, proceda-se ao arredondamento das quantidades do período $t = 3$. Como temos $S_{t=3} = 1.9$ e $S'_{t=3} = \lceil 1.9 \rceil = 2$, vem $\Delta = S_{t=3} - S'_{t=3} = 1.9 - 2 = -0.1$. Neste caso, é necessário analisar os efeitos dos possíveis arredondamentos para duas tarefas:

$\Delta^{\uparrow}_{23} = \lceil 1.5 \rceil - 1.5 = 0.5$, e $\left|\Delta + \Delta^{\uparrow}_{23}\right| = |-0.1 + 0.5| = 0.4$

$\Delta^{\downarrow}_{23} = \lfloor 1.5 \rfloor - 1.5 = -0.5$, e $\left|\Delta + \Delta^{\downarrow}_{23}\right| = |-0.1 + (-0.5)| = |-0.6| = 0.6$

$\Delta^{\uparrow}_{33} = \lceil 0.4 \rceil - 0.4 = 0.6$, e $\left|\Delta + \Delta^{\uparrow}_{33}\right| = |-0.1 + 0.6| = 0.5$

$\Delta^{\downarrow}_{33} = \lfloor 0.4 \rfloor - 0.4 = -0.4$, e $\left|\Delta + \Delta^{\downarrow}_{33}\right| = |-0.1 + (-0.4)| = |-0.5| = 0.5$

Segundo o critério de escolha, o melhor arredondamento é o da tarefa T_2 para cima. Procedendo ao arredondamento, fica:

$$S'_{23} = S_{23} + \Delta_{23} = 1.5 + 0.5 = 2 \qquad S_{24} = S_{24} - \Delta_{23} = 1.5 - 0.5 = 1$$

O valor de Δ também deve ser actualizado, pois ainda é necessário arredondar a tarefa T_3 com base nesse valor: $\Delta = \Delta + \Delta_{23} = -0.1 + 0.5 = 0.4$. Pode agora verificar-se de que forma se deve arredondar a tarefa T_3:

$$\left|\Delta + \Delta_{33}^{\uparrow}\right| = |0.4 + 0.6| = 1 \qquad \left|\Delta + \Delta_{33}^{\downarrow}\right| = |0.4 + (-0.4)| = 0$$

Concluindo-se pelo arredondamento da tarefa T_3 para baixo. A matriz de fluxos recalculada será:

$$S = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1.5 & 1.5 \\ 0 & 2 & 0 & 1.6 & 1.2 & 1.2 \end{bmatrix}$$

Para completar o arredondamento da solução basta aplicar o processo descrito aos restantes períodos. ■

Depois de obter uma matriz de fluxos composta na totalidade por valores inteiros, basta aplicar o algoritmo descrito em 5.2.2 para obter uma agenda válida para o conjunto de máquinas.

Este algoritmo de arredondamento tem um tempo de execução da ordem de $O\left(n^2 t_{max}\right)$, o que resulta em tempos de execução muito reduzidos, mesmo para instâncias de grande dimensão.

Este algoritmo pode ser utilizado na raiz da árvore de pesquisa para obter uma solução inteira, eventualmente, de qualidade superior à do modelo de fluxos e das heurísticas de pesquisa local, resultando no cálculo de um limite superior melhorado. Como tem um tempo de execução muito baixo, também poderá ser utilizado durante a pesquisa para arredondar soluções quase inteiras, resultando numa eventual redução da profundidade da pesquisa, embora se acredite que a probabilidade de haver reduções significativas seja muito reduzida.

6.6 Pesquisa Parcial

Ao percorrer a árvore de pesquisa, à medida que os limites inferiores para os nodos o permitem, é possível ir abandonando ramos dessa mesma árvore. No entanto, se o limite superior for muito elevado, como é geralmente o caso ao iniciar a pesquisa, sabe-se de antemão que a probabilidade de encontrar soluções de valor significativamente mais baixo que a solução incumbente é muito elevada. Assim, numa fase inicial, a exploração em profundidade é ineficiente.

Numa tentativa de aceleração do processo de pesquisa, pode estabelecer-se um limite superior virtual, abaixo do qual seja bastante provável encontrar uma solução óptima, e fazer a pesquisa como se o valor daquele limite superior fosse o de uma solução incumbente. Desta forma, à medida que os limites inferiores dos nodos fossem atingindo o limite superior virtual, aqueles seriam abandonados, limitando a pesquisa em profundidade.

Esta pesquisa parcial poderá ter dois resultados: ou se encontra uma nova solução de valor inferior ao limite superior virtual, ou é terminada a pesquisa sem se encontrar qualquer solução.

No primeiro caso, sendo encontrada uma nova solução, a pesquisa pode prosseguir normalmente a partir desse ponto ou, caso se justifique, pode proceder-se a uma pesquisa parcial mais restrita.

No outro caso, não sendo encontrada nenhuma solução válida, significa que o limite inferior deve ser subido para o valor do limite superior virtual, e a pesquisa deve retomar-se no espaço de soluções que tinha sido ocultado pelo limite superior virtual. Tal como no caso anterior, se houver justificação, o procedimento de pesquisa parcial pode ser repetido para o novo intervalo.

De uma maneira geral, quando se utiliza esta técnica, os valores do limite superior virtual a estabelecer devem estar relacionados com os resultados que é usual obter no tipo de problema em questão. Se, por exemplo, se souber que o óptimo de um problema está, em média, 15% acima do limite inferior que se possui, o limite superior virtual pode ser estabelecido a 20 ou 30% do limi-

te inferior, havendo boas probabilidades de encontrar a solução óptima nesse intervalo de valores.

6.7 Redução de Simetria

O problema da simetria é recorrente nos processos de geração de colunas. Diz-se que há simetria quando é possível obter soluções idênticas utilizando conjuntos de colunas diferentes. Quando a simetria é eliminada, as árvores de pesquisa ficam mais pequenas, acelerando o processo de partição e avaliação.

No problema em análise, correspondendo cada coluna a uma agenda de um processador, é possível conceber agendas diferentes para processar as mesmas quantidades das mesmas tarefas aos mesmos custos. Considere a seguinte figura:

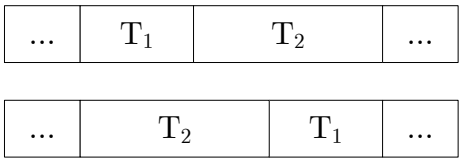


Figura 6.10 - Exemplo de simetria

Na figura estão representadas duas agendas, em tudo iguais, excepto na parte relativa às tarefas T_1 e T_2. Se as datas de disponibilidade e de conclusão o permitirem estas duas agendas têm exactamente o mesmo custo. Em termos de coeficientes na matriz do problema primal, apenas os coeficientes relativos às restrições de partição vão apresentar diferenças. Para evitar a ocorrência de simetria, o subproblema deveria apenas poder gerar uma daquelas colunas.

Note-se que se o exercício anterior fosse feito com 3 tarefas, haveria desde logo 6 colunas idênticas. É fácil concluir que, ocorrendo a simetria de forma generalizada, a árvore de pesquisa pode ficar consideravelmente maior.

No caso em estudo, embora não tivesse sido possível eliminar todos os tipos de simetria, foi possível identificar duas situações que permitem eliminar algumas formas de simetria.

Proposição 6.2

Considere-se um problema de agendamento onde não é necessário recorrer a trabalho atrasado. Para quaisquer duas tarefas T_{j_1} e T_{j_2}, verificando-se que $r_{j_1} \leq r_{j_2}$ e $d_{j_1} \leq d_{j_2}$, não há nenhuma razão para agendar T_{j_2} antes de T_{j_1}.

Prova

Considere-se que as tarefas estão agendadas na sequência T_{j_2}, T_{j_1}. Como a tarefa T_{j_1} está agendada sem atrasos, implica que a tarefa T_{j_2} também está agendada sem atrasos uma vez que $d_{j_1} \leq d_{j_2}$. Logo, se forem trocadas, continuam ambas a ser agendadas sem atrasos. Logo, o custo de uma solução onde as tarefas apareçam na sequência $T_{j_2} - T_{j_1}$ nunca será inferior ao custo de uma solução onde estas tarefas apareçam na sequência inversa. ■

Com base nesta conclusão pode inferir-se o seguinte:

Corolário 6.1

Num problema de agendamento em que não haja necessidade de trabalho atrasado, qualquer subconjunto de tarefas que partilhe as mesmas datas de disponibilidade e de conclusão devem aparecer nas agendas sempre na mesma ordem, que pode ser estabelecida arbitrariamente.

Prova

Como as tarefas têm todas as mesmas datas de disponibilidade e de conclusão e não é necessário agendar com atrasos, a ordem das tarefas é irrelevante. Com vista à total eliminação de agendas simétricas, nas agendas geradas as tarefas devem aparecer sempre pela mesma ordem. ■

Embora haja casos em que estas duas regras não têm qualquer efeito sobre a simetria associada à sequência de processamento, como é o caso de problemas com trabalho atrasado ou com janelas de execução encapsuladas, há outros

casos em que toda a simetria pode ser eliminada, como no caso de todas as tarefas serem concordantes e não haver atrasos.

Tal como já se viu anteriormente, nos casos em que $w_j \gg p$ é possível verificar se um determinado problema necessita ou não de trabalho atrasado recorrendo à análise da solução fornecida pelo problema de fluxo de custo mínimo, onde o trabalho atrasado é mínimo.

Este assunto será desenvolvido em mais detalhe no capítulo seguinte, onde se verificará que esta característica torna mesmo possível a reformulação do subproblema.

6.8 Introdução de Cortes Primais

Com vista à aceleração do processo de geração de colunas, foi possível derivar um conjunto de cortes primais que posteriormente se introduziu na formulação. Trata-se de um conjunto de cortes que aproveitam características específicas do problema e têm como restrição o facto de não poderem ser aplicados ao problema conforme ele foi definido na sua forma genérica, mas apenas a casos específicos, como se verá mais adiante.

Como os cortes introduzidos são de dois tipos, optou-se por denominar uns por cortes do tipo I e os outros por cortes do tipo II, sendo esta nomenclatura arbitrária.

6.8.1 Cortes do Tipo I

Os cortes deste tipo baseiam-se no número mínimo de colunas que devem compor a agenda de cada tarefa. Se não houver necessidade de atrasos, todas as tarefas devem ser agendadas no intervalo de tempo $\left[r_j, d_j\right]$. Logo, a tarefa terá que ser agendada num mínimo de $\left\lceil p_j \middle/ \left(d_j - r_j\right) \right\rceil$ máquinas.

Partindo daquela constatação, podem introduzir-se no modelo n restrições (uma por tarefa) a reforçar este limite inferior. As restrições terão a forma:

$$\sum_k q_{jk} x_k \geq \left\lceil \frac{p_j}{d_j - r_j} \right\rceil$$

Em que $q_{jk} = 0$ se $a_{jk} = 0$ e $q_{jk} = 1$ quando $a_{jk} \geq 0$. Relembre-se que a_{jk} representa a quantidade agendada da tarefa T_j na agenda associada à coluna k.

Caso haja a possibilidade de incluir trabalho atrasado no agendamento a desigualdade apresentada deve ser corrigida de forma a reflectir esse facto:

$$\sum_k q_{jk} x_k \geq \left\lceil \frac{p_j - L^Q}{d_j - r_j} \right\rceil$$

Recorde-se da secção 6.2.2 que L^Q é a quantidade de cada tarefa que pode ser agendada com atraso e pode ser obtida do problema de fluxo de custo mínimo.

Este tipo de restrições modifica a informação dual que é fornecida ao subproblema e, se a estrutura apresentada para o subproblema se mantiver, este deixará de identificar correctamente as colunas atractivas. Note-se que a estas novas restrições está associado uma variável dual que representa o custo ou benefício de agendar uma ou mais unidades de determinada tarefa. No entanto, os estados do subproblema apenas contêm informação sobre qual a última tarefa agendada, sendo necessário agora informação sobre quais as tarefas que têm pelo menos uma unidade agendada. Para manter uma estrutura do subproblema idêntica à original, seriam necessários um número de estados da ordem de 2^n, o que é claramente impraticável.

No capítulo seguinte será apresentado um modelo de subproblema que consegue incorporar correctamente e com aumento de eficiência este tipo de restrições, embora à custa de uma restrição ao âmbito de aplicação.

6.8.2 Cortes do Tipo II

Uma vez que os coeficientes da matriz tecnológica são valores inteiros, é possível reforçar ainda mais a formulação apresentada. As vantagens desta operação estão na obtenção de um limite inferior de melhor qualidade para a

relaxação linear. A forma de fazer este reforço é através da adição de cortes à formulação.

Estes cortes são baseados na teoria das funções super-aditivas (ver Nemhauser e Wolsey, [45]) e já foram utilizadas em geração de colunas por Vanderbeck [52] e por Alves [4].

Nemhauser e Wolsey [45], demonstraram que é possível obter desigualdades válidas para $\mathbb{Z} \cap \left\{x \in \mathbb{R}_+^n : Ax \leq b\right\}$ aplicando uma função F a todos os coeficientes de uma determinada restrição. Para isso, basta que a função F utilizada seja super-aditiva e monótona crescente em todo o seu domínio D.

Uma função é monótona crescente no seu domínio se

$$x \leq y \Rightarrow F(x) \leq F(y), \qquad \forall x, y \in D$$

Essa função será também super-aditiva se

$$F(x) + F(y) \leq F(x+y), \qquad \forall x, y, x+y \in D$$

Vanderbeck [52]apresentou uma classe de restrições válidas para

$$S = \left\{x \in \mathbb{N}^n : \sum_i a_i x_i \leq b, a \in \mathbb{N}^n, b \in \mathbb{N}, a_i \leq b\right\}$$

Essas restrições são obtidas transformando os coeficientes de uma restrição nas condições referidas através da aplicação da função

$$F^{\gamma}(z) = \max\left\{0, \left\lfloor \frac{\gamma z}{b} \right\rfloor - 1\right\}$$

com $\gamma = \{2, 3, \cdots, b\}$

No caso concreto vamos aplicar esta transformação ao seguinte conjunto de restrições:

$$\sum_k a_{jk} x_k \leq p_j$$

Se esta restrição fosse introduzida no problema primal restrito, seria equivalente a substituir as restrições que já existem, de forma

$$\sum_k a_{jk} x_k \geq p_j \,,$$

por restrições de igualdade, uma operação válida. Note-se que as restrições da formulação original até são de igualdade, tendo sido substituídas por restrições de desigualdade devido a vantagens computacionais.

No entanto, as restrições que se acrescentarão ao modelo serão as restrições obtidas por aplicação da transformação proposta por Vanderbeck:

$$\sum_k \left(\left\lfloor \frac{\gamma \cdot a_{jk}}{p_j} \right\rfloor - 1 \right) x_k \leq \gamma - 1$$

com $\gamma = \{2, 3, \cdots, p_j\}$.

Para ilustrar o funcionamento na prática deste tipo de restrições, considere-se o seguinte exemplo:

Exemplo 6.4

Considere-se uma tarefa com procura igual a 11 unidades. Considere-se ainda a seguinte restrição, que não é incluída no modelo mas dará origem aos cortes:

$$8x_1 + 5x_2 \leq 11$$

Fazendo $\gamma = 2$ e aplicando a transformação apresentada, obtém-se a seguinte restrição:

$$x_1 \leq 1$$

Esta restrição significa que uma agenda com 8 unidades desta tarefa só pode ser colocada, no máximo, ao nível 1. Repare-se que, com a restrição original, era possível colocar a variável x_1 a um nível igual a $11/8$.

Fazendo $\gamma = 3$, obter-se-ia a seguinte restrição:

$$2x_1 + x_2 \leq 2$$

Note-se que esta restrição, garante que a variável x_2 só pode ser colocada a um nível máximo de 2 e, adicionalmente, garante que se a variável x_1 for colocada a um nível de 1, a variável x_2 terá que ser colocada a 0. ■

Em geral, à medida que se vão acrescentando restrições com um valor de γ mais elevado, vão sendo eliminadas combinações de variáveis inválidas numa solução inteira, o que faz elevar o limite inferior da relaxação linear.

Verificou-se em testes computacionais preliminares que não há grandes vantagens computacionais em utilizar valores de γ muito elevados. Embora em teoria se possam utilizar valores tão grandes como p_j, isso resultaria em acrescentar um grande número de restrições ao problema, tornando a resolução de cada iteração mais lenta. Na prática verificou-se que valores de γ máximo superiores a cerca de 10% a 20% do valor de p_j médio não traziam vantagens computacionais.

Saliente-se que, tal como as restrições do tipo I apresentadas, também estas são incompatíveis com o modelo de subproblema existente, mais uma vez por ser necessário para os cálculos saber, em cada estado, todos os valores dos a_{jk} provisórios. Como já se viu, isso conduziria a um aumento exponencial e intratável do número de estados do subproblema.

6.8.2.1 Dual Feasible Functions

Recentemente, Alves [4] relacionou a teoria das funções super-aditivas com a teoria das *dual feasible functions*, obtendo reforços dos limites inferiores em esquemas de geração de colunas. Nesta secção aborda-se essa aplicação, começando por fazer uma breve apresentação das *dual feasible functions*, para depois as aplicar no âmbito do presente trabalho.

A título informativo, o termo *dual feasible functions* foi introduzido na literatura por Lueker [43] e foi usado para designar uma classe de funções que possuem a propriedade de transformar qualquer vector $x_1, x_2, \cdots, x_n$ relativo a tamanhos de itens para o problema de empacotamento, num vector $u(x_1), u(x_2), \cdots, u(x_n)$ que constitua uma solução dual válida para o correspondente problema de empacotamento não inteiro.

Diz-se que uma função $u:[0,1]\rightarrow[0,1]$ é uma *dual feasible function* se, para qualquer conjunto finito de valores do seu domínio S, o seguinte for verdadeiro:

$$\sum_{x\in S} x \leq 1 \Rightarrow \sum_{x\in S} u(x) \leq 1$$

Recentemente, Fekete e Schepers [27] utilizaram este tipo de funções para derivar limites inferiores de melhor qualidade para o problema de empacotamento. Alves [4] demonstrou que certas *dual feasible functions* são super-aditivas e monótonas crescentes e podem ser utilizadas para derivar cortes para problemas de programação inteira. Esta última conclusão pode ser utilizada para derivar cortes para o problema em análise no presente trabalho.

Considere-se novamente a seguinte classe de restrições, como já se viu, válidas para o problema em análise:

$$\sum_{k} a_{jk} x_k \leq p_j$$

Dividindo ambos os membros da desigualdade por p_j obtém-se:

$$\sum_{k} \frac{a_{jk}}{p_j} x_k \leq 1$$

Considere-se agora a seguinte *dual feasible function* apresentada em [27]:

$$u^{(\tau)}(x) = \begin{cases} x, & x(\tau+1)\in\mathbb{Z} \\ \dfrac{\lfloor(\tau+1)x\rfloor}{\tau}, & \textit{outros } x \end{cases}$$

Note-se que $x\in[0,1]$ e que $\tau\in\mathbb{N}$. É demonstrado em [4] que esta função é super-aditiva e monótona crescente. Reproduz-se de seguida essa prova a título informativo.

Proposição 6.3

A função $u^{(\tau)}(x)$ é monótona crescente e super-aditiva para $x\in[0,1]$ e para $\tau\in\mathbb{N}$.

Prova

Quanto à monotonia, a função é claramente monótona crescente, omitindo-se a demonstração. Para demonstrar que ela é super-aditiva, considere-se que x, y e $x+y$ são reais pertencentes ao intervalo $[0,1]$.

Se $(\tau+1)(x+y) \notin \mathbb{Z}$, vem:

$$u^{(\tau)}(x+y) = \frac{\lfloor(\tau+1)(x+y)\rfloor}{\tau} = \begin{cases} \dfrac{\lfloor(\tau+1)x\rfloor}{\tau} + \dfrac{\lfloor(\tau+1)y\rfloor}{\tau}, & se\ \Delta < 1 \\ \dfrac{\lfloor(\tau+1)x\rfloor}{\tau} + \dfrac{\lfloor(\tau+1)y\rfloor}{\tau} + \dfrac{1}{\tau}, & se\ \Delta \geq 1 \end{cases}$$

com $\Delta = (\tau+1)(x+y) - \left(\lfloor(\tau+1)x\rfloor + \lfloor(\tau+1)y\rfloor\right)$.

Neste caso pode verificar-se uma das seguintes situações:

1. $(\tau+1)x \in \mathbb{Z}$ e $(\tau+1)y \notin \mathbb{Z}$. Vindo,

   $u^{(\tau)}(x) + u^{(\tau)}(y) = x + \dfrac{\lfloor(\tau+1)y\rfloor}{\tau}$ e

   $\dfrac{\lfloor(\tau+1)x\rfloor}{\tau} = \dfrac{(\tau+1)x}{\tau} \geq x$.

   Logo, $u^{(\tau)}(x) + u^{(\tau)}(y) < u^{(\tau)}(x+y)$

2. $(\tau+1)x \notin \mathbb{Z}$ e $(\tau+1)y \notin \mathbb{Z}$. A ser assim, vem

   $u^{(\tau)}(x) + u^{(\tau)}(y) = \dfrac{\lfloor(\tau+1)x\rfloor}{\tau} + \dfrac{\lfloor(\tau+1)y\rfloor}{\tau}$,

   sendo que $u^{(\tau)}(x) + u^{(\tau)}(y) \leq u^{(\tau)}(x+y)$

Tem ainda que considerar-se os seguintes casos:

1. $(\tau+1)x \in \mathbb{Z}$ e $(\tau+1)y \in \mathbb{Z}$. Isto implica que $(\tau+1)(x+y) \in \mathbb{Z}$.

   Logo, $u^{(\tau)}(x) + u^{(\tau)}(y) = x + y = u^{(\tau)}(x+y)$

2. $(\tau+1)x \notin \mathbb{Z}$, $(\tau+1)y \notin \mathbb{Z}$ e $(\tau+1)(x+y) \in \mathbb{Z}$. Neste caso, implica que $\Delta = 1$, ou seja, $(\tau+1)(x+y) - \left(\lfloor(\tau+1)x\rfloor + \lfloor(\tau+1)y\rfloor\right) = 1$. Rearranjando a equação pode obter-se,

   $x + y = \dfrac{\lfloor(\tau+1)x\rfloor}{\tau} + \dfrac{\lfloor(\tau+1)y\rfloor}{\tau} + \dfrac{1-(x+y)}{\tau}$

Como $u^{(\tau)}(x)+u^{(\tau)}(y)=\frac{\lfloor(\tau+1)x\rfloor}{\tau}+\frac{\lfloor(\tau+1)y\rfloor}{\tau}$, implica que

$u^{(\tau)}(x)+u^{(\tau)}(y)=u^{(\tau)}(x+y)$ ■

Logo, a seguinte classe de cortes é válida para o problema de programação que estamos a tratar:

$$\sum_k u^{(\tau)}\left(\frac{a_{jk}}{p_j}\right)x_k \le 1 \Leftrightarrow \sum_k \frac{\left\lfloor(\tau+1)\,a_{jk}/p_j\right\rfloor}{\tau}x_k \le 1$$

A validade destes cortes decorre da definição de *dual feasible function*, uma vez que qualquer valor a_{jk}/p_j é um real pertencente ao intervalo $[0,1]$.

Também em [4] é demonstrado que estas restrições dominam sobre as restrições derivadas do trabalho de Vanderbeck, daí que devem ser utilizadas em detrimento daquelas. Também aqui se reproduz a prova:

Proposição 6.4

Para $S=\left\{x\in\mathbb{N}^n : \sum_i a_i x_i \le b, a\in\mathbb{N}^n, b\in\mathbb{N}, a_i\le b\right\}$ e para $\tau\in\mathbb{N}$ as restrições do tipo $\sum_k \frac{\left\lfloor(\tau+1)\,a_{jk}/p_j\right\rfloor}{\tau}x_k \le 1$ (1.º tipo) são equivalentes ou dominam as restrições do tipo $\sum_k\left(\left\lceil\frac{\gamma\cdot a_{jk}}{p_j}\right\rceil-1\right)x_k \le \gamma-1$ (2.º tipo), para $\tau=\gamma-1$

Prova

Por uma questão de simplificação faça-se $z_{jk}=a_{jk}/p_j$. As restrições do 2.º tipo podem ser reescritas na forma

$$\sum_k \frac{\lceil\gamma z_{jk}\rceil-1}{\gamma-1}x_k \le 1\,.$$

Para $\gamma z_{jk}\notin\mathbb{Z}$ vem $\frac{\lceil\gamma z_{jk}\rceil-1}{\gamma-1}=\frac{\lfloor\gamma z_{jk}\rfloor}{\gamma-1}=\frac{\lfloor(\tau+1)z_{jk}\rfloor}{\tau}$, sendo os coeficientes das restrições de ambos os tipos iguais.

Para $\gamma z_{jk} \in \mathbb{Z}$ os coeficientes das restrições do 1.º tipo são iguais a z_{jk}. Para as restrições do 2.º tipo vem,

$$\frac{\lceil \gamma z_{jk} \rceil - 1}{\gamma - 1} = \frac{\gamma z_{jk} - 1}{\gamma - 1} = \frac{(\tau + 1) z_{jk} - 1}{\tau} = z_{jk} + \frac{z_{jk}}{\tau} - \frac{1}{\tau}$$

que, uma vez que $z_{jk} \leq 1$, implica que

$$z_{jk} + \frac{z_{jk}}{\tau} - \frac{1}{\tau} \leq z_{jk},$$

o que por sua vez implica que as restrições do 2.º tipo são mais fracas que as do 1.º tipo. ■

Ilustre-se agora a aplicação deste tipo de cortes com um exemplo:

Exemplo 6.5

Considere-se uma tarefa com procura (p_j) igual a 9 unidades e a seguinte restrição:

$$3x_1 + 5x_2 \leq 9 \Leftrightarrow \frac{1}{3}x_1 + \frac{5}{9}x_2 \leq 1.$$

Fazendo $\tau = 1$ obtém-se a seguinte restrição:

$$x_2 \leq 1$$

que é igual à que se obteria utilizando os cortes derivados de Vanderbeck para $\gamma = 2$.

Fazendo agora $\tau = 2$ a restrição gerada será:

$$\frac{1}{6}x_1 + \frac{1}{2}x_2 \leq 1$$

Utilizando os cortes derivados de Vanderbeck com $\gamma = 3$ obter-se-ia:

$$x_2 \leq 2 \Leftrightarrow \frac{1}{2}x_2 \leq 1$$

que, como se pode ver, é mais fraca que a anterior. ■

Em testes computacionais preliminares, ao usar estes cortes, baseados em *dual feasible functions*, notou-se uma redução sensível e consistente em termos de colunas geradas e de nodos percorridos na árvore de pesquisa, quando comparados com os cortes derivados de Vanderbeck. Embora estas reduções não fossem acompanhadas de redução nos tempos de resolução, as diferenças não eram significativas. Estes resultados confirmam as observações teóricas apresentadas nesta secção.

Foram ainda consideradas no trabalho de Alves mais duas funções, mas não foi provada a dominância dos cortes resultantes sobre os cortes derivados de Vanderbeck. Por esse motivo, e porque estaríamos a falar em eventuais ganhos computacionais de carácter marginal, elas não foram utilizadas neste trabalho.

6.9 Conclusão

Neste capítulo resumiu-se o trabalho de implementação computacional que foi feito desde o momento em que se estabeleceu a formulação apresentada. Embora grande parte das técnicas sejam adaptações de técnicas genéricas já conhecidas e utilizadas na geração de colunas, a sua aplicação ao problema concreto por vezes não foi trivial.

Relativamente aos cortes apresentados na secção 6.8, no capítulo seguinte descrevem-se em detalhe as alterações que é necessário efectuar ao subproblema para que seja possível introduzir este tipo de cortes.

Capítulo 7

Caso Particular: Tarefas Ordenáveis

Neste capítulo refere-se um caso particular do problema de agendamento que tem vindo a ser tratado, que se verifica quando é possível estabelecer uma ordem de agendamento entre as tarefas, que deve ser respeitada em todas as máquinas. Começa-se por apresentar a definição de tarefas ordenáveis, passando-se depois à especificação de uma nova versão do subproblema que pretende tirar partido das características das tarefas ordenáveis. Tal como se irá ver, o facto de o problema ter tarefas ordenáveis permite a introdução desta nova versão do subproblema que, por sua vez, permite a introdução dos cortes primais apresentados no capítulo anterior.

7.1 Definições

O conjunto de tarefas de um problema designa-se por ordenável se for possível estabelecer uma ordem de execução entre todas as tarefas. Esta ordem pode ser estabelecida quando, cumulativamente, se verificarem as seguintes condições:

Condição 7.1

A solução óptima para o problema não possui trabalho atrasado.

Condição 7.2

Para quaisquer duas tarefas, é possível designar T_p (tarefa precedente) e T_s (tarefa subsequente) de forma a que, $r_p \leq r_s \wedge d_p \leq d_s$.

A prova de que a ordem das tarefas, quando as condições anteriores são verificadas, deve ser $T_p - T_s$ já foi feita na secção 6.7, quando se demonstrou a validade da Proposição 6.2.

É o facto de se poder estabelecer uma ordem de execução entre as tarefas que permite, tal como se irá ver na secção seguinte, o desenho de um novo subproblema, computacionalmente mais eficiente.

Note-se ainda que o facto de considerar as tarefas ordenáveis não é particularmente restritivo: é normal que num ambiente de produção com tarefas mais ou menos idênticas que as tarefas que estão disponíveis mais cedo tenham de ser completadas mais cedo, sendo os pesos mais ou menos constantes. Quanto à questão de não poder haver atrasos, se a unidade de produção fizer uma boa negociação de prazos, não deverá ser necessário atrasar tarefas.

7.2 Subproblema Revisto

Apresenta-se agora a versão revista do problema que pode ser utilizada com tarefas ordenáveis e permite implementar os cortes primais apresentados em

6.8. Esta versão continua a utilizar a programação dinâmica, mas os estágios foram redefinidos.

Na versão original do subproblema, cada unidade de agendamento era um estágio do problema e a decisão consistia em escolher a tarefa a colocar nessa unidade de agendamento. Na versão revista, como as tarefas, por serem ordenáveis, têm uma ordem de agendamento óptima, cada estágio corresponde à decisão de agendar uma determinada quantidade da tarefa T_j. Assim, o número de estágios é igual ao número de tarefas, n, e em cada estágio deve ser decidido qual a quantidade de tarefa a agendar, um valor entre 0 e $\min\left(p_j, T_{max} - t\right)$, em que t é o estado em que a decisão é tomada e corresponde ao número de períodos já agendados (t pode variar entre o estado inicial, em que $t = 0$, e o estado final, em que $t = T_{max}$).

A figura que se segue esquematiza a rede de transição de estados:

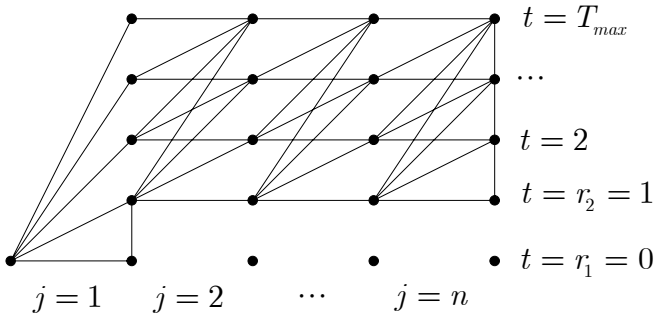


Figura 7.1 – Rede de transição de estados (revista)

A título de exemplo, com vista a ilustrar algumas características da rede e sem perda de generalidade, admita-se que $r_1 = 0$ e que $r_2 = 1$. Por uma questão de simplicidade, admita-se ainda que $p_j \geq T_{max}$ para todo o j.

Em qualquer estágio pode ser agendada uma quantidade da tarefa T_j entre 0 e $\min\left(p_j, T_{max} - t\right)$. Repare-se agora no pormenor de não ser agendado nada no estágio $j = 1$: como $r_2 = 1$, a tarefa T_2 não pode ser agendada no intervalo de tempo$[0,1]$. Devido a este facto, é necessário passar do estado $t = 0$ para o estado $t = 1$. Essa passagem pode ser feita a custo nulo agendando tempo livre. Em geral, antes de fazer a passagem de estágio j, é necessário

passar todos os estados em que $t < r_j$ para o estado $t = r_j$ (bem como, no final, passar todos os estados para $t = T_{max}$), através da inclusão de tempo livre na agenda. Em rigor, entre cada dois estágios do problema correspondentes a tarefas com datas de disponibilidade diferentes há um estágio intermédio em que estas passagens são feitas a custo nulo pela adição de tempo livre à agenda. Como é uma operação computacionalmente simples e, no fundo, não implica qualquer decisão, não vai ser incluída de forma explícita no modelo que se apresenta, embora fique representada na rede pela linhas de transição de estado verticais.

Analiticamente, as funções de transição de estado são as seguintes:

$$F_0(0) = \pi_M$$

$$F_j\left(t + a_j^t\right) = \max\left(F_{j-1}(t) + \Pi_{t,a_j^t} - a_j^{tL} w_j - a_j^{tP} p\right)$$

com $j = 1,2,\cdots,n$ e com os valores de t limitados pelos valores possíveis para a_j^t nos estágios anteriores.

O valor de a_j^t corresponde à decisão de agendar a_j^t unidades da tarefa T_j e está limitado a valores entre 0 e $\min\left(p_j, T_{max} - t\right)$.

O valor de Π_{t,a_j^t} representa a soma dos preços duais associados às restrições principais (excluindo π_M) e dos preços duais associados aos conjuntos de restrições de partição G e H, conforme definidos em 5.5.1, podendo ser calculado pela seguinte expressão:

$$\Pi_{t,a_j^t} = a_j^t \pi_j + \sum_{t'=t+1}^{t+a_j^t} \left(\sum_{l \in G_{jt'}} \mu_l + \sum_{l \in H_{jt'}} \nu_l \right)$$

Relembre-se que $G_{jt'}$ é o conjunto de restrições de partição do tipo $\leq$ que incidem sobre a tarefa j e o período de tempo t', sendo μ o respectivo vector de variáveis duais. $H_{jt'}$ e ν têm um significado equivalente para as restrições de partição do tipo $\geq$.

Adicionalmente,

$$a_j^{tL} = \begin{cases} 0, & t + a_j^t \leq d_j \\ \sum_{l=1}^{t+a_j^t-d_j} l, & t + a_j^t > d_j \end{cases}$$

fornece o factor pelo qual vai ser multiplicado w_j para determinar o custo do trabalho atrasado e

$$a_j^{tP} = \begin{cases} 0, & a_j^t = 0 \\ 1, & a_j^t > 0 \end{cases}$$

indica se houve ou não uma preparação, conforme a quantidade agendada, a_j^t, seja ou não superior a 0.

Exemplo 7.1

Relembre-se o exemplo tratado ao longo do Capítulo 5. No exemplo dado, pode verificar-se que as tarefas são concordantes e já estão ordenadas (Tabela 5.1). A certo ponto havia uma solução dual $\pi = [0.8, 4.8, 1.8, 0]$ e era necessário determinar a agenda de uma nova coluna. Utilizando o subproblema apresentado nessa secção, obteve-se a agenda 2-2-2-2-0 com um custo reduzido igual a 18.2.

Apliquem-se agora as equações de recorrência do subproblema revisto:

$F_0(0) = \pi_M = 0$

$F_1(0) = 0$	$F_1(1) = -0.2$	$F_1(2) = 0.6$	$F_1(3) = 1.4$	$F_1(4) = 2.2$	$F_1(5) = -7$
$F_2(0) = 0$	$F_2(1) = 3.8$	$F_2(2) = 8.6$	$F_2(3) = 13.4$	$F_2(4) = 18.2$	$F_2(5) = 13$
	$F_2(1) = 3.8$	$F_2(2) = 8.6$	$F_2(3) = 13.4$	$F_2(4) = 18.2$	$F_2(5) = 13$

Como se pode ver, os resultados obtidos com esta formulação são iguais aos resultados obtidos com a formulação original, devendo ser agendadas 4 unidades da tarefa 2 nos primeiros 4 períodos, ficando o último período livre.

Represente-se também a rede de transição de estados:

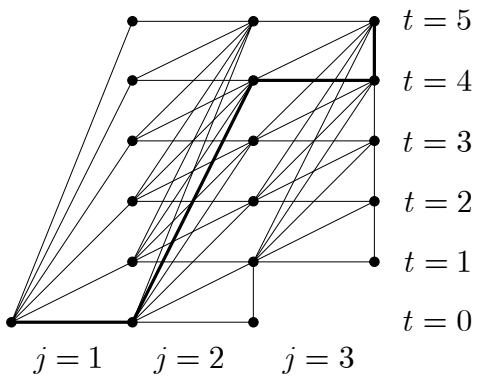


Figura 7.2 – Rede de transição de estados (Exemplo 7.1)

Na figura estão representadas todas as transições de estado possíveis, sendo o percurso correspondente à solução óptima do problema representado com uma linha mais carregada. A sequência de decisões óptimas é não agendar nada no estágio 1, agendar 4 unidades no estágio 2 (tarefa 2) e voltar a não agendar nada no estágio 3. ■

Tal como de pode verificar, este algoritmo tem $T_{\max} + 1$ estados e n estágios. No máximo, há $(T_{max} + 1)(T_{max} + 2)/2$ transições de estado em cada estágio, pelo que a complexidade do algoritmo irá ser, aproximadamente, $O\left(nT_{max}^2/2\right)$.

Uma vez apresentado este algoritmo alternativo para o subproblema, a próxima secção é dedicada à apresentação da forma de introduzir nesta versão do subproblema os cortes primais referidos em 6.8.

7.3 Introdução dos Cortes Primais

Ao introduzir no problema principal os cortes primais apresentados em 6.8, a estrutura do subproblema altera-se. Em termos de decomposição de Dantzig-Wolfe, a introdução de novos cortes no problema principal, provoca uma alteração na função objectivo do subproblema, mantendo-se a restante estrutura. Ou seja, o que muda é a forma de avaliar a atractividade das colunas que ainda não estão no problema principal.

Foram então propostos dois tipos de cortes: n cortes do tipo I, com o formato

$$\sum_k q_{jk} x_k \geq \left\lceil \frac{p_j}{d_j - r_j} \right\rceil$$

e $n\tau_{max}$ cortes do tipo II, no formato

$$\sum_k \frac{\left\lfloor (\tau+1)\, a_{jk}/p_j \right\rfloor}{\tau} x_k \leq 1$$

e com $\tau = \{1,2,\cdots,\tau_{max}\}$ (se fossem utilizados os cortes derivados de Vanderbeck, as alterações seriam semelhantes).

O custo reduzido de uma qualquer coluna, k, é dado por $c_B B^{-1} A_k - c_k$. Designando $c_B B^{-1}$ por π e incluindo as novas restrições do problema principal no vector, ele será dado por:

$$\pi = [\pi_1, \pi_2, \cdots, \pi_n, \pi_M, \mu, \nu, \rho, \sigma]$$

em que μ e ν são os vectores de variáveis duais associados às restrições de partição, ρ é um vector de n elementos associados às restrições relativas aos cortes do tipo I e σ é um vector de $n\tau_{max}$ elementos associados às restrições relativas aos cortes do tipo II. Designem-se ainda os elementos de ρ por ρ_j e os elementos de σ por $\sigma_{j\tau}$ (com j a variar entre 1 e n, e com τ a variar entre 1 e τ_{max}).

É então necessário acrescentar duas novas parcelas ao cálculo do custo reduzido das colunas geradas no subproblema, relativas aos dois tipos de restrições. Se houver restrições do tipo I, deve ser incluída a seguinte parcela:

$$\sum_j q_j \rho_j$$

Se houver restrições do tipo II, deve também ser incluída a parcela

$$\sum_j \sum_\tau \frac{\left\lfloor (\tau+1)\, a_{jk}/p_j \right\rfloor}{\tau} \sigma_{j\tau}$$

Em termos de subproblema de programação dinâmica, isto implica uma mudança na equação de recorrência, que ficará como se segue:

$$F_j\left(t+a_j^t\right)=\max\left\{F_{j-1}\left(t\right)+\Pi_{t,a_j^t}-a_j^{tL}w_j-a_j^{tP}p+a_j^{tP}\rho_j+\frac{\left\lfloor\left(\tau+1\right)a_{jk}/p_j\right\rfloor}{\tau}\sigma_{j\tau}\right\}$$

introduzindo as duas últimas parcelas os prémios ou penalidades associados às variáveis duais correspondentes às restrições do tipo I e do tipo II, respectivamente.

Desta forma é possível introduzir os cortes primais apresentados no capítulo anterior na formulação revista do subproblema. Note-se que não é obrigatório introduzir simultaneamente os dois tipos de cortes, uma vez que eles são completamente independentes. Desta forma, é possível introduzir cada tipo isoladamente ou os dois em simultâneo.

7.4 Datas de Disponibilidade e de Conclusão Comuns

Se tivermos um problema em que as datas de disponibilidade sejam todas iguais (tipicamente, $r_j=0,\forall j$) e que as datas de conclusão também sejam todas iguais $d_1=d_2=\cdots=d_n$, podem ser aplicadas regras de agendamento que, em algumas instâncias, fornecem imediatamente a solução óptima.

Proposição 7.1

O problema com datas de disponibilidade e de conclusão comuns em que não haja necessidade de trabalho atrasado é um caso particular do problema com tarefas ordenáveis.

Prova

Qualquer que seja a ordem pela qual são ordenadas as tarefas, qualquer par de tarefas respeita a Condição 7.2. ■

As regras que se apresentam nesta secção não visam substituir o processo de geração de colunas, mas apenas o recurso inicial ao problema de fluxo de custo mínimo, calculando um limite superior melhor e diminuindo assim o tamanho da árvore a pesquisar.

Note-se que este problema tem semelhanças com o problema de empacotamento (*bin-packing*), correspondendo as tarefas aos itens que necessitam de ser distribuídos por contentores (máquinas do problema). No entanto, enquanto que no problema de empacotamento se pretende minimizar o número de contentores utilizados, neste problema o número de máquinas (equivalente aos contentores) já é conhecido e não necessita de ser minimizado. Por outro lado, no problema de empacotamento, o tamanho dos itens deve ser inferior ao tamanho dos contentores, ao contrário do que se passa com a relação entre tarefas e máquinas (uma tarefa pode necessitar de mais que uma máquina para o seu agendamento integral).

7.4.1 Regra FFD (First Fit Decreasing)

Através desta regra, é agendada em primeiro lugar a maior tarefa ainda não agendada na primeira máquina com capacidade disponível. Se não for possível agendar na máquina com maior capacidade disponível a totalidade da tarefa, é agendada apenas a parte igual à capacidade disponível, indo a parte restante para a lista de tarefas ainda por agendar. A regra é aplicada repetidamente até que a lista de tarefas por agendar esteja vazia.

Como as tarefas têm datas de disponibilidade e de conclusão comuns, seja $r_j = 0, \forall j$ e $d_j = t_{max}, \forall j$. Seja p_j^R a quantidade de tarefa T_j por agendar. Inicialmente, $p_j^R = p_j, \forall j$. Denote-se a capacidade disponível em determinado processador por c_i, sendo que, inicialmente, $c_i = t_{max}, \forall i$. Utilizando esta simbologia, o pseudo código da regra FFD será:

```
begin
    while ∑_{∀j} p_j^R > 0 do
    begin
        tk := j : p_j^R = max( p_j^R );
        pc := FindFirstFit(tk);
        if pc = ∅ then pc := i : c_i = max( c_i );
        agenda(tk,pc) := min( p_tk^R , c_pc );
        p_tk^R = p_tk^R - agenda(tk,pc);
        c_pc = c_pc - agenda(tk,pc);
    end;
```

```
end;
```

No pseudo código apresentado a função *FindFirstFit(tk)* devolve o índice do primeiro processador com capacidade suficiente para processar a tarefa de índice *tk*, ou $\varnothing$ se não houver nenhum processador nessas condições.

Esta regra conduz a uma ocupação arbitrária dos processadores que têm capacidade disponível para processar determinada tarefa.

Como facilmente se depreende, para o agendamento se poder fazer sem atrasos, é necessário e suficiente que se verifique a condição

$$\sum_j p_j^R \leq \sum_i c_i$$

Ou seja, a capacidade disponível nos processadores não pode ser inferior às exigências de processamento das tarefas.

7.4.2 Regra LFD (Largest Fit Decreasing)

A regra LFD é em tudo igual à regra FFD excepto no facto de o agendamento ser sempre feito no processador com maior disponibilidade, seja este máximo inferior ou superior à quantidade que se pretende agendar.

Utilizando a simbologia proposta para a regra FFD, o pseudo código da regra LFD, é o que se segue:

```
begin
    while ∑_{∀j} p_j^R  > 0 do
    begin
        tk := j : p_j^R = max( p_j^R );
        pc := i : c_i = max( c_i );
        agenda(tk,pc) := min( p_tk^R , c_pc );
        p_tk^R = p_tk^R - agenda(tk,pc);
        c_pc = c_pc - agenda(tk,pc);
    end;
end;
```

Esta regra conduz a uma ocupação homogénea dos processadores, uma vez que o processador com mais disponibilidade é sempre o primeiro a ocupar. Tendencialmente, as tarefas mais pequenas são agendadas no final, quando os

processadores já têm pouca disponibilidade, o que conduzirá a menor fragmentação das tarefas pelos processadores, o que é desejável em termos de custos de preparação.

7.4.3 Regra BFD (Best Fit Decreasing)

A regra BFD também é semelhante às anteriores, no entanto, a escolha do processador é feita de modo a agendar as tarefas nos processadores que tenham uma capacidade disponível não inferior à quantidade a agendar, mas o mais próximo possível deste valor. Se não existir nenhum processador com a capacidade necessária, a tarefa é agendada no de maior capacidade, indo a quantidade restante para a lista de tarefas a agendar.

Em termos de pseudo código fica:

```
begin
    while ∑_{∀j} p_j^R > 0 do
    begin
        tk := j : p_j^R = max( p_j^R );
        pc := i : c_i - p_tk^R = min( c_i - p_tk^R );
        if c_pc - p_tk^R < 0 then pc := i : c_i = max( c_i );
        agenda(tk,pc) := min( p_tk^R , c_pc );
        p_tk^R = p_tk^R - agenda(tk,pc);
        c_pc = c_pc - agenda(tk,pc);
    end;
end;
```

Como se pode verificar a escolha do processador recai sobre aquele que tenha a capacidade disponível mais ajustada à tarefa a agendar ou sobre o de maior capacidade se nenhum existir com capacidade suficiente.

A ideia subjacente a esta heurística é a de ir *fechando* processadores com tarefas que completem a sua capacidade disponível, garantindo deste modo que nesse processador não ocorrerá mais nenhuma preparação.

7.4.4 Comparação das Heurísticas

Em testes preliminares realizados com o objectivo de verificar qual destas três heurísticas fornecia as melhores soluções, verificou-se que não há entre elas diferenças significativas, produzindo todas elas, quase sempre, soluções idênticas. No entanto, havendo que escolher uma delas, os resultados mostraram que a heurística BFD fornecia as melhores soluções, quase sempre idênticas às da heurística FFD, e que a heurística LFD fornecia as piores soluções. As soluções fornecidas pela heurística BFD estavam, em média, cerca de 23% acima do valor óptimo.

Na prática, como os tempos de execução de todas elas são bastante aceitáveis e apenas é necessário executá-las uma única vez, em implementações computacionais podem ser executadas as três, aproveitando-se a melhor solução.

7.5 Conclusão

Neste capítulo foram apresentadas metodologias que, embora sendo aplicáveis apenas a casos específicos do problema principal que foi tratado, conseguem melhorias computacionais significativas nos casos em que se aplicam (os resultados serão apresentados em detalhe na secção seguinte).

O motivo pelo qual se dedicou algum tempo a estes casos mais simples prende-se sobretudo com a elevada complexidade do problema estudado. Por outro lado, o desenvolvimento de metodologias para acelerar a resolução de casos particulares em nada prejudica a aplicação das metodologias genéricas propostas anteriormente.

Capítulo 8

Resultados Computacionais

Nesta secção são apresentados os resultados computacionais obtidos na implementação computacional feita a partir das ideias apresentadas nos capítulos anteriores. Numa primeira parte são feitas algumas considerações genéricas às metodologias utilizadas para produzir estes resultados. Nas três secções seguintes são apresentados resultados para os três tipos de problemas abordados, nomeadamente, problemas com janelas genéricas, problemas com janelas concordantes e problemas com janelas comuns.

8.1 Introdução

A implementação computacional dos algoritmos apresentados foi feita em C++, utilizando o compilador do Microsoft Visual Studio, versão 6.0. O programa foi desenvolvido como uma aplicação de consola (linha de comandos) e não tem qualquer interacção com o utilizador durante o funcionamento. A interface consiste na indicação, na linha de comando, de um ficheiro com os dados do problema e de um ficheiro onde se deseja colocar a informação sobre a solução.

Para resolver os problemas de programação linear é utilizada a biblioteca de funções CPLEX (versão 8.1.1) na qual se recorre à biblioteca de funções para C padrão. Trata-se de um pacote de software comercial, largamente utilizado em aplicações industriais de optimização.

Os testes foram executados numa máquina equipada com um processador Pentium IV a 2.2 GHz e com 768 MBytes de memória RAM, a correr o sistema operativo Microsoft Windows XP com SP2. Durante os testes são desactivados todos os serviços não essenciais (por exemplo, antivírus, correio electrónico, serviços de rede) para que os tempos de processamento não sejam significativamente afectados por tempos gastos com aqueles serviços.

8.2 Problemas com Janelas Genéricas

Neste tipo de problemas, com janelas de execução genéricas, não é imposta qualquer restrição sobre os valores das datas de disponibilidade ou de conclusão das tarefas, à excepção de que o problema tem que ter soluções admissíveis.

Em problemas destes não é possível utilizar os cortes apresentados em 6.8 e, em consequência, a versão revista do subproblema apresentada no Capítulo 7, uma vez que estas metodologias apenas são aplicáveis a casos particulares deste problema, conforme descrito quando foram apresentadas.

Para realizar os testes foram geradas aleatoriamente instâncias com 10, 20, 30, 40 e 50 máquinas e com horizontes de agendamento de 21 e de 42 períodos. Foram ainda utilizados 4 modelos para a geração dos tempos de processamento: tarefas pequenas (duração entre 1 e 30), tarefas médias (durações entre 31 e 120), tarefas grandes (durações entre 121 e 240) e ainda um modelo com tamanhos arbitrários entre 1 e 240 unidades.

Os tempos de processamento foram gerados utilizando a distribuição uniforme, fazendo variar o valor mínimo e máximo de acordo com o tipo de tarefas que se pretendia gerar.

As datas de disponibilidade foram geradas utilizando também uma distribuição uniforme com mínimo igual a 0 e máximo igual a metade do horizonte de agendamento. Se a instância se revelasse impossível (o procedimento não garante a geração de instâncias solúveis) era substituída por uma outra. Para que o agendamento pudesse começar no instante 0, caso não houvesse nenhuma tarefa com data de disponibilidade 0, a data de disponibilidade da maior tarefa era colocada a 0.

As datas de conclusão foram geradas através de uma distribuição uniforme com mínimo igual a d_{min} e máximo igual ao horizonte de agendamento, sendo d_{min} calculado através da seguinte expressão:

$$d_{min} = \left\lceil r_j + \frac{p_j}{m} \right\rceil$$

Note-se que esta expressão garantiria, se não houvesse outras tarefas a necessitar de agendamento no mesmo intervalo, a não necessidade de recurso a trabalho atrasado.

Para avaliar a qualidade das soluções utilizou-se um custo de preparação igual a 1 unidade e um custo de atraso igual a 32 unidades, constante para todas as tarefas.

Esta metodologia daria origem a 40 grupos de problemas (5 valores para m, 2 valores para t_{max} e 4 tipos de tarefas), no entanto, como havia grupos evidentemente triviais (o algoritmo resolveu consistentemente instâncias de

maior complexidade) e grupos demasiado complexos (o algoritmo não resolveu problemas de complexidade inferior), apenas foram geradas instâncias para grupos com interesse. Para cada um destes grupos foram geradas 10 instâncias, tendo sido resolvidas, no total, 230 instâncias do problema.

Nas tabelas que se seguem são indicadas as principais características das instâncias utilizadas nos testes, bem como os indicadores de desempenho computacional normalmente encontrados na literatura:

m: número de máquinas;

t: número de períodos de agendamento (horizonte);

n: número médio efectivo de tarefas por instância, calculado como a média aritmética do número de tarefas presentes nas 10 instâncias geradas;

S: número de instâncias, das 10 resolvidas, para as quais foi encontrada a solução óptima dentro de um período de tempo aceitável;

C: número médio de colunas utilizadas nas instâncias resolvidas até ao óptimo;

N: número médio de nodos percorridos na árvore de pesquisa durante a resolução dos problemas resolvidos até ao óptimo;

D: profundidade média das árvores de pesquisa dos problemas resolvidos até ao óptimo, sendo calculada como a média aritmética das profundidades máximas atingidas em cada instância considerada;

ST: tempo médio decorrido na pesquisa da solução óptima para os problemas resolvidos até ao óptimo.

Considerou-se como *tempo aceitável de resolução* o intervalo de 15 minutos (900 segundos). No entanto, embora alguns problemas não sejam resolvidos até ao óptimo naquele intervalo de tempo, quando se interrompe o processo já há normalmente soluções de boa qualidade.

Cada tabela que se apresenta de seguida respeita a cada um dos tamanhos das tarefas utilizados. Este parâmetro está directamente relacionado com o

número de tarefas por instância e, como se vai ver, tem grande influência sobre a complexidade das instâncias.

8.2.1 Tarefas Pequenas

Tal como se pode verificar na Tabela 8.1, com este tipo de tarefas ($1 \leq p_j \leq 30$) apenas foram resolvidos de forma consistente (8 em 10) problemas com 10 máquinas e 21 períodos de agendamento. Todas as outras combinações de número de máquinas e número de períodos de agendamento resultam em problemas demasiado complexos que não são resolvidos em tempo aceitável.

t	m	n	S	C	N	D	ST
21	10	11.4	8	3684	8636	24	155.0
	20	26.8	1	748	7533	24	109.0
42	10	25.8	0	-	-	-	-

Tabela 8.1 - Resultados para janelas genéricas e tarefas pequenas

8.2.2 Tarefas Médias

Ao usar tarefas de tamanho superior, reduz-se o número de tarefas por instância e a complexidade é menor. Tal como se pode verificar na Tabela 8.2, foi possível resolver instâncias com 40 máquinas e 21 períodos de agendamento ou instâncias com 10 máquinas e 42 períodos de agendamento.

t	m	n	S	C	N	D	ST
21	20	4.6	10	151	706	14	2.8
	30	8.2	10	285	6021	20	78.1
	40	10.0	9	428	5991	27	118.0
	50	13.6	2	125	41	12	0.5
42	10	5.1	9	671	5289	24	131.7
	20	10.4	1	212	91	18	2.0

Tabela 8.2 - Resultados para janelas genéricas e tarefas médias

Se forem comparadas as instâncias com 21 períodos de agendamento e 40 máquinas (840 períodos de agendamento totais) e as instâncias de 42 períodos

de agendamento e 20 máquinas (também 840 períodos de agendamento totais) que à primeira vista podem julgar-se semelhantes, salta à vista a disparidade de resultados computacionais: das primeiras, são resolvidas 9 das 10 instâncias, e das segundas, apenas é resolvida 1 instância.

Este facto decorre da forma como é efectuada a partição, uma vez que o número de fluxos sobre os quais podem incidir restrições de partição é igual a nt_{max}, dependente do horizonte de planeamento e independente do número de máquinas. Logo, instâncias com horizontes de planeamento curtos e um maior número de máquinas são melhor resolvidas do que instâncias com menos máquinas e horizontes de planeamento mais longos, mesmo que o número de tarefas e o número total de períodos a programar sejam idênticos.

8.2.3 Tarefas Grandes

Ao passar o tamanho das tarefas para um valor ainda mais elevado, o número destas diminui e torna-se possível programar mais máquinas e horizontes de agendamento maiores. Na Tabela 8.3 são apresentados os resultados computacionais obtidos para este tipo de instâncias.

t	m	n	S	C	N	D	ST
21	20	2.0	10	30	11	5	0.2
	30	3.1	10	86	338	13	2.9
	40	4.3	10	78	46	11	0.6
	50	5.2	10	183	786	20	11.6
42	10	2.0	10	80	34	10	1.7
	20	4.3	7	199	185	21	5.7
	30	6.6	2	166	185	22	6.5

Tabela 8.3 - Resultados para janelas genéricas e tarefas grandes

As instâncias com horizonte de agendamento de 21 períodos têm um número de tarefas particularmente reduzido e, por esse facto, podem ser consideradas triviais, sendo a sua resolução bastante rápida e 100% bem sucedida nas 40 instâncias testadas.

Quando se aumentou o horizonte de planeamento para 42 períodos ainda foi possível resolver a maioria das instâncias com 20 máquinas.

8.2.4 Tarefas de Tamanho Arbitrário

Verificou-se que este tipo de instâncias tem uma complexidade semelhante às instâncias com tarefas grandes, apenas um pouco superior, visto que o número médio de tarefas também é superior. A Tabela 8.4 resume os resultados obtidos.

t	m	n	S	C	N	D	ST
21	20	2.8	10	53	46	8	0.1
	30	5.8	10	462	10947	19	103.6
	40	7.0	9	297	3849	27	13.9
	50	8.7	8	556	14484	24	58.9
42	10	3.3	10	113	146	13	0.9
	20	6.6	8	2319	12040	28	176.5
	30	11.1	1	177	101	11	0.0

Tabela 8.4 - Resultados para janelas genéricas e tarefas de tamanho arbitrário

Tal como se verifica, os resultados são semelhantes aos obtidos com tarefas grandes, embora ligeiramente piores.

8.2.5 Considerações Gerais

Tal como se pode verificar, a implementação computacional realizada consegue resolver em tempos aceitáveis instâncias com 40 máquinas e horizonte de agendamento de 21 períodos (num total de $40 \times 21 = 840$ períodos de agendamento) se as instâncias forem compostas por tarefas de dimensão média ou superior (o que significa problemas com cerca de 10 tarefas). Quando se passa para 42 períodos de agendamento ainda são resolvidas algumas instâncias de 30 máquinas com um número de tarefas até aproximadamente 6. Note-se ainda na resolução de algumas instâncias com 50 máquinas e 21 períodos, desde que o número de tarefas fique abaixo de 10.

Atente-se também nos tempos médios de resolução, que são muito inferiores, em média, aos 15 minutos limite, nas instâncias em que é encontrada a solução óptima.

Dos resultados obtidos, pode inferir-se que o tempo de resolução está muito ligado ao número de tarefas que compõem a instância. Esta observação faz todo o sentido, uma vez que o universo de colunas possíveis para o problema cresce exponencialmente com o número de tarefas. Um maior número de tarefas também dá origem a fenómenos de simetria em maior escala.

8.3 Problemas com Tarefas Ordenáveis

A verificação das condições que permitem considerar as tarefas ordenáveis, permite utilizar os cortes apresentados em 6.8 na resolução do problema e aproveitar os ganhos computacionais daí decorrentes.

Para testar o impacto destes cortes e da formulação revista do subproblema a eles associada foi gerado aleatoriamente um conjunto de instâncias com aquelas características. Tal como na secção anterior, testaram-se instâncias com diversas dimensões para verificar quais os limites da formulação.

Em termos de horizonte de agendamento, foram testados ambientes de 21 e de 42 períodos. Estes horizontes foram combinados com conjuntos de 10, 20, 30, 40, 50 e 60 máquinas e com os 4 tamanhos de tarefas utilizados na secção anterior. Também aqui não foram testadas combinações comprovadamente triviais ou demasiado complexas. Na totalidade, foram resolvidas 170 instâncias do problema.

Quanto à metodologia de geração das instâncias, utilizou-se o procedimento da secção anterior para gerar os valores de p_j. Os valores das datas de disponibilidade foram gerados da seguinte forma: para a primeira tarefa fez-se $r_j = 0$; para as tarefas seguintes utilizou-se a distribuição uniforme para gerar o valor de $r_j - r_{j-1}$ entre 0 e o resultado arredondado ao inteiro mais próximo da expressão

$$\frac{p_{min} + p_{max}}{2m}.$$

Para determinar as datas de conclusão começa-se por calcular o valor mínimo para d_j, através da expressão

$$d_j^{min} = \max\left\{\left\lceil \frac{1}{m}\sum_{l=1}^{j-1} p_l \right\rceil, r_j + \left\lceil \frac{p_j}{m} \right\rceil\right\}$$

A este valor é somado um factor obtido da distribuição uniforme entre 0 e $\left(p_{min} + p_{max}\right)\big/2m$ arredondado ao inteiro mais próximo.

Com esta forma de cálculo pretende-se evitar que tenha que haver recurso a trabalho atrasado. No entanto, como há uma pequena probabilidade de serem geradas instâncias em que haja necessidade de trabalho atrasado, sempre que isso aconteça a instância é substituída por outra.

Tal como na secção anterior foi usada uma penalização por cada preparação igual a 1 unidade e valores de w_j iguais a 32 unidades (irrelevante, uma vez que não são necessários atrasos).

Os resultados são apresentados nas secções seguintes seguindo o esquema utilizado na apresentação dos resultados para instâncias com janelas genéricas.

8.3.1 Tarefas Pequenas

Tal como aconteceu com as tarefas com janelas ordenáveis, quando as tarefas são em grande número, não é possível resolver instâncias com muitas máquinas ou com horizontes de agendamento longos. Tal como se pode verificar na Tabela 8.5 apenas foi possível resolver com consistência instâncias de 10 máquinas com um horizonte de planeamento de 21 períodos.

t	m	n	S	C	N	D	ST
21	10	13.2	10	419	458	17	3.7
	20	25.9	2	186	30	13	0.0
42	10	26.4	1	3852	8235	38	510.0

Tabela 8.5 - Resultados para janelas ordenáveis e tarefas pequenas

8.3.2 Tarefas Médias

Com este tipo de tarefas e um horizonte de agendamento de 21 períodos foi possível resolver consistentemente instâncias com 50 máquinas. Mesmo com 60 máquinas, a maioria das instâncias também foi resolvida. A Tabela 8.6 resume os resultados obtidos.

t	m	n	S	C	N	D	ST
21	20	5.1	10	85	192	18	1.8
	30	8.8	10	212	1537	21	20.9
	40	10.6	9	330	2481	27	37.1
	50	13.6	8	253	2839	23	47.9
	60	15.7	6	233	556	22	9.8
42	10	4.9	10	128	167	20	3.8
	20	10.3	5	995	2906	44	250.4

Tabela 8.6 - Resultados para janelas ordenáveis e tarefas médias

Quando se utilizou ambientes com um horizonte de agendamento de 42 períodos já só foi possível resolver metade das instâncias com 20 máquinas.

Mais uma vez pode notar-se que, quanto à relação número de máquinas versus horizonte de agendamento, para um número de tarefas e um número total de períodos de agendamento idênticos, as instâncias com mais máquinas e horizontes mais curtos são claramente favorecidas.

8.3.3 Tarefas Grandes

A Tabela 8.7 resume o conjunto de resultados obtidos para este tipo de instâncias.

t	m	n	S	C	N	D	ST
21	60	6.4	10	117	376	16	12.2
42	20	4.5	10	86	72	23	4.6
	30	6.6	7	212	805	39	62.1
	40	8.8	4	463	2164	42	249.3

Tabela 8.7 - Resultados para janelas ordenáveis e tarefas grandes

Embora a totalidade dos problemas com horizonte de agendamento de 21 períodos e 60 máquinas tenham sido resolvidos, ao olhar para os resultados da Tabela 8.8 (tarefas de tamanho arbitrário) não é de prever que se consigam resolver consistentemente instâncias muito maiores.

Quando se utiliza um horizonte de 42 períodos, ainda se consegue resolver a maioria dos problemas com 30 máquinas.

8.3.4 Tarefas de Tamanho Arbitrário

Com este tipo de tarefas, os resultados são ligeiramente piores que os obtidos com tarefas grandes. A Tabela 8.8 expressa esses resultados.

t	m	n	S	C	N	D	ST
21	60	11.5	6	268	2440	24	17.3
42	20	5.8	9	233	4437	30	48.9
	30	10.0	1	1237	10735	45	411.0

Tabela 8.8 - Resultados para janelas ordenáveis e tarefas de tamanho arbitrário

8.3.5 Considerações Gerais

Tal como era esperado o algoritmo utilizado para tarefas ordenáveis tem uma performance superior ao algoritmo genérico. Por exemplo, nas instâncias de 50 máquinas e 21 períodos com tarefas de tamanho médio, onde o algoritmo genérico apenas resolveu 2 problemas, o algoritmo para tarefas ordenáveis ainda conseguiu resolver 8 instâncias em tempo aceitável. Mesmo com 60 máquinas e 21 períodos (que nem sequer foi testado com o algoritmo genérico) ainda há um conjunto significativo de instâncias resolvidas, mesmo com um número de tarefas elevado (com 15.7 tarefas, em média, ainda foram resolvidos 6 em 10 problemas).

Como é possível argumentar que o tipo de instâncias utilizadas para testar o algoritmo para tarefas ordenáveis pode ter uma complexidade menor que as instâncias com janelas genéricas, na secção seguinte apresentam-se os resultados obtidos com o algoritmo genérico nas instâncias testadas nesta secção.

Desta forma será possível testar a performance dos dois algoritmos sem a influência do tipo de instância utilizada.

8.3.6 Comparação com o Algoritmo Genérico

Para testar de forma exacta a diferença de performance entre o algoritmo específico para tarefas ordenáveis e o algoritmo genérico, utilizou-se o algoritmo genérico para resolver as instâncias utilizadas para testar o algoritmo para tarefas ordenáveis. Nas tabelas que se seguem, apresenta-se o diferencial dos resultados: cada valor das tabelas, formatadas à semelhança das que têm sido apresentadas ao longo do capítulo, corresponde à diferença média, nas instâncias que ambos os algoritmos conseguiram resolver até ao óptimo, entre o valor da medida obtido no algoritmo para tarefas ordenáveis e o valor da medida obtida, nas mesmas instâncias, com o algoritmo genérico (excepto nas duas primeiras colunas, que caracterizam as instâncias).

t	m	S	C	N	D	ST
21	10	10	-895	-3083	-4	-42.7
	20	2	-61	-38	-5	0.0
42	10	0	-	-	-	-

Tabela 8.9 - Diferencial de resultados para tarefas pequenas

t	m	S	C	N	D	ST
21	20	10	-110	-4015	-4	-13.2
	30	10	-115	-8780	-4	-42.0
	40	7	-101	-1075	0	6.6
	50	8	-112	-6597	-4	-9.0
	60	6	-96	-12376	-6	-124.2
42	10	10	-462	-1816	-8	-26.7
	20	3	-883	-3059	0	-144.0

Tabela 8.10 - Diferencial de resultados para tarefas médias

t	m	S	C	N	D	ST
21	60	10	-65	-701	-6	9.4
42	20	10	-312	-4763	-5	-49.4
	30	6	-337	-7698	2	-16.7
	40	2	-13	-492	9	146.5

Tabela 8.11 - Diferencial de resultados para tarefas grandes

t	m	S	C	N	D	ST
21	60	6	-81	-7476	-3	-26.3
42	20	8	-290	-5174	-1	-52.6
	30	1	-629	-6234	5	-58.0

Tabela 8.12 - Diferencial de resultados para tarefas de tamanho arbitrário

Na coluna **S** aparece o número de problemas que ambos os algoritmos optimizaram enquanto que nas outras é mostrada a diferença média, entre os resultados do algoritmo especializado e os resultados do algoritmo genérico. No global, o número de problemas resolvidos por um e por outro algoritmo é idêntico, notando-se apenas uma pequena diferença favorável ao algoritmo especializado (resolveu apenas mais 6 problemas que o algoritmo genérico).

Ao olhar para as diferenças de colunas geradas e de nodos pesquisados, verificam-se reduções muito significativas e consistentes quando se utiliza o algoritmo especializado. Globalmente, para os problemas resolvidos por ambos os algoritmos, o algoritmo especializado em tarefas ordenáveis necessitou de menos 54.6% de colunas e de menos 81.8% de nodos. Também a profundidade máxima da pesquisa teve uma redução média de 13.0%.

Quanto a tempo gasto na optimização, o algoritmo especializado gastou, no global, menos 54.6% de tempo.

Estes resultados devem-se às melhorias introduzidas no subproblema, que é computacionalmente mais eficiente, e à introdução dos cortes primais já referidos.

8.4 Problemas com Janelas Comuns

O algoritmo de resolução de problemas com janelas comuns é idêntico ao algoritmo utilizado para as tarefas ordenáveis, sendo a única diferença o modo de geração de soluções iniciais: nos problemas com janelas comuns é utilizada a heurística BFD (apresentada em 7.4.3) em vez do problema de fluxo de custo mínimo.

A geração deste tipo de instâncias é muito mais simples do que a geração de instâncias para tarefas ordenáveis ou genéricas, uma vez que apenas é necessário gerar o tamanho das tarefas.

Para fazer a geração do tamanho das tarefas utilizou-se a metodologia já apresentada para os casos anteriores, sendo utilizada a distribuição uniforme para gerar os 4 tipos de tarefas.

As datas de disponibilidade de todas as tarefas foram colocadas a 0 e as datas de conclusão igualaram-se ao horizonte de agendamento. As penalidades utilizados foram iguais aos casos anteriores.

8.4.1 Tarefas Pequenas

A primeira tabela que se apresenta (Tabela 8.13) contém os resultados da resolução de instâncias compostas por tarefas pequenas.

t	m	n	S	B&P	C	N	D	ST
21	20	25.4	9	1	22527	15904	43	751
	30	40.3	6	0	-	-	-	-
	40	54.5	8	0	-	-	-	-
	60	81.8	8	0	-	-	-	-
42	20	53.0	9	3	313	48	23	1
	30	80.2	10	1	312	53	26	9
	40	109.8	10	2	1809	172	85	28
	60	162.4	8	1	3565	277	138	104

Tabela 8.13 - Resultados para janelas comuns e tarefas pequenas

Como se pode verificar, mesmo em instâncias de grande dimensão, a solução óptima é quase sempre encontrada. Após análise dos dados concluiu-se que a

maior parte das instâncias optimizadas nem sequer entravam no processo de partição. Isto aconteceu porque a solução encontrada pela heurística BFD era quase sempre coincidente com o limite inferior calculado e, por isso, óptima.

Para evidenciar este resultado foi acrescentada à tabela usual mais uma coluna, rotulada como **B&P**, que indica o número de instâncias, das que foram optimizadas, que entraram no processo de partição e avaliação.

Como se pode ver, o número de instâncias que foram optimizadas na fase de partição e avaliação é muito reduzido, quando comparado com o número de instâncias para as quais foi calculada uma solução óptima. Isto significa que a maior parte das soluções iniciais são óptimas.

8.4.2 Tarefas Médias

Quando se utilizaram tarefas de dimensão média, os resultados foram muito semelhantes aos obtidos com tarefas pequenas. Atente-se na Tabela 8.14 onde são resumidos esses resultados.

t	m	n	S	B&P	C	N	D	ST
21	20	4.9	10	1	198	107	17	3.0
	30	7.5	10	0	-	-	-	-
	40	11.4	10	1	3165	13357	45	625.0
	60	15.8	8	0	-	-	-	-
42	20	10.4	8	2	1650	1417	25	207.0
	30	16.1	9	0	-	-	-	-
	40	22.0	5	0	-	-	-	-
	60	33.1	6	0	-	-	-	-

Tabela 8.14 - Resultados para janelas comuns e tarefas médias

8.4.3 Tarefas Grandes

Apenas foram geradas instâncias com tarefas grandes para horizontes de 42 períodos e 40 ou 60 máquinas. Tal como se pode verificar na Tabela 8.15 a heurística BFD continua a fornecer soluções óptimas, não sendo quase nunca necessário recorrer à partição e avaliação para as calcular.

t	m	n	S	B&P	C	N	D	ST
42	40	8.6	8	0	-	-	-	-
	60	13.7	10	0	-	-	-	-

Tabela 8.15 - Resultados para janelas comuns e tarefas grandes

Eventualmente, instâncias com maior número de máquinas seriam igualmente resolvidas. No entanto, tal não foi testado uma vez que se estaria apenas a testar a heurística BFD, uma vez que o algoritmo de partição e avaliação não consegue tratar, em tempo aceitável, instâncias com estas dimensões.

8.4.4 Tarefas de Tamanho Arbitrário

O que foi dito relativamente às tarefas de grande dimensão é aplicável também aqui. A Tabela 8.16 evidencia os resultados obtidos para este tipo de tarefas.

t	m	n	S	B&P	C	N	D	ST
42	40	13.2	9	1	2277	1501	27	407.0
	60	20.4	8	0	-	-	-	-

Tabela 8.16 - Resultados para janelas comuns e tarefas de dimensão arbitrária

8.4.5 Considerações Gerais

Uma vez que a heurística BFD forneceu soluções iniciais óptimas para grande parte dos problemas, não foi possível testar os limites do algoritmo de partição e avaliação. No entanto, uma vez que ele é igual ao algoritmo utilizado com as tarefas ordenáveis, o desempenho não deve ser muito diferente.

Note-se que estas instâncias, no caso em que a heurística BFD não encontra a solução óptima, deverão ser mais complexas que instâncias semelhantes com janelas ordenáveis. De facto, o conjunto de soluções válidas para uma instância com tarefas ordenáveis, está contido no conjunto de soluções válidas para a mesma instância se as datas de disponibilidade forem igualadas a 0 e as datas de conclusão igualadas ao horizonte de planeamento.

8.5 Conclusão

Dos resultados apresentados, pode concluir-se que os algoritmos desenvolvidos são bastante sensíveis ao número de tarefas, cuja elevação provoca grandes aumentos na carga computacional, e, em menor escala, do tamanho do horizonte de agendamento.

Estes resultados eram os esperados, uma vez que ao longo do texto foi visto que a complexidade do algoritmo dependia destes dois parâmetros. O aumento do número de tarefas faz aumentar, principalmente o número de colunas simétricas geradas, enquanto que o aumento do horizonte de planeamento aumenta significativamente o número de variáveis sobre as quais incide a partição.

Sem se poder afirmar que o modelo consegue tratar problemas de dimensão muito grande, pode afirmar-se que trata problemas com uma dimensão razoável. Nos casos específicos em que foi possível especializar o algoritmo, os resultados foram ainda melhores.

Por outro lado, uma das vantagens da geração de colunas é, nos casos em que o problema não é optimizado, dispor sempre de soluções de recurso de boa qualidade, ou pelo menos, de qualidade mensurável, uma vez que se conhecem limites inferiores para o valor da solução óptima.

Capítulo 9

Conclusões e Trabalho Futuro

Neste trabalho foi apresentado um algoritmo que resolve de forma exacta, com recurso aos métodos de geração diferida de colunas e de partição e avaliação, problemas de agendamento de máquinas paralelas idênticas com tarefas maleáveis e datas de disponibilidade arbitrárias, sendo o objectivo a minimização de uma função do trabalho atrasado e da quantidade de preparações. Tanto quanto é do nosso conhecimento, não há referências a trabalhos semelhantes na literatura sobre agendamento.

Embora a abordagem que se fez esteja orientada para o problema específico, alguns conceitos, tal como a definição das variáveis do modelo matemático ou o esquema de partição, depois das necessárias adaptações, podem ser utilizados em outros modelos semelhantes.

Quando se fala em partição e geração diferida de colunas, o algoritmo não é composto apenas pelo problema primal restrito, o subproblema e a articulação entre eles, mas por um conjunto vasto de técnicas auxiliares e pequenos melhorias que contribuem de forma decisiva para os resultados finais.

De facto, para fazer com que um algoritmo deste tipo funcione, é necessário desenvolver um conjunto de heurísticas, tais como, heurísticas de pesquisa local, de arredondamento de soluções, de escolha de variáveis a particionar, etc. Também é normalmente necessário adicionar restrições extra (cortes) que fortalecem a formulação ou retirar outras que a enfraquecem, por exemplo. Ou seja, há um vasto conjunto de técnicas que é necessário juntar para que o resultado final seja aceitável.

Como trabalho futuro, há um conjunto de melhorias e extensões que, na nossa opinião, têm interesse científico:

- Uma grande melhoria no modelo apresentado seria a derivação de um conjunto de regras de agendamento, a passar para o subproblema, que eliminassem grande parte da simetria associada à geração de colunas do modelo genérico;
- Outra ideia que não foi possível concretizar consistia na derivação de cortes primais para o algoritmo genérico (coisa que foi possível fazer para o algoritmo especializado). Acreditamos que estes cortes poderiam trazer um acréscimo de desempenho significativo, à semelhança do que aconteceu com os algoritmos especializados;
- Ampliação do modelo para que, além de máquinas paralelas idênticas, possa tratar problemas com máquinas paralelas uniformes ou não relacionadas. Isto envolveria, entre outras coisas, a criação de vários subproblemas, um para cada tipo de máquina existente no problema. Seria também necessário um esquema de "tradução" das datas de disponibilidade e de conclusão atendendo à velocidade dos vários tipos de máquinas. Embora, à partida, não pareçam alterações de vulto, iriam contribuir para um aumento dos tempos de computação, pois em cada nodo seria necessário avaliar vários subproblemas;

- Utilização de tempos de preparação dependentes da sequência. Esta alteração permitiria resolver problemas em que os tempos de preparação não se pudessem considerar constantes, mas envolveria grandes alterações no modelo matemático formulado, mesmo limitanto o valor dos tempos de preparação a inteiros;
- Inclusão no modelo de paragens programadas (para manutenção, por exemplo) das máquinas. Em princípio levaria à criação de vários subproblemas, um para as máquinas sempre disponíveis e outros para cada máquina ou conjunto de máquinas com uma agenda de paragens programadas diferente. Esta aproximação também serve para o caso de as máquinas terem datas de disponibilidade associadas;

Bibliografia

[1] Ahuja, R., Magnanti, T., Orlin, J. – *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, New Jersey, 1993.

[2] Akker, J. M., Hoogeveen, J. A., Velde, S. L. – *Parallel Machine Scheduling by Column Generation*, Report COSOR, Dept. Math. Comp. Sci. – Eindhoven University of Technology, 1995.

[3] Akker, J. M., Hurkens, C. A. J., Savelsbergh, M. W. P. – Time-Indexed Formulations for Machine Scheduling: Column Generation, INFORMS Journal on Computing, 12, 111-124, 2000.

[4] Alves, Cláudio M. M. – *Cutting and Packing: Problems, Models and Exact Algorithms*, PhD Thesis, Universidade do Minho, 2005.

[5] Barnhart, C., Johnson, L., Nemhauser, G. L., Savelsbergh, M. W. P., Vance, P. H. – *Branch-and-Price: Column Generation for Solving Huge Integer Problems*, Operations Research 46, 1998.

[6] Bianco L., Blazewicz J., Dell'Olmo P., Drozdowski M. – *Preemptive Multiprocessor Task Scheduling With Release Times and Time Windows*, Annals of Operations Research, 70, 43–57, 1997.

[7] Blazewicz, J., Ecker, K., Schmidt, G., Weglarz, J. – *Scheduling in Computer and Manufacturing Systems*, 2nd ed., Springer, New York, 1994.

[8] Blazewicz, J., Kovalyov, M. Y., Machowiak, M., Trystram, D., Weglarz, J. – *Scheduling Malleable Tasks on Parallel Processors to Minimize the Makespan*, Annals of Operations Research 129, 65–80, 2004.

[9] Blazewicz, J., Lenstra, J. K., Rinnooy Kan, A. – *Scheduling subject to resource constraints: classification and complexity*, Discrete Applied Mathematics, 5, 1983.

[10] Bruno, J., Gonzalez, T. – *Scheduling Independent Tasks with Release Dates and Due Dates on Parallel Machines*, Technical Report 213, Computer Science Department, Pennsylvania State University, College Station, 1976.

[11] Chen, Z. L., Lee, C. Y. – *Parallel Machine Scheduling With a Common Due Window*, European Journal of Operations Research, 136, 512-527, 2002.

[12] Chen, Z. L., Lee, C. Y. – *General Multiprocessor Task Scheduling*, Naval Research Logistics, 46, 57-74, 1999.

[13] Chen, Z. L., Powell, W. B. – *A Column Generation Based Decomposition Algorithm for a Parallel Machine Just-in-time Scheduling Problem*, European Journal of Operations Research, 116, 220-232, 1999.

[14] Chen, Z. L., Powell, W. B. – *Exact Algorithms for Scheduling Multiple Families of Jobs on Parallel Machines*, Naval Research Logistics, 50, 823-840, 2003.

[15] Chen, Z. L., Powell, W. B. – *Solving Parallel Machine Scheduling Problems by Column Generation*, INFORMS Journal on Computing, 11, 1, 78-97, 1999.

[16] Cheng, T. C. E.; Sin, C. C. S. – *A state-of-the-art review of parallel-machine scheduling research*, European Journal of Operational Research 47, 271-292, North-Holland, 1990.

[17] Conway, R. W., Maxwell, W. L., Miller, L. W. – *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.

[18] Dantzig, G. B., Wolfe, P. – *Decomposition Principle for Linear Programs*, Operations Research 8, 1960.

[19] Dell'Amico, M., Martello, S. – *Optimal Scheduling of Tasks on Identical Parallel Processors*, ORSA Journal on Computing, 7, 1995.

[20] Desaulniers, G., Desrosiers, J., Solomon, M. M. – *Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems*, Essays and Surveys in Metaheuristics, C. Ribeiro and P. Hansen (eds.), Kluwer, Norwell, MA, 309-324, 2002.

[21] Desrochers, M., Desrosiers, J., Solomon, M. – *A New Optimization Algorithm for the Vehicle Routing Problem With Time Windows*, Operations Research, 1992.

[22] Drozdowski, Maciej – *Scheduling Multiprocessor Tasks – an overview*, European Journal of Operations Research, 94, 215-230, 1996.

[23] Duarte, António – *Sistema de Planeamento Fino da Produção com Sequenciamento de Lotes e Agendamento de Máquinas*, Tese de Mestrado, Universidade do Minho, 2000.

[24] Du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P. - *Stabilized Column Generation*, Discrete Mathematics, 194, 1999.

[25] Elmaghraby, S. E., Park, S. H., *Scheduling Jobs on a Number of Identical Machines*, AIIE Transactions, 6, 1-13, 1974.

[26] Farley, A. A. – *A Note on Bounding a Class of Linear Programming Problems, Including Cutting Stock Problems*, Operations Research, Vol. 38, No. 5, 1990.

[27] Fekete, S., Schepers, J. – *New classes of fast lower bounds for bin packing problems*, Mathematical Programming, 91, 11–31, 2001.

[28] Ford, L. R., Fulkerson, D. R. – *A suggested computation for maximal multicommodity network flows*, Management Science, 5, 97-101, 1958.

[29] Garey, M., Johnson, D. – *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[30] Gilmore, P. C., Gomory, R. E. – *A Linear Programming Approach to the Cutting Stock Problem – part ii*, Operations Research 11, 1963.

[31] Gonzalez, T., Sahni, S. – *Preemptive Scheduling of Uniform Processor Systems*, Journal of the Association of Computing Machinery, 25, 92-101, 1978.

[32] Graham, R. L. – *Bounds on Multiprocessing Timing Anomalies*, SIAM Journal of Applied Mathematics 17:263-269, 1969.

[33] Graham, R., Lawler, E., Lenstra, J. K., Rinnooy Kan, A. – *Optimization and Approximation in Deterministic Sequencing and Scheduling: a survey*, Annals of Discrete Mathematics, Vol. 5, North-Holland, 1979.

[34] Hoogeveen, J., Lenstra, J. K., Van De Velde, S. – *Sequencing and Scheduling - Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli and S. Martello (eds.), John Wiley & Sons, 12, 181-197, 1997.

[35] Horn, W. A. – *Minimizing Average Flow Time with Parallel Machines*, Operations Research, 21, 846-847, 1973.

[36] Horvath, E. C., Lam, S., Sethi, R. – *A Level Algorithm for Preemptive Scheduling*, Journal of the Association of Computing Machinery, 24, 32-43, 1977.

[37] Kawaguchi, T., Kyan, S. – *Worst Case Bound of an LRF for the Mean Weighted Flow Time Problem*, SIAM Journal of Computing, 15, 1119-1129, 1986.

[38] Labetoulle, J., Lawler, E. L., Lenstra, J. K. Rinnooy Kan, A. H. G. – *Preemptive Scheduling of Uniform Machines Subject to Release Dates*, Progress in Combinatorial Optimization, W. R. Pulleyblank (ed.), 254-261, Academic Press, NY, 1984.

[39] Ladson, Leon S. – *Optimization Theory for Large Systems*, The Macmillan Company, Collier-Macmillan Limited, London, 1970.

[40] Lawler, E. L., Labetoulle, J. – On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming, Journal of the Association of Computing Machinery, 25, 612-619, 1978.

[41] Leung Y-T. – *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman & Hall/CRC Computer & Information Science Series, 2004.

[42] Lopes, Manuel J. P. – *Resolução de Problemas de Programação de Máquinas Paralelas pelo Método de Partição e Geração de Colunas*, Tese de Doutoramento, Universidade do Minho, 2004.

[43] Lueker, G. S. – *Bin packing with items uniformly distributed over intervals [a, b]*, Proc. 24th Annual Symposium Foundation of Computational Science (FOCS 1983), 289–297, 1983.

[44] McNaughton, R. – *Scheduling with deadlines and loss functions*, Management Science, Vol. 6, 1-12, 1959.

[45] Nemhauser, G., Wolsey, L. – *Integer and Combinatorial Optimization*, Wiley and Sons, New York, 1988.

[46] Pinedo, Michael – *Scheduling: theory, algorithms, and systems*, Englewood Cliffs, Prentice Hall, 1995.

[47] Soumis, F. – *Decomposition and column generation*, in *Annotated Bibliographies in Combinatorial Optimization*, Dell'Amico, F. Maffioli and S. Martello (eds.), Wiley, Chichester, 115-126, 1997.

[48] Sousa, J. P., Wolsey, L. A. – *A time indexed formulation of non-preemptive single machine scheduling problems*, Mathematical Programming 54, 1992.

[49] Valério de Carvalho, J. M. V. – *Exact Solution of Bin-Packing Problems Using Column Generation and Branch-and-Bound*, Annals of Operations Research 86, 1999.

[50] Valério de Carvalho, J. M. V. – *Using Extra Dual Cuts to Accelerate Convergence in Column Generation*, INFORMS Journal on Computing, 17, 2, 175-182, 2005.

[51] Vance, P. H., Barnhart, C., Johnson, E. L., Nemhauser, G. L. – *Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound*, Computational Optimization and Applications 3, 1994.

[52] Vanderbeck, F. – *Exact algorithm for minimizing the number of setups in the one-dimensional cutting stock problem*, Operations Research, 48(6): 915–926, 2000.