UNIVERSIDADE DO MINHO
Departamento de Produção e Sistemas

Resolução de Problemas de Programação de Máquinas Paralelas pelo Método de Partição e Geração de Colunas

Manuel Joaquim Pereira Lopes

Tese de Doutoramento

Realizada sob a orientação de
Professor Doutor José Manuel Valério de Carvalho,
Professor Catedrático do Departamento de Produção e Sistemas da Universidade do Minho

Braga, Dezembro de 2004

Aos meus pais

Agradecimentos

Ao Professor Doutor Valério de Carvalho com quem tive o privilégio de trabalhar durante os últimos anos. Para além dos muitos ensinamentos científicos e a forma dedicada como exerceu a orientação guardo também a sua amizade.

Ao Professor Coordenador Afonso Fernandes e ao Professor Coordenador Ismael Cavaco do Instituto Superior de Engenharia do Porto pelo seu apoio e amizade.

Ao Instituto Superior de Engenharia do Porto e à Universidade do Minho pela disponibilização dos seus meios.

A todos aqueles que directa ou indirectamente contribuíram para este trabalho.

À minha família pelo seu apoio e compreensão.

Este trabalho foi realizado ao abrigo de uma Bolsa do PRODEP concedida no concurso público nº 4/5.3/PRODEP/2000.

Resumo

O problema de programação de máquinas paralelas não-idênticas com tempos de preparação dependentes da sequência é uma generalização do problema de programação de máquinas paralelas que envolve a determinação da melhor afectação e sequenciamento de *n* trabalhos a *m* máquinas, com tempos de preparação dependentes da sequência de trabalhos processados, de forma a minimizar uma função de custo.

Neste trabalho, apresentamos um novo algoritmo de optimização para a solução de problemas de máquinas paralelas não-idênticas com tempos de preparação dependentes da sequência e datas de disponibilidade para as máquinas e para os trabalhos, de forma a minimizar a soma ponderada dos desvios positivos (atrasos) às datas de entrega dos trabalhos.

É apresentada uma primeira formulação matemática em que a função de custo é não-linear. Pela aplicação do método de decomposição de Dantzig-Wolfe obtemos uma formulação de programação inteira equivalente em que cada variável de decisão (coluna) representa uma sequência de afectação numa máquina. A relaxação linear desta formulação é resolvida pelo método de geração de colunas, onde as colunas são geradas em *m* subproblemas, em que cada um deles representa um problema de programação de máquina única. Colunas válidas são adicionadas à solução inicial admissível pela resolução de um problema de caminho mais curto com datas de disponibilidade das máquinas e dos trabalhos, usando programação dinâmica. São estudados vários modelos de programação dinâmica e é proposto um método para melhorar a eficiência do modelo adoptado.

A solução da relaxação linear obtida fornece geralmente um bom limite inferior, que é usado num algoritmo de partição e avaliação para resolver a formulação de programação inteira. É desenvolvida uma regra específica de partição que reduz significativamente o número de nodos explorados na árvore de pesquisa. É ainda desenvolvida uma heurística para determinar uma solução inicial para o problema.

São apresentados métodos de aceleração e estabilização do algoritmo de geração de colunas e é proposto um novo método que designamos por "*primal boxstep*".

Por último, são apresentados resultados de testes computacionais para uma gama alargada de problemas com diferentes características e diferentes níveis de congestionamento do sistema, que mostram que esta aproximação é capaz de resolver problemas de dimensão significativa em tempo computacional considerado razoável.

Summary

We consider a class of scheduling problems of unrelated parallel machines with sequence-dependent setup times which involves finding the best assignment and sequencing of *n* jobs to *m* machines, with sequence-dependent setup times, to minimize a cost function.

In this work, we develop a new optimization algorithm for the solution of the unrelated parallel machines with sequence-dependent setup times and release dates for the machines and the jobs scheduling problem, to minimize the weighted tardiness of the jobs.

First we give a mathematical formulation for the problem where the cost function is non-linear. By applying the Dantzig-Wolfe decomposition to this formulation we obtain an equivalent integer programming formulation where each decision variable (column) represents a single machine schedule. The linear programming relaxation of the integer programming formulation is solved by column generation, where columns are generated on *m* subproblems which are single-machine scheduling problems. Feasible columns are added as needed by solving a shortest path problem with release dates for the jobs and machines, using dynamic programming. Several dynamic programming models are studied and a method to increase the efficiency of the model chosen is proposed.

The linear programming solution obtained generally provides a strong lower bound that is used in a branch-and-bound algorithm to solve the integer formulation. A specific branching rule that reduces significantly the number of nodes explored is presented.

Column generation stabilization and accelerating methods are presented and a new method designated by "*primal boxstep*" is proposed.

A heuristic was developed to find an initial solution for the problem.

The computational results show that the approach is capable of solving problems of large size to optimality within reasonable computational time for a wide variety of problems with different characteristics and levels of congestion of the system.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Exemplos

Capítulo 1

Introdução

Os problemas de programação, designados na literatura anglo-saxónica como "*scheduling problems*", têm sido objecto de intensa investigação ao longo das últimas décadas dando corpo a uma literatura muito vasta. Com cerca de uma centena de capítulos dedicados a áreas específicas e contribuições de cerca de uma centena de autores, o manual "Handbook of Scheduling: Algorithms, Models, and Performance Analysis" de Joseph Leung (2004), constitui uma referência nesta área. A revista "Journal of Scheduling" dedicada exclusivamente a esta área e lançada em 1998 ou os recursos na internet, dos quais destacamos uma biblioteca de recursos sobre programação, http://benli.bcc.bilkent.edu.tr/~lors/ (Library of Online Resources on Scheduling), são outros exemplos da importância desta área científica.

Os problemas de programação desempenham na actualidade um papel crucial tanto na indústria como nos serviços. A comprová-lo estão os livros recentes publicados sobre este tema: "Scheduling: Theory, Algorithms, and Systems" de Pinedo (2001) e "Operations Scheduling with Applications in Manufacturing and Services" de Pinedo e Chao (1999).

O problema básico de programação é o da afectação de recursos limitados a actividades ao longo do tempo. Trata-se de um processo de decisão que visa a optimização de um ou vários objectivos. Os recursos e as actividades podem tomar variadas formas. Os recursos podem ser máquinas num centro de trabalho, tripulações de aviões, processadores de computadores (cpu), camiões de distribuição, etc. As actividades

podem assumir a forma de operações num processo produtivo, aterragens e descolagens de aviões num aeroporto, encomendas a distribuir pelos clientes, operações a executar no computador, etc.

Os objectivos podem também assumir várias formas sendo mais comuns os baseados nos tempos de conclusão e nas datas de entrega das actividades. No actual ambiente de elevada competição, a utilização eficiente dos recursos disponíveis ou o cumprimento das datas de entrega acordadas com os clientes tornaram-se factores fundamentais para a sobrevivência das empresas.

Os problemas de programação são normalmente difíceis tanto do ponto de vista técnico como do ponto de vista da implementação das soluções. As dificuldades técnicas estão normalmente relacionadas com a sua natureza combinatória enquanto que as dificuldades de implementação estão relacionadas com a modelação de problemas reais e a aquisição de dados (Pinedo, 2001).

As muitas situações reais onde a programação desempenha um papel crucial e a sua grande complexidade (ver http://www.mathematik.uni-osnabrueck.de/research/OR/class/ para a classificação de complexidade de problemas) inspiraram investigadores de diferentes áreas, tais como, Gestão Industrial, Ciências de Computação, Investigação Operacional, Matemática e Gestão, a estudar intensivamente este tema construindo uma forte base de conhecimento que a transformou numa área teórica com resultados muito interessantes.

Neste trabalho é apresentado um método de resolução para uma classe de problemas de programação de grande complexidade (NP-difíceis) com relevância teórica e prática (ver Secção 1.1 para um exemplo real): a programação de máquinas paralelas não-idênticas (*unrelated parallel machines scheduling*). Esta classe de problemas de programação trata o caso mais geral de uma família mais vasta de problemas de programação de máquinas paralelas (*parallel machines scheduling*) que inclui as máquinas paralelas idênticas (*identical parallel machines*) e as máquinas paralelas uniformes (*uniform parallel machines*).

A programação de máquinas paralelas pode ser considerada com um processo em duas fases. Em primeiro lugar, determinar em que máquina é que cada um dos trabalhos vai ser processado e, em segundo lugar, determinar a sequência de processamento dos trabalhos afectados a cada uma das máquinas.

A programação matemática é considerada como uma forma natural de formular este tipo de problemas (para uma revisão de formulações de programação matemática para problemas de programação de máquinas ver Blazewicz, Dror e Weglarz, 1991).

O desenvolvimento de técnicas numéricas e algorítmicas sofisticadas, muitas vezes explorando a estrutura particular do problema em estudo, em conjunto com o enorme progresso tecnológico dos computadores, têm permitido importantes avanços na abordagem de problemas combinatórios de grande dimensão e considerados difíceis, através de métodos de solução exacta.

A utilização de procedimentos baseados na combinação de geração diferida de colunas com o método de partição e avaliação para a solução destes modelos constitui uma abordagem muito promissora. Embora os métodos enunciados sejam já conhecidos há mais de três décadas, o desenvolvimento de procedimentos baseados na sua combinação é bastante recente. Na literatura anglo-saxónica, esta técnica de solução tem sido designada por *branch-and-price*. No nosso trabalho, utilizamos este método de solução exacta para a resolução de problemas de programação de máquinas paralelas não-idênticas.

1.1. Motivações

A motivação para a realização deste trabalho surge na sequência de um contacto estabelecido com uma empresa têxtil multinacional implantada no norte de Portugal. Através deste contacto foi-nos dado a conhecer um problema de programação da produção relacionado com a secção de Tinturaria (este problema é comum a outras empresas têxteis que possuem secções de Tinturaria).

Se bem que tenha existido um grande interesse inicial por parte da empresa acima referida, o mesmo não foi confirmado posteriormente a ponto de permitir uma parceria no desenvolvimento de um projecto com interesse relevante comum aos mundos académico e empresarial. Contudo, ficaram a forte convicção de se tratar de um problema real importante (para esta indústria e muitas outras) e a esperança de um dia poder complementar o trabalho académico entretanto desenvolvido com uma implementação em ambiente real.

Entendendo que muita da informação, de carácter geral, entretanto recolhida na empresa poderia ajudar a uma melhor compreensão do problema em questão, resolvemos apresentá-la aqui. É neste sentido que devem ser entendidos os parágrafos seguintes desta subsecção.

Esta empresa possui uma estrutura vertical que vai desde a Tricotagem até à Confecção e opera num ambiente *Make-to-Order*. Os seus clientes são sobretudo clientes internacionais muito exigentes, principalmente ao nível da qualidade e do cumprimento dos prazos de entrega. Enfrentam forte concorrência na Europa, por parte sobretudo de empresas italianas e turcas.

Processam entre 200 a 250 encomendas por semana a que corresponde cerca de 10 toneladas de fio processado por dia. Tipicamente, recebem um pedido de fabrico de uma amostra ao qual se segue uma grande encomenda. O prazo de entrega típico é de 4 a 6 semanas e existem penalizações por entregas em atraso mas as entregas antes da data acordada não representam qualquer problema para os seus clientes.

As encomendas são recebidas pelos Serviços Comerciais que as enviam para o Planeamento Central que efectua uma programação para trás (*backward scheduling*) para determinar as datas limite de lançamento das encomendas, como ordens de fabrico, no sistema de produção e as datas da sua conclusão nas várias áreas de produção por onde estas têm de passar. As ordens de fabrico são lançadas no sistema de produção o mais cedo possível tentando assim garantir o cumprimento das datas de entrega acordadas.

A decisão relativa ao momento exacto em que cada operação, constante do roteiro de produção de cada encomenda, é levada a cabo, é tomada localmente e é da responsabilidade de cada encarregado fabril da respectiva área de produção.

As matérias-primas são aprovisionadas de acordo com as necessidades específicas de cada encomenda. Existem, contudo, algumas matérias-primas que são comuns à maioria das encomendas (tal como o fio) e que são aprovisionadas para stock permitindo, assim, reduzir os prazos de entrega.

A área de Tinturaria tem como área de produção vizinha a montante a Tecelagem e como áreas de produção vizinhas a jusante, a área de Embalamento, para as encomendas de tecido em rolo, e a área de Corte de Tecido para as restantes encomendas. É constituída por máquinas de tingimento, com o nome técnico de "Jets", com capacidades e velocidades de processamento diferentes e todas elas controladas por computador.

Estas máquinas representam um grande investimento (superior a 500 mil euros por máquina) e têm custos, relativos a consumos de energia eléctrica, produtos químicos, água e tempo do operador e da máquina, e tempos de mudança de cor, dependentes da sequência de cores a processar, elevados, constituindo o gargalo produtivo (*bottleneck*) típico.

Encontrar o melhor programa de produção que tenha em consideração as características específicas desta área crítica, é o problema que os responsáveis do Planeamento e da Produção enfrentam regularmente.

A ideia defendida pelos responsáveis da empresa, em termos genéricos, é a de que, partindo das datas limite de conclusão das ordens de fabrico, determinadas pelo *backward scheduling*, para a área de Tinturaria e as datas mais cedo a que a área imediatamente precedente (área de Tecelagem) pode terminar o processamento das ordens de fabrico em-curso, determinadas por um planeamento para a frente (*forward scheduling*), fosse possível determinar o programa de produção que melhor utilização fizesse dos recursos instalados na área de Tinturaria minimizando a soma dos atrasos

ponderados pela prioridade atribuída a cada encomenda, que é uma função do cliente e das características particulares da encomenda. O programa de produção optimizado para a área de Tinturaria definiria novas datas limite de conclusão das ordens de fabrico para a área imediatamente precedente (área de Tecelagem).

De referir que esta aproximação sugerida pelos responsáveis da empresa se encontra referenciada na literatura como "*critical resource schedule*" ou "*bottleneck schedule*". O princípio que lhe está subjacente é o de que o uso eficiente de um recurso crítico tornará o sistema globalmente mais eficiente. No *bottleneck schedule*, o recurso crítico é programado em primeiro lugar para depois os outros recursos serem programados em função do programa determinado para o primeiro.

Os modelos apresentados neste trabalho (Capítulo 5) fazem uma abstracção deste problema real, tentando capturar, entre o grande número de factores presentes num problema de programação da produção, não mais do que a complexidade fundamental para a resolução de um problema de programação de máquinas paralelas.

1.2. Contribuições da Tese

Salientam-se as seguintes contribuições deste trabalho:

- A apresentação de um algoritmo de resolução exacta do problema de programação de n trabalhos, com datas de disponibilidade e datas de entrega, em m máquinas paralelas não-idênticas, com datas de disponibilidade e tempos de preparação dependentes da sequência de trabalhos a processar. Este problema é um caso geral dos problemas de programação de máquinas paralelas e o algoritmo desenvolvido pode ser aplicado a outros casos particulares com medidas de desempenho regulares;

- A apresentação de um modelo de programação matemática e a sua decomposição pelo método de Dantzig-Wolfe que conduz a uma reformulação do problema que é resolvida utilizando o método de partição e geração de colunas;

- O desenvolvimento de um método (custos reduzidos decrescentes) para melhorar a eficiência do algoritmo de programação dinâmica proposto para a resolução dos subproblemas;
- O desenvolvimento de uma regra específica de partição que diminui significativamente o números de nodos explorados na árvore de pesquisa;
- O desenvolvimento de uma heurística para a obtenção de soluções para o problema, nomeadamente uma solução inicial;
- O desenvolvimento de uma estratégia de aceleração dos processos de geração de colunas, que designamos por *primal boxstep*, baseada numa restrição temporária do espaço de soluções primal, o que favorece uma mais rápida convergência do algoritmo para uma região próxima da solução óptima, e se traduz numa redução muito significativa do tempo de resolução computacional;
- A apresentação de um estudo computacional com a resolução de instâncias com características muito diversificadas;
- A verificação de que é possível, usando o modelo que resulta da decomposição de Dantzig-Wolfe e estratégias de aceleração, resolver instâncias de dimensão significativa em tempo computacional razoável.

1.3. Estrutura da Tese

No Capítulo 2 é feita a apresentação formal dos problemas de programação de máquinas paralelas. Depois de uma breve introdução aos modelos máquinas paralelas, é apresentada uma definição para os problemas de programação de máquinas paralelas. Por último, é apresentada uma notação para a sua classificação.

No Capítulo 3 é introduzido o método de partição e geração de colunas para um problema genérico de programação inteira. Este método pode ser encarado como uma combinação (não trivial) dos métodos de geração de colunas e de partição e avaliação o que justifica a sua análise neste Capítulo.

No Capítulo 4 apresentamos uma revisão da literatura sobre máquinas paralelas focada principalmente nos métodos de solução exacta, em geral, e no método de partição e geração de colunas, em particular. Antes de abordar os métodos de solução exacta, fazemos uma muito breve abordagem aos métodos heurísticos.

No Capítulo 5 começamos por apresentar a definição formal do problema de programação. Segue-se a apresentação da formulação matemática para o problema de programação de máquinas paralelas não-idênticas em estudo. Depois de apresentada uma formulação compacta de programação inteira, é descrita a aplicação do método de decomposição de Dantzig-Wolfe ao problema em estudo. Segue-se a descrição de três modelos de programação dinâmica para a resolução do subproblema que representam um percurso evolutivo na selecção do modelo mais eficiente para o problema em estudo. É ainda apresentado um método para melhorar a eficiência do algoritmo de programação dinâmica proposto. Finalmente, é apresentado o algoritmo de partição e avaliação, onde se referem as estratégias de pesquisa mais frequentes e a estratégia escolhida, e é provada a sua convergência.

No Capítulo 6 são apresentadas estratégias para a estabilização e aceleração do algoritmo de geração de colunas e a aplicação ao problema em estudo daquelas que nos pareceram mais adequadas. Por último, propomos um método de aceleração do algoritmo de geração de colunas baseado na restrição do espaço primal (*primal boxstep*).

No Capítulo 7 refere-se a implementação efectuada e apresentam-se e discutem-se duas formas de obter o primeiro problema primal restrito válido, necessário para a inicialização do método de geração de colunas. Para a obtenção de uma solução inicial, é descrita uma heurística que foi desenvolvida tendo em conta a especificidade do nosso problema. Apresentam-se ainda os resultados dos testes computacionais efectuados em instâncias com diferentes características e uma análise de sensibilidade do método à variação dos parâmetros do problema. Finalmente, são apresentadas extensões do modelo desenvolvido a outras funções de custo.

No Capítulo 8 são apresentadas as principais conclusões extraídas deste trabalho e apontadas algumas direcções para trabalho futuro.

Capítulo 2

Problemas de Programação de Máquinas Paralelas

Neste Capítulo é feita a apresentação formal dos problemas determinísticos de programação de máquinas paralelas segundo uma versão adaptada de Blazewicz, Lenstra e Kan (1983) e Pinedo (1995). Depois de uma breve introdução aos modelos máquinas paralelas (Secção 2.1), é apresentada uma definição para este tipo de problemas (Secção 2.2). Por último, é apresentada uma notação para a sua classificação (Secção 2.3).

2.1. Introdução

O estudo dos problemas de máquinas paralelas constitui um tema importante tanto do ponto de vista teórico como do ponto de vista prático. Do ponto de vista prático, é importante porque a ocorrência de recursos em paralelo é frequente no mundo real. Do ponto de vista teórico, trata-se de uma generalização da máquina única e um caso particular de um sistema de produção flexível. Também, as técnicas aplicadas aos problemas de máquinas paralelas são frequentemente usadas em procedimentos de decomposição para sistemas multi-estratégia.

Não é difícil apresentar exemplos práticos de programação de máquinas paralelas. Na área das ciências de computação, temos o exemplo dos processadores em paralelo. Nos sistemas de produção tipo *Flow Shop*, existe muitas vezes mais do que uma máquina do

mesmo tipo para evitar que todo o sistema pare quando uma única máquina avaria. Mais ainda, a existência de mais do que uma máquina do mesmo tipo reduz a possibilidade de ocorrência de situações de gargalo produtivo (*bottleneck*). Nos sistemas de produção tipo *Job Shop*, a adição de um número apropriado de máquinas pode aumentar a sua flexibilidade que, por sua vez, pode reduzir o tempo de fluxo dos trabalhos (Murty, 1995).

A programação de máquinas paralelas pode ser vista como um processo em dois passos. Em primeiro lugar, determina-se a afectação das máquinas aos trabalhos; em segundo lugar, determina-se a sequência de processamento dos trabalhos.

Uma simplificação do problema das máquinas paralelas é considerar as máquinas como um sistema agregado. Neste caso, os resultados da teoria de programação da máquina única podem ser aplicados aos sistemas de máquinas paralelas. A limitação desta aproximação é que o modelo, focado somente no desempenho global do sistema, perde informação sobre o desempenho individual de cada máquina. Apesar desta simplificação, a programação de um único sistema agregado não é, de forma alguma, um problema de fácil resolução.

Matematicamente, um problema de máquinas paralelas envolve um nível de complexidade semelhante ao da máquina única ao qual é adicionado mais um nível de complexidade resultante da afectação das máquinas aos trabalhos. Assim, a complexidade geral dos problemas de máquinas paralelas é inflacionada, muitas vezes exponencialmente, quando o número de máquinas ou trabalhos ou ambos aumentam.

Para uma revisão de literatura sobre máquinas paralelas ver Cheng e Sin (1990) e Hoogeveen, Lenstra e van de Velde (1997).

2.2. Definições

No texto, iremos usar o termo tarefa para designar uma operação elementar a ser realizada numa máquina e o termo trabalho (*job*) como um subconjunto de tarefas. Para o caso particular da programação de máquinas paralelas, os termos tarefa ou trabalho

serão usados indiferentemente já que é assumido que cada trabalho realiza uma única operação elementar numa máquina. Também os termos medida de desempenho, critério de optimalidade ou função objectivo, serão usados indiferentemente ao longo do texto.

Os problemas de programação de máquinas paralelas são caracterizados pelas seguintes entidades:

- Conjunto das n tarefas, $T=\{T_1,T_2,...,T_j,...,T_n\}$;
- Conjunto das m máquinas, $M=\{M_1,M_2,...,M_k,...,M_m\}$;
- Conjunto dos s recursos adicionais (operadores, ferramentas, etc.), $R=\{R_1, R_2,...,R_s\}$.

Um problema de programação de máquinas paralelas normalmente significa afectar máquinas de M a tarefas de T, de forma a concluir o seu processamento respeitando as condições impostas. O número de tarefas é representado por n e o número de máquinas é representado por m. Normalmente, os índices i e j referem-se às tarefas e o índice k refere-se à máquina.

Existem duas condições gerais na teoria clássica da programação:

- Cada tarefa deve ser processada em uma máquina de cada vez, no máximo;
- Cada máquina é capaz de processar uma tarefa de cada vez, no máximo.

Existem algumas novas aplicações em que a primeira restrição é relaxada.

A especificação de um problema de programação de máquinas paralelas requer a descrição das características das máquinas, das características das tarefas e do critério de optimalidade. As máquinas podem ser dos seguintes tipos:

- *Paralelas*: executam o mesmo tipo de funções;
- *Dedicadas*: especializadas na execução de certas tarefas.

Dentro das máquinas paralelas, dependendo da sua velocidade de processamento das tarefas, podemos distinguir três tipos de máquinas diferentes:

- *Idênticas* (*identical*): iguais velocidades de processamento das tarefas;
- *Uniformes* (*uniform*): diferentes velocidades de processamento das tarefas mas a velocidade de cada máquina é constante não dependendo da tarefa a processar;
- *Não-idênticas* (*unrelated*): velocidade de processamento dependente da tarefa particular a ser executada.

De uma forma geral, uma máquina $k \in M$ é caracterizada pela seguinte informação:

- Vector do estado inicial das máquinas $L=[l_1,l_2,...,l_m]$, onde l_k representa a tarefa em processamento na máquina k antes de esta ficar disponível;
- Vector das datas de disponibilidade $A=[a_1,a_2,...,a_m]$ onde a_k é o instante de tempo em que a máquina k fica disponível para poder ser utilizada.

De uma forma geral, uma tarefa $j \in T$ é caracterizada pela seguinte informação:

- Vector de tempos de processamento $P_j=[p_{j1},p_{j2},...,p_{jm}]$, para j=1, 2, ...,n, onde p_{jk} é o tempo necessário para a máquina k processar a tarefa j. No caso de máquinas idênticas, teremos $p_{jk}=p_j$, k=1, 2,...,m. Se as máquinas forem uniformes então teremos $p_{jk}=p_j/b_k$, onde p_j é o tempo de processamento standard (normalmente medido na máquina mais lenta) e b_k é o factor de velocidade de processamento da máquina k;
- Vector das datas de disponibilidade (*release dates*) $R=[r_1,r_2,...,r_n]$, onde r_j é o instante de tempo em que a tarefa j fica disponível para ser processada;
- Vector das datas de entrega (*due dates*) $D=[d_1,d_2,...,d_n]$, onde d_j é o limite de tempo até ao qual a tarefa j deve ficar concluída. Normalmente são definidas funções de penalidade para a violação das datas de entrega;

- Vector das datas limite de entrega (*deadline*), $\widetilde{D} = \left[\widetilde{d}_1, \widetilde{d}_1, ..., \widetilde{d}_n\right]$ que define o limite de tempo até ao qual a tarefa *j* tem que ser impreterivelmente concluída;

- Vector dos pesos/prioridades $W=[w_1,w_2,...,w_n]$, onde w_j representa o peso associado à tarefa *j*; podem expressar custos de processamento, custos de imobilização, penalidades associadas a avanços ou atrasos, ou a urgência relativa da tarefa *j*;

- Vector de tempos de preparação (*setup*) da máquina dependentes da sequência de tarefas a processar $S_j=[s_{1j},s_{2j},...,s_{nj}]$, para $j=1,2,...,n$, onde s_{ij} é o tempo de preparação da máquina se o processamento da tarefa *j* for realizado imediatamente após o processamento da tarefa *i*. No caso dos tempos de preparação da máquina entre as tarefas *i* e *j* serem dependentes da máquina, é incluído o índice *k* referente à máquina e s_{kij} é o tempo de preparação da máquina *k* se o processamento da tarefa *j* for realizado imediatamente após o processamento da tarefa *i*; s_{k_j} representa o tempo de preparação da máquina *k* quando a primeira tarefa a ser processada é a tarefa *j*. Se os tempos de preparação das máquinas são independentes da sequência de tarefas a processar podem simplesmente ser adicionados aos tempos de processamento;

- Necessidades de recursos (se necessários).

Assumimos que todos estes parâmetros tomam valores inteiros. Na prática, esta assunção não se torna muito restritiva já que ela afecta sobretudo os parâmetros definidos pela variável tempo e, para esta variável, o factor escala permite uma grande flexibilidade. Assumimos também que os recursos são afectados às tarefas quando o processamento destas se inicia ou se reinicia, e que são libertados quando o processamento das tarefas está concluído ou é interrompido.

Uma programação diz-se *interrompida* (*preemptive*) se o processamento de cada tarefa pode ser interrompido em qualquer momento e reiniciado mais tarde, sem custo adicional, possivelmente noutra máquina. Se o processamento das tarefas não pode ser interrompido, a programação diz-se *não-interrompida* (*nonpreemptive*).

Podem ser definidas restrições de precedência entre as tarefas do conjunto T. $T_i \prec T_j$ significa que o processamento da tarefa i deve ser concluído antes de se dar início ao processamento da tarefa j. As tarefas do conjunto T dizem-se *dependentes* se a ordem de execução de pelo menos duas das suas tarefas tem uma restrição de precedência. Caso contrário, as tarefas dizem-se *independentes*.

A tarefa j diz-se disponível no instante de tempo t se $r_j \leq t$ e todas as tarefas suas predecessoras já estiverem concluídas.

Vamos agora apresentar algumas definições relativamente à programação e critérios de optimalidade.

Um *programa* é uma afectação de máquinas do conjunto M (e possivelmente recursos do conjunto R) a tarefas do conjunto T num tempo tal que as seguintes condições sejam satisfeitas:

- Em qualquer momento cada máquina está a processar uma tarefa, no máximo, e cada tarefa está, no máximo, a ser processada por uma máquina[1],
- A tarefa j é processada no intervalo de tempo $[r_j, \infty)$,
- Todas as tarefas estão concluídas,
- Se as tarefas i e j estão relacionadas em $N_i \prec N_j$, o processamento da tarefa j não pode começar antes da tarefa i estar concluída,
- No caso de programação *não-interrompida* nenhuma tarefa é interrompida (o programa diz-se *não-interrompido*), caso contrário o número de interrupções de cada tarefa é finito e a programação diz-se *interrompida*,
- As restrições associadas aos recursos, se existentes, são satisfeitas.

[1] Como já mencionamos, esta restrição pode ser relaxada

Para representar programas vamos usar diagramas de Gantt. Os seguintes parâmetros podem ser calculados para cada tarefa j=1,2,...,n, processada num dado programa:

- *Tempo de conclusão* (*completion time*) - C_j;
- *Tempo de fluxo* (*flowtime*) $F_j = C_j - r_j$, como a soma dos tempos de espera e processamento;
- *Desvio à data de entrega* (*lateness*) - $L_j = C_j - d_j$;
- *Avanço* (*earliness*) - $E_j = \max\{d_j - C_j, 0\}$;
- *Atraso* (*tardiness*) - $T_j = \max\{C_j - d_j, 0\}$;
- *Tarefa com atraso* (*tardy job*) - $U_j = 1$ se $C_j > d_j$; caso contrário, $U_j = 0$;

Para avaliar programas podemos usar várias *medidas de desempenho* (*critérios de optimalidade* ou *funções objectivo*) de entre as quais destacamos:

Baseadas no tempo de conclusão

- *Makespan* (*duração máxima*), $C_{\max} = \max\{C_j\}$;
- Soma dos tempos de conclusão (*total completion time*), $C = \sum_{j=1}^{n} C_j$, ou soma ponderada dos tempos de conclusão (*total weighted completion time*), $C_w = \sum_{j=1}^{n} w_j C_j$;

Baseadas no tempo de fluxo

- Tempo de fluxo total (*total flowtime*), $F = \sum_{j=1}^{n} F_j$, ou soma ponderada dos tempos de fluxo (*total weighted flowtime*), $F = \sum_{j=1}^{n} w_j F_j$;

Baseadas na data de entrega

- Desvio máximo à data de entrega (*maximum lateness*), $L_{\max} = \max\{L_j\}$;

- Soma dos atrasos (*total tardiness*), $T = \sum_{j=1}^{n} T_j$, ou soma ponderada dos atrasos (*total weighted tardiness*), $T_w = \sum_{j=1}^{n} w_j T_j$;

- Número total de tarefas com atraso (*number of tardy jobs*), $U = \sum_{j=1}^{n} U_j$, ou número total ponderado de tarefas com atraso (*weighted number of tardy jobs*), $U_w = \sum_{j=1}^{n} w_j U_j$;

- Soma dos avanços e atrasos (*total earliness-tardiness penalty*), $\sum_{j=1}^{n} \left(E_j + T_j \right)$, ou soma ponderada dos avanços e atrasos (*weighted total earliness-tardiness penalty*) $\sum_{j=1}^{n} \left(u_j E_j + v_j T_j \right)$, em que podem também ser atribuídos pesos diferentes aos avanços e atrasos, u_j e v_j, respectivamente.

Podemos ainda ter valores médios para as medidas de desempenho do tipo aditivo (por exemplo, *tempo de fluxo médio*, $\overline{F} = 1/n \sum_{j=1}^{n} F_j$, ou *média ponderada do tempo de fluxo*, $\overline{F}_w = \sum_{j=1}^{n} w_j F_j / \sum_{j=1}^{n} w_j$).

É importante referir alguns aspectos relacionados com o significado prático e utilização das medidas de desempenho atrás referidas.

Makespan: a sua minimização conduz, normalmente, a níveis elevados de utilização das máquinas porque uma vez que todas as tarefas estejam processadas as máquinas ficam disponíveis para outras tarefas. Para situações de uma única máquina, este objectivo só tem interesse quando existem tempos de preparação das máquinas dependentes da sequência. De outra forma, o *makespan* é igual à soma dos tempos de processamento e, portanto, independente da sequência. Quando tratamos de máquinas em paralelo, este objectivo tem um interesse significativo. Na prática, o problema de balanceamento das cargas nas máquinas em paralelo ocorre frequentemente e a minimização do *makespan* assegura bons resultados.

Medidas de desempenho relacionadas com os tempos de conclusão: a sua minimização resulta num menor tempo médio de permanência no sistema de produção o que resulta

num menor tempo médio de resposta aos pedidos dos clientes e menor custo médio de stocks em-curso de fabrico. Note-se ainda que as medidas de desempenho relacionadas com o tempo de fluxo fazem sentido quando as datas de disponibilidade r_j das tarefas não são iguais.

Medidas de desempenho relacionadas com as datas de entrega: estas medidas de desempenho são de grande importância especialmente em ambientes *make-to-order* e normalmente têm um impacto directo na fidelização dos clientes e podem constituir uma vantagem competitiva. A sua minimização resulta num melhor serviço prestado ao cliente. De referir que as medidas de desempenho relacionadas com os atrasos são adequadas quando o processamento das tarefas antes da data de entrega não traz qualquer penalidade; existem só penalidades associadas às entregas tardias. Na prática, as medidas de desempenho relacionadas com o número de tarefas em atraso são muitas vezes usadas para medir a percentagem de encomendas concluídas a tempo (*on-time delivery*). Quando existe a possibilidade de as encomendas sofrerem atrasos inaceitáveis, são usadas medidas de desempenho que têm em conta o valor dos atrasos (*Lmax* ou *T*). Para ambientes do tipo *JIT* (*Just-in-time*) as medidas de desempenho devem ter em conta os avanços e os atrasos (*E* e *T*).

2.3. Classificação

A grande variedade de problemas de programação da produção, originou a necessidade de introduzir uma notação sistemática para a sua classificação, de forma a facilitar a sua apresentação e discussão. Iremos descrever e seguir a notação proposta por Graham, Lawler, Lenstra e Rinnooy Kan (1979) e Blazewicz, Lenstra e Kan (1983), limitando a nossa descrição ao âmbito das máquinas paralelas.

A notação é composta por três campos $\alpha|\beta|\gamma$, com os seguintes significados:

- $\alpha=\alpha_1\alpha_2$ descreve o tipo de máquina e o seu número;
    - $\alpha_1=1$: uma única máquina;

- α_1=*P*: máquinas *idênticas*;
- α_1=*Q*: máquinas *uniformes*;
- α_1=R: máquinas não-idênticas;
- α_2= número de máquinas do problema (quando não mencionado significa que o número de máquinas é variável).

- β descreve as características das tarefas e recursos adicionais. É neste campo que são indicadas características como a existência de datas de disponibilidade, *deadlines*, e tempos de preparação das máquinas.

- γ denota a medida de desempenho.

Um programa cujo valor de uma determinada variável de medida de desempenho γ está no seu mínimo será chamado de óptimo, e o correspondente valor de γ será designado por γ^*. Podemos agora definir um problema de programação Π como um conjunto de parâmetros descritos nesta secção e um critério de optimalidade, sem lhes associar valores numéricos. Uma *instância I* do problema Π é obtida pela especificação dos valores particulares para todos os parâmetros do problema.

Um algoritmo de programação é um algoritmo que constrói um programa para um dado problema Π.

Capítulo 3

Método de Partição e Geração de Colunas

Neste capítulo, fazemos uma introdução ao método de partição e geração de colunas que tem por objectivo servir de suporte à sua utilização que é feita nos Capítulos seguintes.

3.1. Introdução

Se bem que já estejam decorridas mais de quatro décadas sobre a publicação do princípio da decomposição de Dantzig-Wolfe (1960) foi nas duas últimas décadas que este método revelou todo o seu potencial na resolução de problemas de grande dimensão. Se bem que os conceitos básicos se tenham mantido inalterados, a sua associação ao método de partição e avaliação, chamada de método de partição e geração de colunas (*branch and price*), e o desenvolvimento notável da tecnologia computacional, tornaram possível a solução exacta de problemas de programação inteira de grande dimensão.

O método de partição e geração de colunas tem vindo a ser aplicado com sucesso na resolução de diversos tipos de problemas difíceis tais como, programação de rotas (*vehicle routing*) - Desrochers, Desrosiers e Solomon (1992), *graph colouring* - Mehrotra e Trick (1996), afectação (*generalized assignment*) - Savelsbergh (1997), corte (*cutting stock*) - Vance (1998), empacotamento (*bin-packing*) - Valério de Carvalho (1999), e *lot sizing* - Kang, Malik e Thomas (1999). Para uma revisão da

literatura na utilização de geração de colunas na resolução de problemas de programação inteira, ver Barnhart, Johnson, Nemhauser, Savelsbergh e Vance (1998).

Os princípios básicos da decomposição de Dantzig-Wolfe (1960) podem ser descritos da seguinte forma. Os problemas de programação linear podem apresentar não só um elevado número de variáveis como também um elevado número de restrições. Uma forma de ultrapassar esta dificuldade é transformando o problema numa formulação equivalente de programação linear com menos restrições mas com mais variáveis. São depois usados procedimentos de geração de colunas para resolver o problema equivalente.

A adopção de formulações com um elevado número de variáveis para a obtenção de soluções para problemas de programação inteira justifica-se por várias razões:

(i) A formulação original pode apresentar uma relaxação linear fraca. Frequentemente a relaxação linear pode ser reforçada por uma reformulação que envolve um elevado número de variáveis. É esta a principal razão para a aplicação do método em problemas de corte e empacotamento (Valério de Carvalho, 1999) e no problema de afectação generalizado (Savelsbergh, 1997). Note-se que, muitas vezes, a motivação para a aplicação da decomposição não é acelerar a resolução da relaxação linear, já que, a relaxação linear do problema reformulado pode ser mais difícil que a relaxação linear do problema original, mas sim a obtenção de limites (inferiores/superiores) mais apertados.

(ii) A formulação original pode apresentar simetria que provoca a falta de efectividade das partições efectuadas. Isto significa que quando uma solução fraccionária é excluída num qualquer nodo da árvore de pesquisa, aparece novamente num outro nodo da árvore de pesquisa com diferentes valores das variáveis, tornando praticamente inútil a partição efectuada.

(iii) A decomposição do problema num problema principal e subproblemas pode ter uma interpretação natural no contexto do problema em estudo permitindo a incorporação de novas e importantes restrições no subproblema.

(iv) Não ser possível obter uma formulação com um número reduzido de variáveis.

A programação de máquinas constitui um bom exemplo de duas formulações diferentes para o mesmo problema. As primeiras formulações para este problema adoptaram variáveis de decisão que representavam o instante de tempo em que o trabalho *j* era iniciado na máquina *k*. Uma formulação alternativa passa por considerar variáveis de decisão associadas a sequências de afectação admissíveis dos trabalhos às máquinas para depois as combinar de forma a obter um programa admissível.

A primeira destas formulações apresenta desvios significativos entre os valores da função objectivo da relaxação linear do problema de programação inteira e da solução óptima inteira. A segunda formulação requer a geração das sequências de afectação admissíveis como *input* do problema sendo que o seu número pode ser muito grande. Contudo, apesar de o problema poder apresentar uma maior dimensão, conduz tipicamente a desvios menores entre os valores da função objectivo para a relaxação linear do problema de programação inteira e o valor da solução óptima inteira.

Diz-se que uma formulação é mais forte (ou de melhor qualidade) quando este desvio é menor. É sabido (Nemhauser e Wolsey, 1988) que a qualidade das formulações é um factor crucial no processo de resolução de problemas de programação inteira através do método de partição e avaliação. Assim, a adopção de formulações mais fortes permite a abordagem de problemas de maior dimensão.

Para o problema em estudo, para além desta razão existe uma segunda razão para a reformulação; o facto da expressão de cálculo do custo ser não linear e a utilização de métodos convencionais de linearização conduzir normalmente a formulações fracas.

3.2. Decomposição de Dantzig-Wolfe

Muitos dos problemas de programação inteira (e as respectivas relaxações lineares) encontrados em muitas das suas aplicações práticas exibem estruturas em que as matrizes dos coeficientes tecnológicos são tipicamente dispersas (*sparse*), com um número de elementos não nulos de uma ordem de grandeza de 1%, ou menos, devido ao

facto de muitas das actividades associadas às variáveis de decisão apenas participarem directamente num pequeno número de restrições. Também é muitas vezes possível identificar uma segmentação hierárquica, geográfica ou lógica na estrutura do problema em que os elementos não nulos se encontram agrupados formando subsistemas independentes de variáveis, possivelmente ligados por um conjunto distinto de restrições e/ou variáveis. Este tipo de estrutura designa-se habitualmente por estrutura angular em blocos.

A ideia geral do método de decomposição de Dantzig-Wolfe é a de tratar a estrutura de ligação a um nível superior, de coordenação, e o(s) subsistema(s) independente(s) a um nível inferior explorando a sua estrutura.

Consideremos o seguinte problema geral com dois conjuntos de restrições e restrições de integralidade:

$$\begin{array}{ll} \min & c(x) \\ \text{suj. a} & Ax \le b \\ & x \in P \\ & x \text{ inteiro.} \end{array} \qquad (3.1)$$

A função c(x) não necessita de ser linear. O número de variáveis e o número de restrições deste modelo podem ser limitados por uma função polinomial do número de tarefas e máquinas do problema. Por essa razão, esta formulação é conhecida na literatura por formulação compacta (*compact formulation*).

No método da decomposição de Dantzig-Wolfe, as restrições do problema original são divididas em dois conjuntos: o primeiro conjunto engloba as restrições gerais $Ax \le b$, enquanto o segundo, $x \in P$, contém um conjunto de restrições que normalmente têm uma estrutura especial. O primeiro conjunto de restrições define o problema principal que é habitualmente designado na literatura anglo-saxónica por *master problem*. O segundo conjunto define um ou vários subproblemas.

Seja X o conjunto de pontos inteiros pertencentes a P. De acordo com o Teorema de Minkowski e Weyl (ver Nemhauser e Wolsey, 1988), a envolvente convexa de X,

designada por *conv*(*X*), é um poliedro que pode ser definido por um conjunto finito de pontos extremos e um conjunto finito de raios extremos. Vamos considerar que o poliedro X tem I pontos extremos, que serão designados por $X_1,X_2,\ldots,X_I$, e K raios extremos, que serão designados por $R_1,R_2,\ldots,R_k$. Qualquer ponto x do conjunto X pode ser expresso como uma combinação convexa dos pontos extremos de X mais uma combinação não-negativa dos raios extremos de X:

$$X = \left\{ x \in P \cap Z_+^n : \sum_{i=1}^{I} \lambda_i X_i + \sum_{k=1}^{K} \mu_k R_k, \sum_{i=1}^{I} \lambda_i = 1, \lambda_i \geq 0, \forall i, \mu_k \geq 0, \forall k \right\} \tag{3.2}$$

No que se segue, vamos considerar que o conjunto X é um conjunto fechado e, como tal, qualquer ponto x do conjunto X pode ser expresso como uma combinação convexa dos pontos extremos de X,

$$x = \sum_{i=1}^{I} \lambda_i X_i, \sum_{i=1}^{I} \lambda_i = 1, \lambda_i \geq 0, \forall i \tag{3.3}$$

Se substituirmos x pela expressão (3.3) no modelo (3.1), obtemos a seguinte formulação equivalente do problema:

$$\begin{aligned} \min \quad & \sum_{i=1}^{I} c(X_i)\lambda_i \\ \text{suj. a} \quad & \sum_{i=1}^{I} (AX_i)\lambda_i \leq b \\ & \sum_{i=1}^{I} \lambda_i = 1 \\ & \lambda_i \geq 0 \\ & x = \sum_{i=1}^{I} \lambda_i X_i \\ & x \text{ inteiro.} \end{aligned} \tag{3.4}$$

As variáveis de decisão do problema reformulado são as variáveis λ_i. Tipicamente, esta formulação tem um maior número de variáveis e um menor número de restrições que o modelo original. Por este motivo, é habitualmente designada por formulação extensiva

(*extensive formulation*). O termo cX_i representa os coeficientes da função objectivo e as colunas AX_i são os coeficientes tecnológicos das restrições.

A equação $\sum_{i=1}^{I} \lambda_i = 1$ é referida como sendo a restrição de convexidade referente aos pontos extremos de *conv*(*X*).

Este problema é o problema principal, que é equivalente à formulação original. Qualquer solução válida em termos das variáveis do problema principal pode ser traduzida numa solução válida para o problema original.

De uma forma geral, requerendo a integralidade das variáveis de decisão λ_i não conduz a um problema de programação inteira equivalente ao problema original. Como as variáveis originais x têm de ser inteiras, são usadas como fonte de informação para guiar decisões de partição que conduzem à solução óptima inteira do problema original.

Quando relaxamos a restrição de integralidade para x, a formulação (3.4) torna-se separável em x e λ e é possível também relaxar a sua restrição de ligação $x = \sum_{i=1}^{I} \lambda_i X_i$, obtendo um problema de programação linear nas variáveis de decisão λ_i. Este problema de programação linear é resolvido por um procedimento de geração de colunas que é descrito na próxima Secção.

3.3. Geração de Colunas

O número de pontos extremos de um poliedro é exponencialmente grande, sendo impraticável fazer a sua enumeração completa. Contudo, existem muitos pontos extremos que correspondem a variáveis que não são atractivas para o problema principal e, como tal, não precisam de ser nele incluídos para determinar a solução óptima. Dizemos que os pontos extremos não são enumerados explicitamente, mas implicitamente.

A relaxação linear do problema principal vai ser resolvida através de um procedimento de enumeração implícita, o algoritmo de geração de colunas: começamos com apenas

um subconjunto restrito das variáveis de decisão (colunas), formando o problema primal restrito (*Restricted Master Problem*), sendo que as restantes são tratadas de uma forma implícita.

O problema primal restrito (PPR) "coordena" a geração de soluções em cada um dos subproblemas. Estes dois problemas são resolvidos alternadamente, trocando informação entre eles, até se atingir uma solução que corresponde à solução do problema original.

Em cada iteração, a avaliação dos pontos extremos é feita com base na informação dual da solução óptima do problema primal restrito. Sejam $\pi_1, \pi_2, \ldots, \pi_m$, e υ as variáveis duais associadas às primeiras m restrições e à restrição de convexidade, respectivamente. A solução dual óptima é dada pelo vector $c_B B^{-1} = (\pi_1, \pi_2, \ldots, \pi_m, \upsilon)$.

Cada variável de decisão do problema reformulado corresponde a um ponto extremo do conjunto X e pode ser caracterizado por uma coluna $A_i = (AX_i)$ de coeficientes tecnológicos, e pelo coeficiente de custo $c_i = (cX_i)$. O seu custo reduzido é dado pela expressão,

$$c_B B^{-1} A_i - c_i = \pi A_i + 1 * \nu - c_i = \pi(AX_i) + 1 * \nu - (cX_i) = (\pi A - c)X_i + \nu$$

O valor mínimo do custo reduzido pode ser determinado resolvendo o seguinte subproblema:

$$\begin{aligned} &\min\ (\pi A - c)X_i + \nu \\ &\text{suj. a: } X_i \in X \end{aligned}$$

Se este valor for negativo, a coluna correspondente ao ponto extremo óptimo é uma coluna atractiva para o problema primal restrito.

A este processo chamamos processo de geração de colunas. A articulação do problema primal restrito e do(s) subproblema(s) é feita do modo descrito na Figura 3.1.

O problema primal restrito, que é resolvido através do método simplex, fornece ao(s) subproblema(s) a solução dual óptima do problema que é necessária para a avaliação das colunas. Por sua vez, o(s) subproblema(s) propõe ao problema primal restrito colunas com custo reduzido negativo que designaremos por colunas atractivas. Caso o óptimo do(s) subproblema(s) não produza uma coluna atractiva, então nenhuma outra solução é atractiva, pelo que a solução corrente do problema primal restrito é óptima.

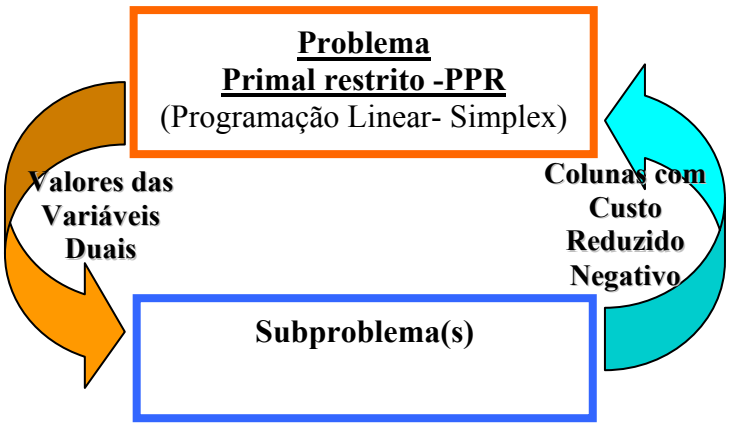


Figura 3.1 – Algoritmo de geração de colunas

A solução óptima do problema primal restrito é também a solução óptima da relaxação linear do problema principal. Se a solução é inteira a solução é óptima para o problema original. Caso contrário, a solução óptima inteira para o problema original é determinada associando ao algoritmo de geração de colunas o método de partição e avaliação que é descrito na Secção seguinte.

3.4. Método de Partição e Avaliação

Para problemas de programação inteira, como é o caso do problema em estudo, não são conhecidas condições gerais de optimalidade como existem para a programação linear (Murty, 1995). Uma das formas de obter soluções óptimas inteiras envolve a

enumeração do conjunto de soluções possíveis. Um dos métodos mais usados para efectuar essa enumeração é o método de partição e avaliação (*branch and bound*). Neste processo, muitas soluções são enumeradas implicitamente, por ser possível reconhecer que elas não podem ser soluções óptimas, pelo que este método é também designado por método de enumeração implícita.

O método de partição e avaliação é um processo de enumeração usado para determinar a solução óptima de problemas de programação inteira. O processo envolve a solução de um problema que constitui uma relaxação do problema original e é baseado nas operações de partição (*branch*) e de avaliação (*bound*).

A partição consiste na criação de subproblemas, feita através da imposição de restrições adicionais. Trivialmente, as restrições são do tipo $x_j \leq \lfloor x_j \rfloor$ e $x_j \geq \lceil x_j \rceil$. Cada subproblema dá, por sua vez, origem a outros subproblemas, sendo criada uma árvore de pesquisa neste processo de enumeração, cujos nodos correspondem aos subproblemas criados. Para um dado nodo da árvore de pesquisa, a regra de partição deve eliminar a solução fraccionária do respectivo subproblema, e garantir que nenhuma das soluções válidas do problema original é perdida. O incumbente corresponde ao valor da melhor solução inteira encontrada até à iteração actual do processo de partição e avaliação. Cada nodo da árvore de pesquisa corresponde a um espaço de soluções válidas, definido pelas restrições lineares do problema e pelas restrições de partição que foram impostas nos ramos da árvore de pesquisa que definem o caminho que vai desde a raiz até esse nodo.

A operação de avaliação destina-se a analisar o interesse em prosseguir a exploração de uma dada sub-árvore. As seguintes situações permitem abandonar a pesquisa de uma sub-árvore:

1. o problema com a restrição adicional que foi imposta no nodo é impossível;
2. o valor da solução obtida para o nodo garante que não é possível obter soluções válidas de melhor qualidade que a do incumbente;
3. a solução obtida no nodo é inteira e é pior que o incumbente.

Se alguma destas situações se verificar o nodo é abandonado porque a exploração dos seus descendentes nunca fornecerá uma solução melhor que o incumbente. Caso contrário, o processo de partição continua e são criados novos nodos na árvore de pesquisa.

A selecção do próximo nodo a avaliar é feita usando regras que determinam o modo como os nodos da árvore de pesquisa são percorridos. Destacamos as seguintes regras de pesquisa:

- Pesquisa em profundidade (*depth first search*) é um método em que são explorados em primeiro lugar os nodos com uma maior profundidade em relação à raiz da árvore de pesquisa. Depois de cada partição é explorado o nodo descendente mais à esquerda.

- Pesquisa em largura (*breadth first search*) é um método em que são explorados em primeiro lugar todos os nodos de um mesmo nível de profundidade antes de passar aos nodos do nível seguinte, começando pelo nodo mais à esquerda.

- Pesquisa do melhor (*best first*) é um método em que são explorados em primeiro lugar os nodos com o valor de solução mais próximo do valor da solução na raiz da árvore de pesquisa.

Estas regras simples podem ainda ser combinadas e dar lugar a regras de pesquisa compostas. Um exemplo de regra de pesquisa composta é a regra *depth first search* mais *best first* que funciona da seguinte forma: se o nodo em avaliação não é abandonado é usada a regra *depth first search* para seleccionar o próximo nodo da árvore de pesquisa a ser avaliado; caso contrário, é usada a regra de *best first*.

3.5. Método de Partição e Geração de Colunas

Em modelos que resultam de uma decomposição de Dantzig-Wolfe que são resolvidos por geração de colunas, uma das formas de obter uma solução inteira consiste em

associar o método de partição e avaliação ao método de geração de colunas. O método resultante desta associação é designado por método de partição e geração de colunas.

O processo inicia-se com a resolução da relaxação linear do problema principal. Quando os subproblemas não geram mais colunas com custo reduzido negativo, o algoritmo Simplex fornece a solução óptima da relaxação linear do problema principal. Se a solução é inteira a solução é óptima para o problema original. Caso contrário, necessitamos de explorar uma árvore de pesquisa, e podem ser geradas colunas adicionais em cada partição. O nodo raiz corresponderá ao problema de programação linear original (relaxação das condições de integralidade). Os seus descendentes corresponderão ao mesmo problema adicionado de restrições de partição.

O fluxograma do método de partição e geração de colunas é apresentado na Figura 3.2

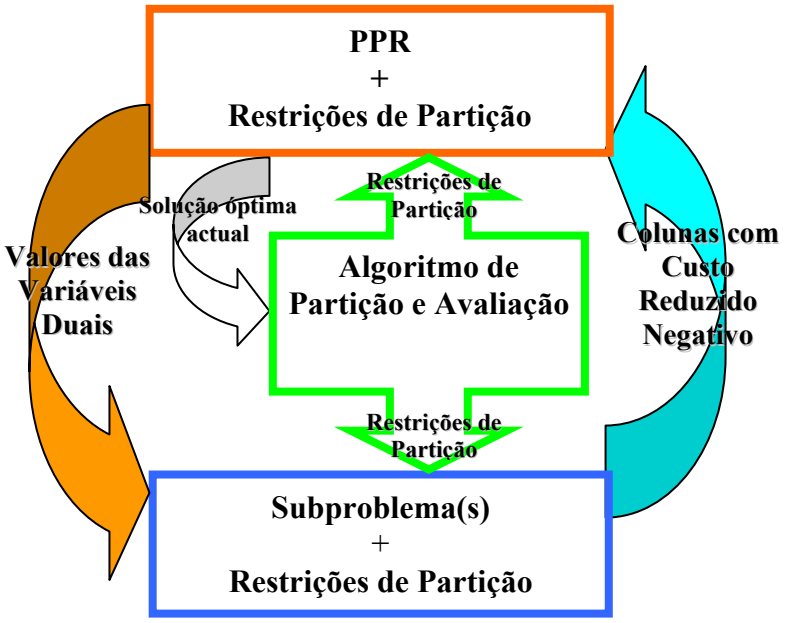


Figura 3.2 – Algoritmo de partição e geração de colunas

O problema primal restrito fornece a solução óptima actual ao algoritmo de partição e avaliação que vai verificar se foi encontrada a solução óptima inteira do problema

principal. Neste caso, o processo termina; caso contrário, vão ser geradas novas restrições de partição que vão ser adicionadas ao problema primal restrito e ao(s) subproblema(s). O novo problema primal restrito é reoptimizado e fornece a solução dual óptima ao(s) subproblema(s) que, por sua vez, propõem ao problema primal restrito colunas atractivas. Caso o óptimo do(s) subproblema(s) não produza uma coluna atractiva, então nenhuma outra solução é atractiva, pelo que a solução corrente do problema primal restrito é óptima e o processo é reiniciado.

Numa primeira abordagem, poderá parecer que a combinação dos métodos de geração de colunas e de partição e avaliação não envolve mais do que a junção dos dois métodos: em cada nodo da árvore de pesquisa do método de partição e avaliação, resolve-se o problema de programação linear até à optimalidade, utilizando o método de geração de colunas (se não fosse permitida a geração de colunas nos nodos da árvore, poder-se-ia concluir erradamente que não existia uma solução inteira para o problema ou obter uma solução não óptima).

No entanto, o problema a resolver em cada nodo da árvore inclui restrições de partição que não estavam presentes na relaxação linear do problema original. Estas restrições podem alterar ou destruir a estrutura do subproblema, tornando impraticável a avaliação da atractividade das variáveis que não fazem parte do problema primal restrito a resolver em cada nodo de pesquisa.

Assim, o desafio que se coloca ao aplicar o método de partição e geração de colunas a um dado problema é o de definir regras de partição compatíveis com o subproblema. Soumis (1997) apresenta uma revisão bibliográfica anotada sobre métodos de decomposição e geração de colunas.

Capítulo 4

Revisão da Literatura

Neste capítulo, apresentamos uma revisão da literatura sobre máquinas paralelas focada principalmente nos métodos de solução exacta, em geral, e no método de partição e geração de colunas, em particular. Também se abordam problemas de máquina única porque eles ocorrem nos subproblemas que resultam da decomposição de Dantzig-Wolfe. Antes de abordar os métodos de solução exacta, fazemos uma muito breve abordagem aos métodos heurísticos.

4.1. Introdução

O problema de programação de máquinas paralelas tem merecido uma atenção considerável por parte da comunidade científica (para uma revisão dos problemas de programação de máquinas paralelas ver Cheng e Sin, 1990, e Hoogeven, Lenstra e van de Velde 1997). O interesse por este tipo de problemas provém do facto de eles serem frequentes em ambientes industriais e de possuírem uma grande complexidade (NP-difíceis). Por este motivo, foram muitas vezes propostos métodos heurísticos para a sua resolução, geralmente baseados em regras de decisão. Foram também desenvolvidos alguns procedimentos de solução exacta, seguindo métodos de enumeração, mas que se mostraram incapazes de resolver problemas de dimensão real. Mais recentemente, a utilização de métodos de enumeração implícita, em particular o método de partição e geração de colunas, permitiu o desenvolvimento de algoritmos de solução exacta para problemas de grande dimensão.

4.2. Métodos Heurísticos

Os métodos heurísticos mais usados para a resolução de problemas de programação de máquinas paralelas são as regras de decisão, os métodos de pesquisa local e as técnicas de enumeração parcial. Segue-se uma abordagem muito genérica a este tipo de métodos. Uma abordagem mais completa pode ser encontrada em Morton e Pentico (1993), Pinedo e Chao (1999), Pinedo (2001) e Leung (2004).

4.2.1. Métodos Heurísticos baseados em Regras de Decisão

Os métodos heurísticos baseados em regras de decisão constroem programas de uma forma sequencial seleccionando a próxima tarefa usando regras de decisão relacionadas com as máquinas ou com as tarefas. Por exemplo, podemos considerar os tempos de processamento das tarefas (*SPT – Shortest Processing Time*, etc.), a disponibilidade das máquinas (*LFM- Least Flexible Machine*, etc.), etc.

Baker e Merten (1973) propõem e comparam regras de decisão para o problema de programação de máquinas paralelas. Panwalker e Iskander (1977) citam 113 regras de selecção de uma actividade que foram propostas ou que são aplicadas na indústria. Uma actualização pode ser encontrada em Panwalker (1990).

Pinedo (1995) generalizou uma condição de dominância estabelecida por Emmons (1969), para o problema $1||\sum T_j$, ao problema $1||\sum w_j T_j$: se existirem dois trabalhos *i* e *j* com $d_i \leq d_j$, $p_i \leq p_j$ e $w_i \geq w_j$, então existe uma sequência óptima em que o trabalho *i* aparece antes do trabalho *j*.

As regras de decisão mais complexas são geralmente construídas a partir de regras simples. Uma descrição detalhada de regras de decisão compostas pode ser encontrada em Bhaskaran e Pinedo (1992). Alguns exemplos são apresentados de seguida.

Lee, Bhaskaran, e Pinedo (1997) propõem uma heurística baseada numa composição de várias regras de decisão para o problema $1|s_{ij}|\sum w_j T_j$ - a heurística ATCS (*Apparent*

Tardiness Cost with Setups). Esta heurística combina as regras WSPT (*Weighted Shortest Processing Time first*) que processa os trabalhos em ordem não crescente dos rácios w_j/p_j, MS (*Minimum Slack first*) que processa os trabalhos em ordem não decrescente da folga entre a data de entrega e data actual, e SST (*Shortest Setup Time first*) que processa os trabalhos em ordem não decrescente dos tempos de preparação.

A heurística funciona do seguinte modo. É programado um trabalho de cada vez; sempre que a máquina fica disponível, é calculado um índice para os trabalhos ainda não programados e é seleccionado o trabalho com o maior valor do índice. O índice referente ao processamento do trabalho *j* logo após a conclusão do processamento do trabalho *i* no instante de tempo *t* é calculado através da seguinte expressão:

$$I_j(t,i) = \frac{w_j}{p_j}\exp\left(-\frac{\max(d_j - p_j - t, 0)}{k_1\overline{p}}\right)\exp\left(-\frac{s_{ij}}{k_2\overline{s}}\right)$$

em que $\overline{p}$ e $\overline{s}$ são as médias dos tempos de processamento e de preparação, respectivamente, dos trabalhos que ainda não foram programados, k_1 um parâmetro de escala relativo às datas de entrega e k_2 um parâmetro de escala relativo aos tempos de preparação.

A escolha destes parâmetros visa uma adaptação da heurística a diferentes condições, em função de três factores:

(i) O factor de aperto das datas de entrega $\tau = 1 - \overline{d}/\hat{C}_{\max}$, onde $\overline{d}$ representa a média das datas de entrega. Valores de τ próximos de 1 indicam que as datas de entrega são apertadas e valores próximos de 0 indicam que as datas de entrega são largas.

(ii) O factor de intervalo das datas de entrega $R = \left(d_{\max} - d_{\min}\right)/\hat{C}_{\max}$. Um valor elevado de R indica um intervalo largo de datas de entrega enquanto um valor baixo indica um intervalo apertado de datas de entrega.

(iii) O factor de severidade dos tempos de preparação $\eta = \overline{s}/\overline{p}$.

O valor do *makespan* é dependente do programa já que os tempos de preparação dependem da sequência de processamento dos trabalhos. Uma forma simples de o estimar é a seguinte: $\hat{C}_{\max} = \sum_{j=1}^{n} p_j + n\bar{s}$.

Para obter bons resultados, os valores de k_1 e k_2 devem ser adequados à instância particular do problema. Isto pode ser feito através de uma análise estatística preliminar aos dados da instância. Contudo, um estudo experimental desta regra, embora inconclusivo, sugeriu as seguintes regras para a selecção dos parâmetros k_1 e k_2:

- k_1=4.5+R, R≤0.5 e k_1=6-2R, R≥0.5
- $k_2 = \tau/(2\sqrt{\eta})$

Esta regra pode também ser aplicada ao problema de máquinas paralelas $P|s_{ij}|\sum w_j T_j$. Neste caso, os parâmetros k_1 e k_2 são também dependentes do número de máquinas m.

Em certos casos, é possível determinar um limite superior para o erro máximo do algoritmo heurístico. Graham, Lawler, Lenstra e Rinnooy Kan (1979) apresentam estes limites para vários tipos de problemas e diferentes heurísticas.

4.2.2. Métodos Heurísticos baseados em Pesquisa Local

Os métodos heurísticos baseados em pesquisa local verificam se é possível melhorar um programa já existente pesquisando na vizinhança da solução actual. Wilkerson e Irwin (1971) utilizaram este método para resolver o problema $1||\sum w_j C_j$. Ho e Chang (1991) estenderam este trabalho ao problema de máquinas paralelas idênticas. Van de Velde (1991) desenvolveu uma heurística dual para o problema de programação de máquinas paralelas em que a vizinhança é explorada segundo as direcções de pesquisa obtidas através dos multiplicadores de Lagrange.

Os métodos de Tabu-Search e Simulated Annealing efectuam igualmente uma pesquisa na vizinhança da solução, mas permitem a possibilidade de movimentos para soluções

vizinhas piores do que a actual. A diferença fundamental entre estes dois métodos tem a ver com o mecanismo usado para aprovar os movimentos candidatos. No Tabu-Search este mecanismo é baseado num processo de natureza determinística enquanto que no Simulated Annealing é baseado num processo de natureza probabilística (Pinedo, 1995).

França, Gendreau, Laporte e Muller (1996) usaram o Tabu-Search para a resolução do problema de minimização do *makespan* em máquinas paralelas com tempos de preparação dependentes da sequência. Potts e Van Wassenhove (1991) e Tan, Narasinham, Rubin e Ragatz (2000), comparam o Simulated Annealing com outras heurísticas para a resolução de problemas $1||\sum w_j T_j$.

Tan e Narasinham (1997), propuseram o Simulated Annealing para a resolução do problema de minimização dos atrasos em uma máquina com tempos de preparação dependentes da sequência. Koulamas, Antony e Jaen (1994) e Barnes (1993) apresentam revisões da utilização dos métodos Simulated Annealing e Tabu-Search, respectivamente, no domínio da programação.

Os Algoritmos Genéticos constituem uma classe de técnicas de pesquisa local mais gerais e mais abstractas do que os métodos de Tabu-Search e Simulated Annealing (Pinedo, 1995). Nos Algoritmos Genéticos, as sequências de afectação ou programas são vistos como indivíduos que são membros de uma população.

O procedimento funciona de uma forma iterativa e em cada iteração é criada uma nova geração de indivíduos que contém indivíduos sobreviventes da anterior geração e novos indivíduos (novas soluções) obtidos a partir da anterior geração. Normalmente, o número de indivíduos da população mantém-se constante. Em cada geração, há um processo de reprodução em que as características dos melhores indivíduos (melhores soluções) são usadas para gerar novos elementos da população que substituem os piores indivíduos (piores soluções), que são eliminados. Este processo de determinação da composição da próxima geração pode ser complexo.

Rubin e Ragatz (1995) aplicaram esta técnica a um conjunto de problemas de programação de máquina-única num ambiente de tempos de preparação dependentes da

sequência. Cheng e Gen (1997) e França, Mendes e Moscato (1999) usaram uma classe de algoritmos genéticos híbridos (*Memetic Algorithms*) para resolverem problemas de programação de máquinas paralelas. Mendes, Muller, França e Moscato (1999) comparam a técnica de Tabu-Search com os Memetic Algorithms na resolução do problema de minimização do *makespan* em máquinas paralelas idênticas com tempos de preparação dependentes da sequência.

4.2.3. Métodos Heurísticos baseados em Enumeração Parcial

Os métodos heurísticos de enumeração parcial são geralmente baseados no método de partição e avaliação mas onde nem todos os nodos de um determinado nível da árvore de pesquisa são avaliados. Só os nodos mais promissores são seleccionados para partição. Os restantes são eliminados de uma forma definitiva. A avaliação e selecção dos nodos mais promissores, em cada nível da árvore de pesquisa, é feita através de um procedimento heurístico. Muitos dos métodos de enumeração propostos para a solução exacta de problemas podem ser utilizados para gerar soluções heurísticas quando são terminados prematuramente. A qualidade da solução obtida depende do número de possibilidades exploradas e das direcções de pesquisa escolhidas.

Morton e Ramnath (1992) descrevem um procedimento de enumeração parcial, que explora apenas certas porções da árvore de pesquisa com base numa função de avaliação aproximada dos subproblemas de um mesmo nível (*Guided Search*), para a resolução aproximada de problemas de minimização dos atrasos em máquina única.

4.3. Métodos Exactos

Nesta Secção começamos por fazer uma breve abordagem a dois métodos gerais de solução exacta, a programação dinâmica e o método de partição e avaliação (para uma abordagem mais completa ver Cheng e Sin, 1990 e Hoogeven, Lenstra e van de Velde,

1997), para depois abordar de uma forma mais completa o método de partição e geração de colunas.

A programação dinâmica e o método de partição e avaliação são muitas vezes usados para determinar soluções exactas para problemas de pequena e média dimensão para medir a qualidade de vários tipos de heurísticas.

4.3.1. Programação Dinâmica

Na programação dinâmica, são construídos sequencialmente programas utilizando princípios de optimalidade. Os programas parciais são comparados e as regras de dominância permitem eliminar aqueles que não conduzem à solução óptima. Foram desenvolvidas algumas aplicações para problemas de programação de uma máquina e propostas generalizações para o problema de máquinas paralelas. Contudo, estas últimas são de difícil aplicação dado o elevado número de estados que são gerados.

Rothkopf (1966) apresentou equações de recorrência para problemas de máquinas idênticas e tarefas independentes. Lawler e Moore (1969) consideram vários critérios de optimalidade para o problema de máquina única e discutem as generalizações para o caso das máquinas paralelas. Os algoritmos pseudo-polinomiais desenvolvidos podem ser resolvidos em tempo $O\left(n\left(\sum_{j=1}^{n} p_j\right)^{m-1}\right)$, tornando-se impraticáveis quando m ou $\sum_{j=1}^{n} p_j$ crescem. Sahni (1976) propõe um algoritmo para o problema de m máquinas paralelas idênticas e o objectivo C_{max}. Leung (1982) considera o problema de minimização do *makespan* para m máquinas paralelas idênticas e tarefas com tempos de processamento restritos a k valores e apresenta um algoritmo de complexidade $O(\log p_{\max} \log m n^{2(k-1)})$.

4.3.2. Método de Partição e Avaliação

Para além das técnicas de programação dinâmica, existem na literatura técnicas de cálculo de limites inferiores que podem ser usadas em algoritmos de partição e

avaliação. O método de partição e avaliação consiste na divisão do espaço de soluções em subespaços mais pequenos. São calculados limites (superiores ou inferiores) para o valor da solução óptima que permitem eliminar partes da árvore de pesquisa.

Os procedimentos propostos na literatura diferem sobretudo ao nível das estratégias de partição e pesquisa e no cálculo dos limites. Eastman, Even e Isaacs (1964), propuseram um limite inferior para o valor óptimo que serviu de base aos algoritmos de partição e avaliação de Elmaghraby e Park (1974), Barnes e Brennan (1977) e Sarin, Ahn e Bishop (1988). Em particular, o algoritmo de Barnes e Brennan (1977) para o problema $P||\sum w_j T_j$ mostrou-se capaz de resolver problemas até 20 trabalhos e 4 máquinas.

Mais recentemente, Webster (1992, 1993, 1995) propôs limites inferiores para o valor óptimo baseado na ideia de considerar um trabalho como uma colecção de subtrabalhos ligados por uma restrição de grupo. Contudo, não são conhecidos algoritmos de partição e avaliação baseados nestes limites inferiores.

Ragatz (1993) aplicou este método ao problema de minimização dos atrasos numa máquina única com tempos de preparação dependentes da sequência, conseguindo obter soluções óptimas apenas para pequenas instâncias.

Belouadah e Potts (1995) formularam este problema como um problema de programação inteira e desenvolveram um algoritmo de partição e avaliação, baseado num esquema de relaxação Lagrangiana para calcular limites inferiores, que se mostrou capaz de resolver problemas de dimensão até 20 trabalhos e 5 máquinas ou 30 trabalhos e 4 máquinas.

4.3.3. Método de Partição e Geração de Colunas

Nas Subsecções seguintes fazemos uma revisão da literatura mais recente da aplicação deste método a problemas de programação de máquinas paralelas, tentando fazer o seu enquadramento numa estrutura cronológica de desenvolvimento de algoritmos de solução exacta. Como, na literatura, este problema tem sido estudado considerando diferentes tipos de máquinas, diferentes parâmetros para as tarefas a processar e

diferentes critérios de optimalidade, apresentamos esta revisão de literatura organizada por classes de problemas começando por aquele que é o mais comum: o problema de programação de máquinas paralelas idênticas com o objectivo de minimizar a soma ponderada dos tempos de conclusão, $P||\sum w_j C_j$.

4.3.3.1. Problema $P||\sum w_j C_j$

Smith (1956) mostrou que o problema $1||\sum w_j C_j$ pode ser resolvido em tempo de ordem $O(n \log n)$. Este algoritmo é baseado na observação de que um programa óptimo possui as duas seguintes propriedades:

- Os trabalhos são processados continuamente a partir de t=0.
- Os trabalhos estão sequenciados em ordem não crescente dos rácios w_j/p_j.

Como para o caso de múltiplas máquinas em paralelo as máquinas são independentes entre si, a partir do momento em que o conjunto de trabalhos é subdividido pelas várias máquinas sabemos que existe um programa óptimo em que cada máquina processa os trabalhos por ordem não crescente dos rácios w_j/p_j e de uma forma contínua (sem tempos espera). Isto implica que a dificuldade associada à resolução deste problema tem a ver com a determinação de uma subdivisão óptima dos trabalhos pelas máquinas.

Van den Akker, Hoogeveen e van de Velde (1999) e Chen e Powell (1999a), propuseram procedimentos de partição e avaliação onde o limite inferior é obtido através da resolução da relaxação linear de uma formulação de programação inteira através de um algoritmo de geração de colunas. Os resultados computacionais mostraram que os limites inferiores obtidos na resolução da relaxação linear do problema são de grande qualidade, apresentando muitas vezes soluções óptimas inteiras. Quando a solução obtida na relaxação linear não é inteira, são impostas restrições de partição sobre as colunas geradas que podem ser facilmente incorporadas no algoritmo do subproblema. Os procedimentos propostos são idênticos divergindo apenas nas estratégias de partição.

A estratégia de partição proposta por Van den Akker, Hoogeveen e van de Velde (1999) é baseada na partição do intervalo de tempo de processamento dos trabalhos. Para cada trabalho é definida uma janela de processamento baseada no seguinte teorema:

Teorema **4.1** (Elmaghraby e Park, 1974) *Existe um programa óptimo quando se verificam as seguintes propriedades:*

- *Os trabalhos são processados continuamente a partir de t=0 e nenhuma máquina está desocupada antes de todos os trabalhos estarem disponíveis.*
- *O último trabalho a ser processado, em qualquer uma das máquinas, é concluído entre o tempo* $H_{\min} = \sum_{j=1}^{n} p_j / m - (m-1)p_{\max} / m$ *e o tempo* $H_{\max} = \sum_{j=1}^{n} p_j / m + (m-1)\, p_{\max} / m$ *onde* $p_{\max} = \max_{j \in N} \{p_j\}$.
- *Em cada máquina, os trabalhos estão sequenciados em ordem não crescente dos rácios* w_j/p_j.
- *Se* $w_i \geq w_j$ *e* $p_i \leq p_j$, *onde pelo menos uma das desigualdades é estrita, então existe um programa óptimo em que o processamento do trabalho i nunca é iniciado depois do início do processamento do trabalho j.*

Em cada nodo da árvore de pesquisa, são identificados os trabalhos com valor fraccionário e, caso existam, são criados dois nodos descendentes para o trabalho fraccionário de menor índice: o primeiro impõe que $C_j \leq C_j^{\min}$ e o segundo impondo que $C_j \geq C_j^{\min} + 1$. A primeira condição define uma nova data limite de processamento para o trabalho *j*; a segunda condição define uma nova data de disponibilidade para o trabalho *j*, que é superior à actual.

Esta estratégia de partição não só reduz a janela temporal de execução do trabalho *j*, como, com base na propriedade (iv), pode reduzir as janelas temporais de outros trabalhos. O mais interessante nesta estratégia de partição é que as condições impostas são facilmente incorporadas no algoritmo do subproblema, que pode ser resolvido em tempo $O\left(n\sum_{j=1}^{n} p_j\right)$, não existindo a necessidade de modificar as equações de recorrência.

Contudo, pode acontecer que as janelas temporais de processamento dos trabalhos correspondentes a um dado nodo da árvore de pesquisa conduzam a uma solução impossível. Como o problema de decidir se existe ou não um programa que obedeça às condições impostas é NP-completo, apenas é verificada a condição necessária para a existência de uma solução admissível. As datas limite de processamento dos trabalhos são substituídas por datas de entrega e é calculado um limite inferior para o problema de optimização $P|r_j|L_{\max}$ usando o procedimento proposto por van de Velde, Hoogeveen, Hurkens e Lenstra (1997) que corre em tempo $O(n^2)$. Existe um programa que obedece às condições impostas quando o limite inferior calculado é positivo; caso contrário, o nodo é abandonado.

Também pode acontecer que o actual conjunto de colunas do problema primal restrito não constitua uma solução admissível após a introdução das novas restrições de partição. Este problema é resolvido removendo as colunas correspondentes a sequências de afectação não admissíveis e que não façam parte da actual solução e aumentando o custo das colunas correspondentes a sequências de afectação não admissíveis que façam parte da actual solução para um valor muito elevado, permitindo que exista uma solução admissível mas com um custo muito elevado.

A estratégia de partição proposta por Chen e Powell (1999a) é baseada na sequência de processamento dos trabalhos incluídos em cada sequência de afectação. Assim, dada a solução óptima da relaxação linear do problema de programação inteira, podemos calcular para cada par de trabalhos (i,j), em que o trabalho i é imediatamente precedente do trabalho j, o valor total das variáveis de decisão x_s (colunas) em que o par de trabalhos (i,j) está presente, $s \in S_{ij}$, $y_{ij} = \sum_{s \in S_{ij}} x_s$.

Caso existam pares de trabalhos (i,j) com valor y_{ij} fraccionário, é seleccionada para partição a variável y_{ij} com valor fraccionário mais próximo de 0,5 e são criados dois nodos na árvore de pesquisa: um impondo que o valor y_{ij} é igual a zero e outro impondo que o valor y_{ij} é igual a um. Em ambos os nodos, são removidas do problema primal restrito as colunas que não satisfazem as condições impostas pelas restrições de partição.

Note-se que, na prática, quando impomos que y_{ij}=1, estamos a impor uma relação de precedência entre os trabalhos i e j, e quando impomos que y_{ij}=0 estamos a eliminar do espaço de soluções admissíveis as sequências de afectação que incluem o par de trabalhos (i,j), em que o trabalho i é imediatamente precedente do trabalho j. Consequentemente, o subproblema necessita de ser ajustado para fazer reflectir estas novas condições o que faz aumentar o seu tempo de execução de um factor $O(n)$.

Se bem que não existam estudos comparativos dos tempos computacionais dos dois algoritmos atrás apresentados, podemos referir alguns aspectos gerais relativamente à sua comparação. O algoritmo proposto por Van den Akker, Hoogeveen e van de Velde (1999) apresenta a vantagem de o seu tempo de execução não ser influenciado pela estratégia de partição enquanto que o algoritmo proposto por Chen e Powell (1999a) é mais simples de implementar já que, em cada nodo da árvore de pesquisa, é possível encontrar um conjunto de colunas que constitua uma solução óptima. Finalmente, ambos os algoritmos se mostraram capazes de resolver problemas com n=100 e m=10 em tempo computacional considerado razoável.

Nas subsecções seguintes vamos apresentar outros problemas, indicando apenas as diferenças mais significativas relativamente ao que foi descrito nesta subsecção.

4.3.3.2. Problema $P||\sum w_j U_j$

Para o problema $P||\sum w_j U_j$, existe um programa óptimo quando cada máquina processa, em primeiro lugar, os trabalhos com atraso nulo em ordem decrescente das datas de entrega e, em segundo lugar, os trabalhos com atraso em qualquer ordem (Lawler e Moore, 1969 e Potts e Van Wassenhove, 1992). Os trabalhos com atraso são processados na parte irrelevante do programa, sendo indiferente quando e onde (em que máquina) são processados. Isto significa que o subproblema se reduz à geração de sequências de afectação com atraso nulo em que os trabalhos estão sequenciados por ordem crescente das datas de entrega. Para uma revisão mais detalhada ver Chen e Powell (1999a).

4.3.3.3. Extensão a Máquinas Não-Idênticas

Quando as máquinas são não-idênticas, os tempos de processamento dependem da máquina onde o trabalho é processado. Isto implica que a aplicação das regras de prioridade pode apresentar resultados diferentes para cada uma das máquinas e, consequentemente, necessitamos de tratar cada máquina separadamente, usando o algoritmo do subproblema *m* vezes (uma vez para cada máquina).

Van den Akker, Hoogeveen e van de Velde (1999) e Chen e Powell (1999a) estudaram a extensão dos algoritmos de resolução do subproblema (apresentados na Subsecção 4.3.3.1) aos problemas $R \| \sum w_j C_j$ e $R \| \sum w_j U_j$. Se bem que esses algoritmos possam continuar a ser usados, apresentando agora complexidades de ordem $O\left(\sum_{k=1}^{m} n \sum_{j=1}^{n} p_{jk}\right)$ e $O\left(\sum_{k=1}^{m} n^2 \sum_{j=1}^{n} p_{jk}\right)$, respectivamente, a estratégia de partição proposta por Van den Akker, Hoogeveen e van de Velde (1999) deixa de ser válida.

4.3.3.4. Problemas $P \| \sum \left(u_j E_j + v_j T_j\right)$

Estes problemas são motivados pela filosofia JIT (*just-in-time*) onde é desejável que os trabalhos sejam concluídos o mais próximo possível das suas datas de entrega. Se um trabalho é concluído antes da sua data de entrega (com avanço) necessita de ser armazenado incorrendo num custo de stock. Consequentemente, para além dos atrasos necessitamos de considerar também os avanços na medida de desempenho. Podem também ser atribuídos pesos diferentes aos avanços e atrasos, u_j e v_j, respectivamente.

Existem dois tipos de problemas estudados na literatura: a data de entrega é um ponto no tempo e a data de entrega é um intervalo de tempo. No primeiro modelo, só não existe penalidade quando o trabalho é concluído exactamente na data de entrega. O segundo modelo existe penalidade quando o trabalho é concluído fora do intervalo de tempo associado à data de entrega.

Este segundo modelo, para além de ser mais realista, inclui o primeiro modelo como caso particular quando o intervalo de tempo da data de entrega é um ponto. Uma data de

entrega comum a todos os trabalhos é considerada larga e não restritiva quando $d \geq \sum_{j=1}^{n} p_j$. Da mesma forma, quando $d < \sum_{j=1}^{n} p_j$ a data de entrega comum a todos os trabalhos é considerada estreita e restritiva.

Chen e Powell (1999b) usaram o método de partição e geração de colunas para resolver o problema de programação de *n* trabalhos, com uma data de entrega comum larga e não restritiva *d*, em *m* máquinas paralelas idênticas de forma a minimizar a soma ponderada dos avanços e atrasos. Como a data de entrega é comum, os trabalhos são penalizados quando não são concluídos exactamente nesta data.

O problema foi inicialmente formulado como um problema de programação inteira e depois reformulado, usando a decomposição de Dantzig-Wolfe, como um problema de partição de conjuntos com restrições adicionais. A relaxação linear do problema de programação inteira é resolvida pelo algoritmo de geração de colunas onde as colunas representam sequências de afectação parciais e são geradas resolvendo dois subproblemas de programação de máquina única.

Estes dois subproblemas reflectem a circunstância de que uma sequência de afectação apresenta uma primeira parte em que os trabalhos são processados com avanço e uma segunda parte em que os trabalhos são processados com atraso. Como a data de entrega é larga, existe sempre um trabalho que é concluído exactamente na data de entrega *d*. Existe uma solução óptima em que os trabalhos da primeira parte da sequência de afectação (com avanço) são processados em ordem não decrescente do rácio u_j/p_j, sendo que o último trabalho é concluído na data *d*, e os trabalhos da segunda parte da sequência de afectação (com atraso) são processados em ordem não crescente do rácio v_j/p_j, sendo que o primeiro trabalho inicia o seu processamento na data *d*.

Os dois subproblemas são resolvidos de forma independente até que não seja possível gerar sequências de afectação de custo reduzido negativo para ambos. A solução óptima inteira é obtida através do algoritmo de partição e avaliação. Os resultados computacionais mostram que o algoritmo é capaz de resolver problemas com um máximo de n=60 em tempo computacional considerado razoável.

Chen e Lee (2002) generalizaram o problema de data de entrega comum ao problema de janela de entrega comum. Neste caso, os trabalhos são penalizados apenas quando são concluídos fora da janela de entrega $[d_1,d_2]$. Em contraste com o problema anterior, não é assumido que o limite inferior, d_1, é largo. Consequentemente, não existe garantia de que um trabalho vai concluir o seu processamento na data d_1 ou d_2. As sequências de afectação apresentam três partes:

- Uma sequência de afectação parcial *early* em que os trabalhos são processados com avanço em ordem não decrescente do rácio u_j/p_j, sendo que o último trabalho é concluído antes ou na data d_1;
- Uma sequência de afectação parcial *tardy* em que os trabalhos são processados com atraso em ordem não crescente do rácio v_j/p_j, sendo que o primeiro trabalho inicia o seu processamento na data d_2 ou depois;
- Uma sequência de afectação parcial *intermédia*.

A primeira e a segunda partes podem ser conjuntos vazios.

O problema é formulado como um modelo de partição de conjuntos com restrições adicionais em que se procura encontrar não mais do que m sequências de afectação parciais de cada um dos tipos descritos (*early*, *intermédia* e *tardy*) que possam ser concatenadas para formar um programa admissível, isto é, para cada sequência de afectação parcial *early* que termina em t_1, existe uma sequência de afectação parcial *intermédia* que se inicia em t_1 e termina em t_2 e, da mesma forma, para cada sequência de afectação parcial *intermédia* que termina em t_2 existe uma sequência de afectação *tardy* que se inicia em t_2.

Em cada iteração do algoritmo de geração de colunas são resolvidos três subproblemas que determinam as sequências de afectação parciais *early*, *tardy* e *intermédia*, de menor custo reduzido. De entre elas, a sequência de afectação parcial de menor custo reduzido é inserida no problema primal restrito. A solução óptima é encontrada quando nenhum dos subproblemas consegue encontrar sequências de afectação parciais de custo reduzido negativo.

A estratégia de partição é semelhante à usada por Chen e Powell (1999a) e descrita na Subsecção 4.3.3.1. Os resultados computacionais mostram que o algoritmo desenvolvido é capaz de resolver problemas com n=40 e qualquer número de máquinas em tempo computacional considerado razoável.

4.3.3.5. Problemas de Programação com Tempos de Preparação

Chen e Powell (2003) estudaram uma extensão do problema $P \| \sum w_j C_j$ envolvendo tempos de preparação. Os trabalhos são divididos em várias famílias tal que o processamento consecutivo de trabalhos de uma mesma família não requer qualquer tempo de preparação. Por outro lado, o processamento consecutivo de trabalhos de diferentes famílias requer um tempo de preparação. Este problema é conhecido por problema de programação de múltiplas famílias (MJF – *Multiple Job Family*).

Foram estudadas duas variantes deste problema: tempos de preparação dependentes da sequência de trabalhos a processar, $P \mid s_{uv} \mid \sum w_j C_j$, e tempos de preparação independentes da sequência de trabalhos a processar, $P \mid s_u \mid \sum w_j C_j$. Para o primeiro caso, é assumido que os tempos de preparação são definidos de forma a satisfazer o princípio das desigualdades triangulares, isto é, $s_{ij}+s_{jv} \geq s_{iv}$, para todos os i,j,v. Vamos apenas descrever o caso mais geral de tempos de preparação dependentes da sequência de trabalhos a processar.

O problema foi formulado como um modelo de partição de conjuntos com restrições adicionais e depois resolvido por um algoritmo de partição a avaliação. Em cada nodo da árvore de pesquisa do algoritmo de partição a avaliação, é resolvida a relaxação linear da formulação de partição de conjuntos através de um procedimento de geração de colunas. As colunas são geradas num subproblema que resolve um problema de programação de máquina única através de um algoritmo de programação dinâmica. Este algoritmo é baseado na observação de que um programa óptimo possui as seguintes propriedades:

- **(Monma e Potts, 1989)** Os trabalhos dentro de cada família e em cada máquina são processados numa ordem não crescente dos rácios w_j/p_j.

- **(Chen e Powell, 2003)** Para m=1, $C_j \geq s_{ou} + \sum_{i \in K_j} p_i \quad \forall j \in N_u$, $1 \leq u \leq b$, em que b representa o número de famílias, s_{0u} o tempo de preparação do primeiro trabalho da família u a ser processado na máquina e K_j representa o subconjunto de trabalhos da mesma família, N_u, que precedem o trabalho i (dentro de cada família, os trabalhos estão ordenados numa ordem não crescente dos rácios w_j/p_j).

- **(Chen e Powell, 2003)** $C_j \leq \min(\alpha_j, \beta_j)$, $\forall j \in N_u$, $1 \leq u \leq b$, onde $\alpha_j = \lfloor (S+P)/m + (m-1)(q_u + p_j)/m \rfloor$, $\beta_j = S + P - \sum_{i \in N_u \setminus K_j} (q_u + p_i)$, $P = \sum_{j \in N} p_j$, $S = \sum_{u=1}^{b} n_u q_u$, $q_u = \max_{0 \leq l \leq b} s_{lu}$ e n_u representa o número de trabalhos da família u.

Apesar de estas propriedades permitirem reduzir o espaço de soluções do subproblema, o único algoritmo conhecido capaz de resolver o problema $P|s_{uv}|\sum w_j C_j$ é o algoritmo de programação dinâmica proposto por Ghosh (1994), de complexidade $O\left(b^2 n^b\right)$, e este mostra-se incapaz de resolver problemas para além da muito pequena dimensão. Por este motivo, foi adoptada uma formulação alternativa de complexidade pseudo-polinomial, inicialmente proposta por Desrochers, Desrosiers e Solomon (1992), em que é permitida a selecção de colunas correspondentes a sequências de afectação não admissíveis em que o mesmo trabalho é processado mais do que uma vez. Estas sequências de afectação são chamadas de cíclicas.

A integralidade das variáveis de decisão garante que as soluções inteiras do problema não incluem sequências de afectação cíclicas. A estratégia de partição é semelhante à usada por Chen e Powell (1999a) e descrita na Subsecção 4.3.3.1. Os resultados computacionais mostram que o algoritmo é capaz de resolver problemas até 40 trabalhos para qualquer número de famílias e duas ou mais máquinas.

4.3.3.6. Problemas de Programação de Trabalhos e Actividades de Manutenção

Em muitas situações, as máquinas são alvo de actividades de manutenção periódica para prevenir a ocorrência de avarias. Durante o período de manutenção, a máquina não está disponível para o processamento de trabalhos. A programação de actividades de manutenção pode ser determinada antes da programação dos trabalhos ou conjuntamente com a programação dos trabalhos.

No primeiro caso, os períodos de manutenção são pré-determinados e fixos e as máquinas ficam indisponíveis para o processamento de trabalhos durante os respectivos períodos de manutenção. Este problema é normalmente referido na literatura como programação com restrições de disponibilidade das máquinas (*scheduling with machine availability constraints*). O segundo caso, trata a programação conjunta de actividades de manutenção e processamento de trabalhos de forma a optimizar uma dada medida de desempenho.

Lee e Chen (2000) estudaram uma outra extensão do problema $P||\sum w_j C_j$ envolvendo actividades de manutenção preventiva. Cada máquina é sujeita a uma actividade de manutenção de duração t no período de tempo $[0,T]$, em que T é um limite superior para a conclusão das actividades de manutenção, tal que $T \geq t$; os trabalhos podem ser concluídos após T. Foram estudadas duas variantes deste problema dependendo da capacidade de manutenção que pode ser de uma máquina ou infinita.

Se a capacidade de manutenção é infinita, qualquer número de máquinas pode ser mantido em simultâneo e, como tal, as sequências de afectação das máquinas são independentes relativamente às actividades de manutenção. A notação para este problema é $P|ind|\sum w_j C_j$. O problema é formulado como um modelo de partição de conjuntos em que as variáveis de decisão representam sequências de afectação parciais de um subconjunto de trabalhos de T e uma actividade de manutenção numa máquina. Pretende-se encontrar exactamente m sequências de afectação parciais que processam uma única vez cada trabalho. As m sequências de afectação parciais garantem que cada máquina é sujeita a uma actividade de manutenção.

Se a capacidade de manutenção é de uma máquina, não pode haver sobreposição de actividades de manutenção entre diferentes máquinas, isto é, as sequências de afectação das máquinas são dependentes relativamente às actividades de manutenção. A notação para este segundo problema é $P|dep|\sum w_j C_j$ e, neste caso, é assumido que $T \geq mt$ para que possam existir sequências de afectação admissíveis.

A formulação adoptada é semelhante à do primeiro problema exceptuando que, agora, existe uma distinção entre as máquinas já que estas são mutuamente dependentes no que diz respeito às actividades de manutenção. É ainda introduzida uma restrição adicional que, baseando-se na assunção de que as máquinas são mantidas pela ordem do seu índice, impõe que a diferença entre os tempos de conclusão das actividades de manutenção em duas máquinas consecutivas tem de ser superior ou igual à sua duração t.

A formulação do subproblema, que resolve um problema de programação de trabalhos e uma actividade de manutenção em uma máquina, é idêntica à que foi descrita na Subsecção anterior permitindo a inclusão do mesmo trabalho mais do que uma vez na mesma sequência de afectação. Os trabalhos são ordenados por ordem não crescente do rácio w_j/p_j e a actividade de manutenção é considerada como sendo o trabalho n+1. Depois são criados mais n trabalhos, n+2,…,2n+1, tal que o trabalho n+j+1 é igual ao trabalho j; o primeiro conjunto de trabalhos só pode ser processado antes da actividade de manutenção e o segundo conjunto de trabalhos só pode ser processado depois da actividade de manutenção.

Nesta nova formulação, o subproblema resolve o problema de encontrar, para uma máquina, um subconjunto ordenado do primeiro conjunto de trabalhos e um subconjunto ordenado do segundo conjunto de trabalhos, de menor custo reduzido, tal que, a actividade de manutenção é concluída não mais tarde do que T e são verificadas as seguintes condições de optimalidade:

- **(Lee e Chen, 2000)** Num programa óptimo, as sequências de afectação são constituídas por dois subconjuntos de trabalhos separados por uma actividade de manutenção e em cada um dos subconjuntos, os trabalhos estão ordenados por

ordem não crescente do rácio w_j/p_j, e não existem tempos de espera das máquinas e as actividades de manutenção iniciam em t=0 ou no tempo de conclusão de um qualquer trabalho.

- **(Lee e Chen, 2000)** Para o problema $P|ind|\sum w_jC_j$, existe um programa óptimo tal que não existem tempos de espera das máquinas e as actividades de manutenção iniciam em t=0 ou no tempo de conclusão de um qualquer trabalho.

- **(Lee e Chen, 2000)** Para o problema $P|ind|\sum w_jC_j$, existe um programa óptimo tal que, em cada máquina, o último trabalho a ser processado antes da actividade de manutenção é concluído nunca depois de $R_1 = \left[P + (m-1)t - p_{\max}\right]/m + p_{\max}$, e o último trabalho a ser processado depois da actividade de manutenção é concluído nunca depois de $R_2 = \left[P - p_{\max}\right]/m + t + p_{\max}$, tal que, $P = \sum_{j\in N} p_j$ e $p_{\max} = \max_{j\in N}\{p_j\}$.

- **(Lee e Chen, 2000)** Para o problema $P|dep|\sum w_jC_j$, existe um programa óptimo tal que, em cada máquina, o último trabalho a ser processado antes da actividade de manutenção é concluído nunca depois de $R_3 = \left[P + (m-1)t + (m-2)p_{\max}\right]/m + p_{\max}$, e o último trabalho a ser processado depois da actividade de manutenção é concluído nunca depois de $R_4 = \left[P + (m-1)p_{\max}\right]/m + t + p_{\max}$.

A estratégia de partição é semelhante à usada por Chen e Powell (1999a) e descrita na Subsecção 4.3.3.1. Contudo, para o problema $P|dep|\sum w_jC_j$, ao ser aplicada a partição no sucessor imediato, pode acontecer que não seja possível encontrar uma solução óptima inteira, apesar dos autores referirem que esta situação nunca aconteceu nos testes computacionais realizados.

Recorde-se que, como podem existir tempos de espera da máquina por uma actividade de manutenção, os mesmos trabalhos podem estar presentes em duas colunas diferentes, mas em que as actividades de manutenção têm diferentes tempos de conclusão. Neste

caso, a partição é efectuada sobre os tempos de conclusão dos trabalhos. Os resultados computacionais mostram que o algoritmo é capaz de resolver problemas até 40 trabalhos e duas ou mais máquinas.

Capítulo 5

Formulação do Problema

Neste Capítulo é apresentada a definição formal do problema de programação máquinas paralelas não-idênticas em estudo, proposta uma formulação para a sua representação matemática, e proposto um algoritmo de partição e geração de colunas para a sua resolução exacta.

5.1. Introdução

Depois da definição formal do problema de programação máquinas paralelas não-idênticas em estudo na Secção 5.2, é apresentada uma formulação compacta de programação inteira na Secção 5.3, e é descrita a aplicação do método de decomposição de Dantzig-Wolfe ao problema em estudo na Secção 5.4. Na Secção 5.5 são descritos três modelos de programação dinâmica para a resolução do subproblema, é proposto um método para aumentar a eficiência do modelo escolhido e é apresentado um algoritmo para a sua implementação. Por último, é apresentado na Secção 5.6 o algoritmo de partição e avaliação.

5.2. Definição do Problema

Neste trabalho, consideramos uma classe de problemas de programação de máquinas paralelas em que pretendemos afectar e sequenciar um conjunto de *n trabalhos*

independentes, $N=\{1,2,\ldots,n\}$, em m máquinas *não-idênticas*, $M=\{1,2,\ldots,m\}$, com o objectivo de minimizar a soma ponderada dos atrasos (*total weighted tardiness*).

Cada máquina $k \in M$ tem uma data de disponibilidade a_k (a máquina não está disponível antes desta data) e uma configuração inicial l_k que representa o último trabalho processado antes da máquina ficar disponível.

Cada trabalho $j \in N$ deve ser processado por exactamente uma das máquinas e tem m tempos de processamento p_{jk} ($k \in M$), uma prioridade w_j, uma data de disponibilidade r_j, e uma data de entrega d_j. O processamento dos trabalhos nas máquinas implica um tempo de preparação dependente da sequência de trabalhos a processar s_{ij}, sempre que $i \neq j$, sendo que o tempo de preparação inicial é dado por $s_{l_k j}$ se o trabalho j for o primeiro a ser processado na máquina k. Se um trabalho for concluído após a sua data de entrega d_j incorre numa penalidade w_j por cada unidade de tempo de atraso.

Uma máquina (disponível) pode aguardar pela disponibilidade de um trabalho, permanecendo inactiva (*idle*) durante o período de espera. No entanto, a preparação da máquina pode ser levada a cabo durante o período de espera.

Assume-se que todos os parâmetros são determinísticos e tomam valores inteiros não negativos. Também, como é normalmente assumido na literatura (Potts and Kovalyov, 2000), os tempos de preparação são definidos de forma a satisfazer o princípio das desigualdades triangulares, ist é, $s_{ij}+s_{jv} \geq s_{iv}$, para todos os $i,j,v \in N$, com a excepção dos tempos de preparação iniciais dados por $s_{l_k j}$.

A notação matemática para este problema é a seguinte:

$$R|a_k,r_j,s_{ij}|\sum w_j T_j \tag{5.1}$$

Este problema é fortemente NP-difícil porque é uma generalização do problema $P||\sum w_j T_j$ que é conhecido como sendo fortemente NP-difícil, mesmo quando $m=1$ (Lenstra, Kan e Brucker, 1977). É também um caso geral dos problemas de programação de máquinas paralelas já que engloba os casos em que não existem datas

de disponibilidade para as máquinas (a_k=0, $\forall k \in M$), datas de disponibilidade para os trabalhos (r_j=0, $\forall j \in N$), tempos de preparação dependentes da sequência (s_{ij}=0, $\forall i,j \in N$), ou em que as máquinas são iguais ($p_{jk}=p_j$, $\forall j \in N$, $\forall k \in M$) ou uniformes ($p_{jk}=p_j/b_k$, $\forall j \in N$, $\forall k \in M$ e onde p_j é o tempo de processamento standard e b_k é o factor de velocidade de processamento da máquina k). Engloba também o caso particular da programação de múltiplas famílias (MJF) em que o processamento consecutivo de trabalhos de diferentes famílias requer um tempo de preparação.

Uma *sequência de afectação* (*machine schedule*) é a afectação e o sequenciamento de um subconjunto de trabalhos de N a uma máquina de M. Uma *sequência de afectação admissível* (*feasible machine schedule*) é uma sequência de afectação em que os trabalhos são processados uma única vez e que satisfaz as restrições de disponibilidade da máquina e dos trabalhos.

Pretendemos encontrar o conjunto ($\leq m$) de sequências de afectação admissíveis que processam todos os trabalhos $j \in N$ e minimizam a soma ponderada dos atrasos.

Exemplo 5.1 – Preparação das máquinas e afectação de trabalhos

No diagrama de Gantt apresentado na Figura 5.1 podemos ver um exemplo ilustrativo de uma possível sequência de afectação de três trabalhos a duas máquinas. A máquina 1 vai processar os trabalhos 2 e 3 e a máquina 2 vai processar o trabalho 1.

A máquina 1 fica disponível na data a_1 e apresenta uma configuração inicial l_1 o que vai originar um tempo de preparação para processar o primeiro trabalho, j=2, igual a $s_{l_1 2}$. Entretanto, quando termina esta preparação inicial, o trabalho 2 já está disponível ($r_2<a_1+s_{l_1 2}$) e fica a aguardar pela disponibilidade da máquina. Segue-se o seu processamento durante o período de tempo p_{21}, para estar concluído no instante de tempo $t=C_2$. Segue-se o tempo de preparação da máquina 1 para processar a trabalho 3 depois de ter processado o trabalho 2, s_{23}. Como o trabalho 3 só vai ficar disponível no instante de tempo $r_3>(C_2+s_{23})$, a máquina 1 vai ter de esperar pelo trabalho 2

permanecendo inactiva durante um período de tempo r_3-(C_2+s_{23}). Finalmente, é processado o trabalho 3, que fica concluído no instante de tempo t=C_3.

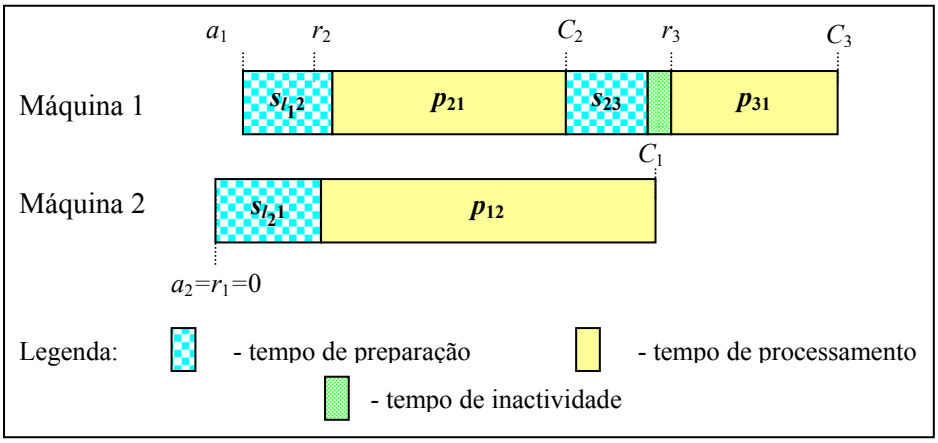


Figura 5.1 – Preparação das máquinas e afectação de trabalhos

A máquina 2 fica disponível na data a_1=0 e apresenta uma configuração inicial l_2 o que vai originar um tempo de preparação para processar o primeiro trabalho, j=1, igual a $s_{l_1 2}$. Como o trabalho 1 fica disponível na data r_1<a_2+ $s_{l_1 2}$, tem que esperar pela conclusão da preparação da máquina seguindo-se o seu processamento durante o período de tempo p_{12}, para estar concluído no instante de tempo t=C_1.□

Do conhecimento que temos da literatura sobre a aplicação de métodos de solução exacta a problemas de programação de máquinas paralelas, trata-se do problema mais geral (e mais próximo daqueles que podemos encontrar em ambientes industriais) pois considera datas de disponibilidade para as máquinas e para os trabalhos, datas de entrega para os trabalhos e tempos de preparação das máquinas dependentes da sequência de trabalhos a processar, e a sua solução exacta nunca foi tentada antes.

5.3. Formulação de Programação Inteira

Vamos definir as seguintes variáveis binárias (0,1) para $i,j \in N$ e $k \in M$.

- x_{ij}^k=1 se o trabalho *j* é processado imediatamente após o trabalho *i* na máquina *k*; x_{ij}^k=0 no caso contrário.

- x_{0j}^k=1 se o trabalho *j* é o primeiro a ser processado na máquina *k*; x_{0j}^k=0 no caso contrário.

- $x_{j,n+1}^k$=1 se o trabalho *j* é o último a ser processado na máquina *k*; $x_{j,n+1}^k$=0 no caso contrário.

A formulação de programação inteira (PI-I) para o problema em estudo é a seguinte:

PI - I :

$$\min \sum_{j\in N} \max(C_j - d_j, 0) w_j \tag{5.2}$$

Sujeito a :

$$\sum_{k\in M} \sum_{i\in N\cup\{0\}|i\neq j} x_{ij}^k = 1, \forall j \in N \tag{5.3}$$

$$\sum_{j\in N} x_{0j}^k \leq 1, \forall k \in M \tag{5.4}$$

$$\sum_{i\in N\cup\{0\}|i\neq j} x_{ij}^k = \sum_{i\in N\cup\{n+1\}|i\neq j} x_{ji}^k, \forall j \in N, \forall k \in M \tag{5.5}$$

$$x_{ij}^k \in \{0,1\}, \forall i \in N\cup\{0\}, \forall j \in N\cup\{n+1\}, \forall k \in M \tag{5.6}$$

sendo o tempo de conclusão

$$C_j = \sum_{k\in M} \sum_{i\in N\cup\{0\}|i\neq j} \left[\max(C_i + s_{ij}, r_j) + p_{jk}\right] x_{ij}^k, \forall j \in N \tag{5.7}$$

$$\text{com } C_0 = a_k \text{ e } s_{0j} = s_{l_k j}, \forall l_k \in L$$

A função objectivo (5.2) minimiza a soma ponderada dos atrasos. O conjunto de restrições (5.3) garante que cada trabalho é processado exactamente uma única vez e o conjunto de restrições (5.4) garante que cada máquina é utilizada uma vez, no máximo. As restrições (5.5) são restrições de conservação do fluxo que garantem um sequenciamento correcto dos trabalhos que conduza a sequências de afectação admissíveis. As restrições (5.6) impõem a integralidade binária das variáveis de decisão, e a equação (5.7) define o tempo de conclusão dos trabalhos.

Este é um problema de programação inteira de grande dimensão cuja resolução é difícil já que envolve uma expressão não linear (5.7). Por este motivo se recorre a métodos de decomposição, e à resolução de um problema reformulado. Assim, esta formulação também é conhecida na literatura por formulação original.

Podemos também observar que este problema de programação matemática é composto por subsistemas semelhantes, representados pelas *m* máquinas, onde as matrizes dos coeficientes tecnológicos apresentam uma estrutura angular em blocos, isto é, um ou mais blocos independentes ligados por equações de ligação, onde se podem identificar submatrizes de grande dimensão constituídas somente por zeros. Isto acontece porque muitas das actividades associadas às variáveis de decisão apenas participam directamente num pequeno número de restrições.

Exemplo 5.2 – Estrutura angular em blocos

A tabela da Figura 5.2 representa a matriz de coeficientes tecnológicos que resulta da aplicação do modelo PI-I a um problema com dois trabalhos e duas máquinas onde é claramente visível uma estrutura angular em blocos (definidos pelas máquinas) e onde também se pode verificar a existência de submatrizes de grande dimensão constituídas somente por zeros.

	x^1_{01}	x^1_{21}	x^1_{02}	x^1_{12}	$x^1_{1,n+1}$	$x^1_{2,n+1}$	x^2_{01}	x^2_{21}	x^2_{02}	x^2_{12}	$x^2_{1,n+1}$	$x^2_{2,n+1}$	
T_1	1	1	0	0	0	0	1	1	0	0	0	0	=1
T_2	0	0	1	1	0	0	0	0	1	1	0	0	=1
M_1	1	0	1	0	0	0	0	0	0	0	0	0	≤1
M_2	0	0	0	0	0	0	1	0	1	0	0	0	≤1
T_1M_1	1	1	0	-1	-1	0	0	0	0	0	0	0	=0
T_2M_1	0	-1	1	1	0	-1	0	0	0	0	0	0	=0
T_1M_2	0	0	0	0	0	0	1	1	0	-1	-1	0	=0
T_2M_2	0	0	0	0	0	0	0	-1	1	1	0	-1	=0

Figura 5.2 – Estrutura angular em blocos

Por exemplo, a coluna referente à variável de decisão x_{21}^{1}, que representa o processamento do trabalho 1 logo após o processamento do trabalho 2 na máquina 1, apenas apresenta coeficientes não nulos nas linhas que representam o trabalho 1 (T_1) e a máquina 1 (M_1), respectivamente, e como o fluxo sai da tarefa 2 para a tarefa 1, tem o coeficiente -1 na linha que representa a tarefa 1 e a máquina 1 (T_1M_1), e o coeficiente 1 na linha que representa a tarefa 2 e a máquina 1 (T_2M_1), respectivamente.□

Podemos explorar esta estrutura especial e aplicar o método de decomposição de Dantzig-Wolfe, decompondo o problema PI-I em *m subproblemas*, com espaços de soluções definidos pelas restrições (5.5) e (5.6) e a função de custo (5.7), e num *problema principal* (PP) que consiste na função objectivo (5.2) e nas restrições (5.3) e (5.4).

5.4. Decomposição de Dantzig-Wolfe

5.4.1. Representação em Rede

Seja $G=(N,A)$ uma rede onde A representa o conjunto de arcos e N representa o conjunto de nodos. O conjunto de nodos contém $n+2$ nodos, $N=\{0,1,\ldots,n,n+1\}$. O nodo "0" representa o nodo fonte (*source node*) e o nodo $n+1$ representa o nodo terminal (*sink node*). No problema em estudo, estes nodos representam o início e o fim de uma sequência de afectação, respectivamente. Os restantes nodos representam os trabalhos.

O conjunto de arcos A consiste nos arcos (i,j), para todos $i\neq j$, $i \in N\cup\{0\}$, $j \in N\cup\{n+1\}$. Um caminho p na rede G pode ser definido como uma sequência de nodos $(j_0,j_1,\ldots, j_H, j_{H+1})$, tal que cada um dos arcos (j_h,j_{h+1}) pertence a A, para $0\leq h\leq H$ e $H\leq n$. Todos os caminhos se iniciam no nodo $j_0=0$ e terminam no nodo $j_{H+1}=n+1$. A cada arco $(i,j) \in A$ estão associados um custo c_{ij} e uma duração t_{ij} (que é função do tempo de processamento p_{jk}, do tempo de preparação s_{ij} e da data de disponibilidade do trabalho r_j). Por definição, o custo e a duração associados aos arcos terminais $(i,n+1)$ são nulos.

Pode-se facilmente verificar que um caminho acíclico (simples) na rede G corresponde a uma sequência de afectação admissível para uma dada máquina k, sendo que as variáveis x_{ij}^k representam o valor do fluxo nos arcos (i,j). Podemos então definir uma sequência de afectação acíclica ou admissível, como o caminho que se inicia no nodo 0, processa uma única vez um subconjunto de trabalhos de N, e termina no nodo n+1, satisfazendo todas as restrições definidas.

Uma sequência de afectação que contenha ciclos, isto é, que repita o processamento de pelo menos um dos trabalhos, designa-se por *sequência de afectação cíclica* ou *não-admissível.* Seja também p_0 a *sequência de afectação nula* (*caminho nulo*), um caminho que se inicia no nodo 0 e termina no nodo n+1, não processando qualquer trabalho de N.

Seja também P^k o conjunto de sequências de afectação (acíclicas ou cíclicas) p para a máquina k.

Exemplo 5.3 – Representação do problema em rede

Na Figura 5.3, está representada a rede G de qualquer máquina $k \in M$, para um problema com dois trabalhos.

As sequências de afectação iniciam-se no nodo "0" e terminam no nodo n+1, equivalendo a caminhos na rede G. O espaço de soluções deste problema, P^k, contém 5 sequências de afectação e está representado na Figura 5.3.

O caminho (0,1,2,n+1) que se inicia no nodo 0, processa uma única vez os trabalhos 1 e 2 e termina no nodo n+1, é um exemplo de uma *sequência de afectação acíclica* ou *admissível*. Aos arcos (0,1) e (1,2) estão associados os custos c_{01} e c_{12} e as durações t_{01} e $t_{12,}$ respectivamente. O custo e a duração associados ao arco terminal (2,n+1) são nulos.

A *sequência de afectação nula* (*caminho nulo*) é representada pelo caminho (0,n+1) que se inicia no nodo 0 termina no nodo n+1, não processando qualquer trabalho de N.

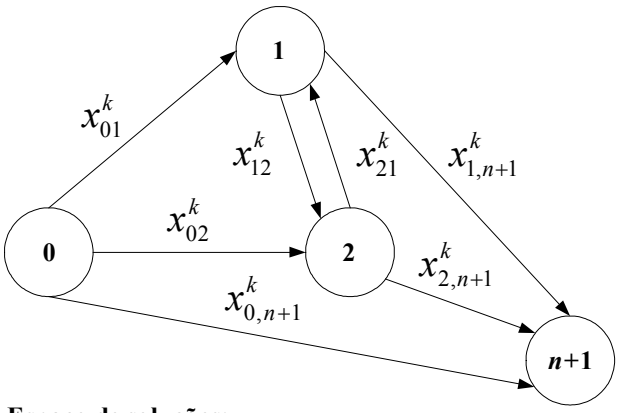


Figura 5.3 – Representação em rede

Estão ainda representadas na Figura 5.3, as variáveis de valor do fluxo nos arcos x_{ij}^k. Para a sequência de afectação (0,2,1,n+1), as variáveis x_{02}^k, x_{21}^k e $x_{1,n+1}^k$ tomam o valor 1 e as restantes, x_{01}^k, x_{12}^k, $x_{2,n+1}^k$ e $x_{0,n+1}^k$, tomam o valor zero.□

5.4.2. Reformulação do Problema

Seja X^k o conjunto de valores de fluxo nos arcos, x_{ij}^k, para a máquina k. Sabemos, da teoria das redes, que um caminho numa rede corresponde a um ponto extremo, $X_p^k = (X_{ijp}^k)$, do poliedro definido pela envolvente convexa do conjunto de pontos X^k - $conv(X^k)$. Consequentemente, podemos exprimir qualquer fluxo nos arcos, x_{ij}^k, como uma combinação convexa de fluxos nos caminhos:

$$x_{ij}^k = \sum_{p \in P^k} X_{ijp}^k y_p^k, \forall (i,j) \in A, \forall k \in M \tag{5.8}$$

$$\sum_{p \in P^k} y_p^k = 1, \forall k \in M \tag{5.9}$$

$$y_p^k \geq 0, \forall p \in P^k, \forall k \in M \tag{5.10}$$

a expressão (5.9) é referida como sendo a restrição de convexidade.

Note-se que, se a máquina k não for utilizada, a variável $y_{p_0}^k$, que corresponde ao caminho nulo p_0, toma o valor 1, representando o ponto extremo do espaço de soluções da máquina k relativo à origem das coordenadas, o que assegura a validade de (5.9). No nosso caso, vamos deixar a sequência de afectação nula fora da formulação já que o seu custo é nulo, passando a escrever a restrição de convexidade como uma desigualdade (≤).

Exemplo 5.4 – Soluções PI-I e caminhos na rede

A rede seguinte representa um problema de programação de dois trabalhos (j=2) numa qualquer máquina k, onde o caminho nulo (0,n+1) não está representado.

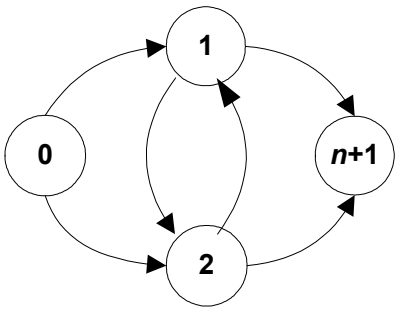


Figura 5.4 – Rede do exemplo

Este problema tem quatro sequências de afectação correspondentes aos caminhos (0,1,3), (0,2,3), (0,1,2,3), (0,2,1,3), representadas no espaço de soluções $conv(X^k)$ pelos respectivos pontos extremos x_1^k, x_2^k, x_3^k e x_4^k.

Ao aplicarmos a formulação PI-I a este mesmo exemplo, obtemos o seguinte conjunto de soluções admissíveis,

$$\left(x_{01}^k = 1, x_{02}^k = 0, x_{12}^k = 0, x_{13}^k = 1, x_{21}^k = 0, x_{23}^k = 0\right),$$

$$\left(x_{01}^k = 0, x_{02}^k = 1, x_{12}^k = 0, x_{13}^k = 0, x_{21}^k = 0, x_{23}^k = 1\right),$$

$$\left(x_{01}^k = 1, x_{02}^k = 0, x_{12}^k = 1, x_{13}^k = 0, x_{21}^k = 0, x_{23}^k = 1\right),$$

$$\left(x_{01}^k = 0, x_{02}^k = 1, x_{12}^k = 0, x_{13}^k = 1, x_{21}^k = 1, x_{23}^k = 0\right).$$

É fácil de verificar que a cada uma das soluções admissíveis do problema PI-I corresponde um caminho *p* na rede, representado pelos pontos extremos x_p^k.□

Substituindo agora as expressões (5.8) a (5.10) na formulação PI-I, obtemos a seguinte formulação extensa (PI-II),

PI - II :

$$\min \sum_{j\in N} \max(C_j - d_j, 0) w_j, \tag{5.11}$$

sujeito a :

$$\sum_{k\in M} \sum_{p\in P^k} \sum_{i\in N\cup\{0\}|i\neq j} x_{ijp}^k y_p^k = 1, \forall j \in N \tag{5.12}$$

$$\sum_{p\in P^k} \sum_{j\in N} x_{0jp}^k y_p^k \leq 1, \forall k \in M \tag{5.13}$$

$$\sum_{p\in P^k} y_p^k \leq 1, \forall k \in M \tag{5.14}$$

sendo

$$C_j = \sum_{k\in M} \sum_{p\in P^k} \sum_{i\in N\cup\{0\}|i\neq j} \left[\max\left(C_i + s_{ij}, r_j\right) + p_{jk}\right] x_{ijp}^k y_p^k, \forall j \in N \tag{5.15}$$

$$\text{com } C_0 = a_k \text{ e } s_{0j} = s_{l_k j}, \forall l \in L$$

$$x_{ij}^k = \sum_{p\in P^k} x_{ijp}^k y_p^k, \forall (i,j) \in A, \forall k \in M \tag{5.16}$$

$$x_{ij}^k \in \{0,1\}, \forall i \in N \cup \{0\}, \forall j \in N \cup \{n+1\}, \forall k \in M \tag{5.17}$$

Nesta formulação (PI-II), as restrições de conservação do fluxo (5.5) estão implicitamente consideradas nas sequências de afectação representadas pelos pontos extremos $x_p^k = \left(x_{ijp}^k\right)$. A equação (5.16) preserva a ligação entre as variáveis de decisão o que nos vai permitir ter sempre rastreabilidade ao problema original. Como as variáveis x_{ij}^k só podem tomar valores binários, então, todos os pontos inteiros do conjunto fechado X^k são também pontos extremos do poliedro convexo definido por $conv(X^k)$

porque os valores das variáveis x_{ij}^k são obtidos através de uma combinação convexa trivial de apenas um ponto extremo do espaço de soluções definido por $conv(X^k)$, e, consequentemente, os valores que as variáveis y_p^k tomam também são binários, tomando o valor 1 se a sequência de afectação p é usada e 0 no caso contrário. O inverso também se verifica, isto é, $x_{ij}^k \in \{0,1\} \Leftrightarrow y_p^k \in \{0,1\}$.

Verifica-se facilmente que a expressão $\sum_{i \in N \cup \{0\} | i \neq j} x_{ijp}^k$ representa o número de vezes que a sequência de afectação $p \in P^k$ processa o trabalho $j \in N$ e que a expressão (5.13) pode ser simplificada já que a utilização da máquina é representada pela variável y_p^k que toma o valor 1 se a sequência de afectação p da máquina k é usada e 0 no caso contrário.

Se $C_{j_h}^k$ for o tempo de conclusão do trabalho j_h na sequência de afectação (j_0,j_1,..., j_H, j_{H+1}), na máquina k, o tempo de conclusão do próximo trabalho na sequência, $C_{j_{h+1}}^k$, pode ser obtido pela seguinte expressão:

$$C_{j_{h+1}}^k = \max\left(r_{j_{h+1}}, C_{j_h}^k + s_{j_h, j_{h+1}}\right) + p_{j_{h+1},k} \quad (5.18)$$
$$\text{com} \quad C_{j_0}^k = a_k \text{ e } C_{j_{H+1}}^k = C_{j_H,p}^k, \ \forall k \in M, 1 \leq h \leq H$$

O custo de um arco (j_h,j_{h+1}) é função do tempo de conclusão do trabalho j_{h+1} na sequência de afectação. Então, podemos definir o custo de um arco (j_h,j_{h+1}) da sequência de afectação p na máquina k, como o custo de completar o trabalho j_{h+1} imediatamente após ter completado o processamento do trabalho j_h (no tempo $C_{j_h}^k$),

$$c_{j_h, j_{h+1}}^k = w_j \left[\max\left(\max\left(r_{j_{h+1}}, C_{j_h}^k + s_{j_h, j_{h+1}}\right) + p_{j_{h+1},k} - d_{j_{h+1}}, 0\right)\right] \quad (5.19)$$

O custo de uma sequência de afectação p, c_p^k, é igual à soma dos custos dos arcos pertencentes a p, e pode ser obtido pela seguinte expressão:

$$c_p^k = \sum_{h=0}^{H} c_{j_h, j_{h+1}}^k \quad (5.20)$$

Desta forma, podemos escrever uma formulação equivalente (ISP) que é conhecida por problema principal (PP) da decomposição de Dantzig-Wolfe,

$$\text{ISP}:$$

$$\min \sum_{k\in M}\sum_{p\in P^k} c_p^k y_p^k, \tag{5.21}$$

$$\text{sujeito a}:$$

$$\sum_{k\in M}\sum_{p\in P^k} a_{jp}^k y_p^k = 1, \forall j \in N \tag{5.22}$$

$$\sum_{p\in P^k} y_p^k \le 1, \forall k \in M \tag{5.23}$$

$$y_p^k \in \{0,1\}, \forall p \in P^k, \forall k \in M \tag{5.24}$$

$$\text{onde}$$

$$a_{jp}^k = \sum_{i\in N\cup\{0\}|i\neq j} x_{ijp}^k, \forall j \in N, \forall p \in P^k, \forall k \in M \tag{5.25}$$

$$x_{ij}^k = \sum_{p\in P^k} x_{ijp}^k y_p^k, \forall (i,j) \in A, \forall k \in M \tag{5.26}$$

O primeiro conjunto de restrições (5.22) garante que cada trabalho é processado uma única vez; o segundo conjunto de restrições (5.23) garante que cada máquina é utilizada no máximo uma vez. A variável a_{jp}^k representa o número de vezes que o trabalho j é incluído na sequência de afectação $p \in P^k$. A equação (5.26) é mantida nesta formulação para explicitar a necessidade de preservar a ligação às variáveis de decisão do problema original.

Note-se que a função objectivo é agora linear e que quando relaxamos a restrição de integralidade de y_p^k (5.24) também podemos relaxar a restrição de ligação (5.26) entre y_p^k e x_{ij}^k.

Nesta formulação, baseada no modelo de partição de conjuntos (ISP-modelo de partição de conjuntos com restrições adicionais), pretendemos, a partir do conjunto N dos trabalhos, encontrar $S \le m$ subconjuntos de N, P_s (representados pelas sequências de afectação $p \in P^k$), disjuntos que minimizem uma função de custo, tal que $\bigcup_{s=1}^{S} P_s = N$ e com a restrição adicional de pertencerem todos a máquinas diferentes. Os subconjuntos

de trabalhos e o seu custo são gerados nos *m* subproblemas que resolvem problemas de programação de uma máquina.

As principais razões para esta reformulação são de facto duas; a primeira razão é o facto da expressão de cálculo do custo (5.7) ser não linear; a segunda razão tem a ver com o facto de que a relaxação linear do PP ser mais forte do que a relaxação linear da formulação original (PI-I) já que as soluções fraccionárias de PI-I que não são combinações convexas dos pontos extremos do espaço de soluções do PP são não admissíveis para PP.

Relativamente a PI-I, esta formulação apresenta um maior número de variáveis de decisão e um menor número de restrições. Se bem que este problema de programação inteira binária possa ser resolvido através da sua relaxação linear, o número de variáveis de decisão y_p^k, correspondentes a sequências de afectação, cresce exponencialmente com o número de trabalhos e o número de máquinas, tornando a resolução deste problema de forma directa (por procedimentos de enumeração completa) só possível para problemas de pequena dimensão.

Exemplo 5.5 – Formulação completa baseada no modelo de partição de conjuntos

Para um (muito) pequeno problema com duas máquinas (*M*1 e *M*2) e dois trabalhos (*T*1 e *T*2) é apresentada na tabela seguinte o conjunto de colunas referente a uma enumeração completa.

	y_1^1	y_2^1	y_3^1	y_4^1	y_1^2	y_2^2	y_3^2	y_4^2	
T_1	1		1	1	1		1	1	=1
T_2		1	1	1		1	1	1	=1
M_1	1	1	1	1					<=1
M_2					1	1	1	1	<=1
	c_1^1	c_2^1	c_3^1	c_4^1	c_1^2	c_2^2	c_3^2	c_4^2	

Figura 5.5 – Formulação completa

O espaço de soluções para cada uma das máquinas tem quatro sequências de afectação correspondentes aos caminhos (0,1,3), (0,2,3), (0,1,2,3), (0,2,1,3).

Exemplificando o preenchimento da tabela, temos que a sequência de afectação 4 na máquina 1, representada pela variável de decisão y_4^1, corresponde ao caminho (0,2,1,3) que processa os trabalhos $T2$ e $T1$ na máquina $M1$ e tem um custo c_4^1.

Relativamente ao Exemplo 5.2, onde está representada uma formulação PI-I para este mesmo problema, verifica-se que esta formulação apresenta um menor número de restrições

Nota: também se verifica que, para este exemplo, esta formulação apresenta um menor número de variáveis de decisão mas isso só acontece porque se trata de um problema de muito pequena dimensão.□

Alternativamente à enumeração completa, a relaxação linear deste problema vai ser resolvida através de um procedimento de enumeração implícita, o algoritmo de geração de colunas. Começamos com apenas um subconjunto restrito das variáveis de decisão (colunas), formando o problema primal restrito, sendo que as restantes são tratadas de uma forma implícita. O problema primal restrito "coordena" a geração de sequências de afectação p em cada um dos m subproblemas, que resolvem um problema de determinação do caminho mais curto numa rede G com custos modificados.

5.4.3. Transferência de Informação Dual para os Subproblemas

Da teoria da programação linear, sabemos que a solução de um problema de minimização é óptima se o custo reduzido de cada uma das variáveis é não negativo. O algoritmo simplex fornece o conjunto de valores das variáveis duais π_j e υ_k, para todos os $j \in N$ e $k \in M$, associados com o primeiro e segundo conjunto de restrições do problema de programação linear, respectivamente, necessários para a resolução dos subproblemas.

O custo reduzido do arco (j_h, j_{h+1}) de uma sequência de afectação p, na máquina k, $\overline{c}^k_{j_h, j_{h+1}}$, pode ser obtido pela seguinte expressão:

$$\overline{c}^k_{j_h, j_{h+1}} = c^k_{j_h, j_{h+1}} - \pi_{j_{h+1}} \tag{5.27}$$

e o custo reduzido de uma sequência de afectação p, $\overline{c}^k_p$, é,

$$\overline{c}^k_p = -\upsilon_k + \sum_{h=0}^{H} \overline{c}^k_{j_h, j_{h+1}} \tag{5.28}$$

Na próxima secção apresentamos a formulação dos subproblemas.

5.5. Formulação dos Subproblemas

Os subproblemas devem ser capazes de representar todas as sequências de afectação para cada uma das máquinas e de as avaliar da forma mais eficiente possível. Nesta secção, vamos apresentar três modelos de programação dinâmica para determinar a solução óptima dos subproblemas.

O primeiro modelo (Modelo 1) determina todas as sequências de afectação admissíveis mas não são conhecidos algoritmos eficientes para a sua resolução. O segundo modelo (Modelo 2), tem um espaço de soluções que contém, para além das sequências de afectação admissíveis, sequências de afectação não-admissíveis, mas apresenta uma complexidade pseudo-polinomial para o caso mais difícil (*worst case*). O terceiro modelo (Modelo 3), é uma versão do Modelo 2 ao qual foi adicionado um esquema de eliminação dos ciclos de ordem 2. Finalmente, propomos um método que tira partido da estrutura especial do subproblema para aumentar muito significativamente a eficiência do Modelo 3.

5.5.1. Modelo 1

Seja $F_j^k(S,t)$, o mínimo custo reduzido da sequência de afectação que vai do nodo máquina "0" até ao nodo do trabalho *j* processando uma única vez todos os trabalhos do conjunto *S*, tendo completado o processamento do trabalho *j* no instante de tempo *t*, na máquina *k*. $F_j^k(S,t)$ pode ser calculado através das seguintes equações de recorrência:

$$F_0^k(\{\},a_k) = -\upsilon_k;$$
$$F_j^k(S,t) = \min_{(i,j)\in A}\{F_i^k(S-\{j\},t') + \bar{c}_{ij}^k \mid t' \le t - s_{ij} - p_{jk}\},$$
$$t = \min_{(i,j)\in A}\{\max(t'+s_{ij}+p_{jk}, r_j + p_{jk}) \mid t' \in \{F_i^k(S-\{j\},t')\}\}$$
$$\text{para todos os } j,S,k \text{ tal que } j \in N, S \subseteq N, k \in M$$

Um caso especial deste problema é o problema de programação de uma única máquina $1||\sum w_j T_j$ que, como já foi referido, é considerado um problema fortemente NP-difícil e não são conhecidos algoritmos pseudo-polinomiais para a sua resolução.

5.5.2. Modelo 2

Christofides, Mingozzi e Toth (1981) propuseram uma relaxação do espaço de estados para problemas de programação dinâmica de forma a obter limites inferiores da função objectivo, fáceis de calcular. Podemos aplicar esta técnica definindo um mapeamento do espaço de estados original (*S,t*) em um novo espaço de estados (*j,t*) e definindo novas transições. Cada estado (*S,t*) é mapeado no estado (*j,t*), e $G^k(j,t)$ é o mínimo custo reduzido da sequência de afectação que vai desde o nodo "0" até ao nodo *j*, tendo completado o processamento do trabalho *j* no instante de tempo *t*, na máquina *k*. Seja também T^k um limite superior para a duração total de uma sequência de afectação na máquina *k*.

As novas equações de recorrência são:

$$G^k(0,a_k) = -\upsilon_k;$$
$$G^k(j,t) = \min_{(i,j)\in A}\{G^k(i,t') + \bar{c}_{ij}^k \mid t' \le t - p_{jk} - s_{ij}\},$$

para todos os j,k,t tal que $j \in N, k \in M$ e $\left(r_j + p_{jk}\right) \le t \le T^k$

Pelo facto de a grandeza física que identifica um estado estar associada ao tempo já decorrido e não conter nenhuma informação sobre os estados já visitados, pode acontecer que uma solução processe o mesmo trabalho mais do que uma vez, ou seja, a solução pode ser uma sequência de afectação cíclica.

Tratando-se de uma relaxação do modelo original, este segundo modelo tem um espaço de soluções mais largo do que o Modelo 1 já que permite a existência de ciclos. Aplicado a uma instância de três trabalhos (e *m* máquinas), tem como espaço de soluções as sequências de afectação acíclicas, mais as seguintes sequências de afectação cíclicas: (0,1,2,1,4), (0,1,3,1,4), (0,2,1,2,4), (0,2,3,2,4), (0,3,1,3,4) e (0,3,2,3,4).

Com uma implementação computacional apropriada, a complexidade deste algoritmo é, no pior dos casos, O(n^2T) porque a rede tem (n+1)T+1 estados e o tempo de computação para cada um deles é 2n (n acções por estado com n somas e n comparações).

Este novo problema é um caso particular do problema saco-mochila, e é também NP-difícil. Contudo, existem algoritmos pseudo-polinomiais para o resolver, baseados em programação dinâmica.

As colunas adicionadas ao PPR serão seleccionadas de sequências de afectação acíclicas ou cíclicas. Contudo, a integralidade binária das variáveis de decisão y_p^k e o primeiro conjunto de restrições do PP garantem que as soluções inteiras não contêm sequências de afectação acíclicas.

Note-se que Desrochers, Desrosiers e Solomon (1992) e van den Akker, Hurkens e Savelsbergh (2000) usaram uma aproximação semelhante na formulação dos seus problemas.

Na próxima secção iremos descrever uma versão deste modelo que elimina os ciclos de ordem 2.

5.5.3. Modelo 3

Um número muito significativo de sequências de afectação cíclicas inclui partes do tipo (i,j,i). Estes ciclos são conhecidos como ciclos de ordem 2 (*2-cycles*). Para o problema TSP (*Travelling Salesman Problem*), Houck, Picard, Queyranne e Vemuganti (1980) mostraram que para um problema com n nodos, o rácio do número de caminhos admissíveis com ciclos de ordem 2 relativamente ao número de caminhos admissíveis sem ciclos de ordem 2 é de $(1+1/(n-3))^{n-3}$. Este rácio é indicativo da redução do número de caminhos admissíveis que é possível eliminar do espaço de soluções quando os ciclos de ordem 2 são eliminados.

A versão do modelo de eliminação dos ciclos de ordem 2 que vamos descrever é baseada no procedimento proposto por Houck, Picard, Queyranne e Vemuganti (1980). O procedimento de eliminação dos ciclos de ordem 2, basicamente, regista, para cada estado (j,t), duas sequências de afectação: $H^k(j,t)$, a sequência de afectação com o mínimo custo reduzido (*best*), que completa o trabalho j no instante de tempo t imediatamente após ter completado o trabalho i no instante de tempo t' e $H_2^k(j,t)$, a sequência de afectação com o mínimo custo reduzido (*next best*), que completa o trabalho j no instante de tempo t imediatamente após ter completado o trabalho v no instante de tempo t'', tal que $v \neq i$. Da mesma forma, as variáveis $g(j,t)$ e $g_2(j,t)$ registam o número de trabalhos processados nas sequências de afectação correspondentes aos estados $H^k(j,t)$ e $H_2^k(j,t)$, respectivamente.

Se definirmos $h(j,t)$ como sendo o trabalho precedente associado ao estado (j,t), as novas equações de recorrência são:

$$H^k(0,a_k) = -\upsilon_k;$$
$$H^k(j,t) = \min_{(i,j)\in A}\{[H^k(i,t') + \bar{c}_{ij}^k \mid j \neq h(i,t'), t' \leq t - p_{jk} - s_{ij}],$$
$$[H_2^k(i,t') + \bar{c}_{ij}^k \mid j = h(i,t'), t' \leq t - p_{jk} - s_{ij}]\}$$

$$H_2^k(j,t) = \min_{(i,j)\in A}\{[H^k(i,t') + \bar{c}_{ij}^k \mid j \neq h(i,t'), i \neq h(j,t),$$
$$t' \leq t - p_{jk} - s_{ij}\,], [H_2^k(i,t') + \bar{c}_{ij}^k \mid j \neq h(i,t'), i = h(j,t), t' \leq t - p_{jk} - s_{ij}]\},$$

para todos os j,k,t tal que $j \in N, k \in M$ e $\left(r_j + p_{jk}\right) \leq t \leq T^k$

Este modelo de programação dinâmica tem um espaço de soluções maior do que o do Modelo 1 mas mais pequeno do que o Modelo 2 já que elimina as sequências de afectação com ciclos de ordem 2. A Figura 5.6 e a Figura 5.7 pretendem ilustrar esta situação.

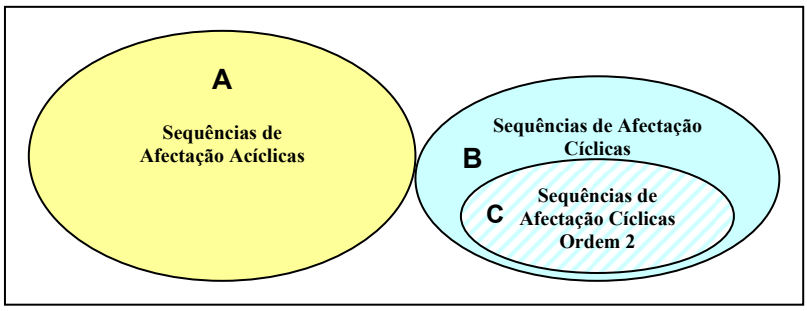


Figura 5.6 – Espaço de soluções do Modelo 2 - (A∪B)

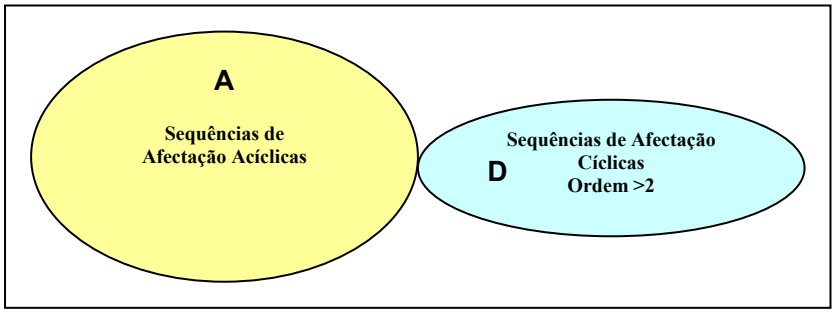


Figura 5.7 – Espaço de soluções do Modelo 3 - A∪D (D=B\C)

Aplicado à mesma instância de três trabalhos (e *m* máquinas), podemos verificar que levaria à eliminação de todas as sequências de afectação acíclicas já que todas elas apresentam ciclos de ordem 2 - (0,1,2,1,4), (0,1,3,1,4), (0,2,1,2,4), (0,2,3,2,4), (0,3,1,3,4) e (0,3,2,3,4).

É também um caso particular do problema saco-mochila, logo NP-difícil, mas, à semelhança do Modelo 2, existem algoritmos pseudo-polinomiais para o resolver, baseados em programação dinâmica.

5.5.4. Sequências de Afectação de Custo Reduzido Decrescente

Nesta secção, vamos apresentar um método que, tirando partido da estrutura especial dos subproblemas, consegue reduzir o espaço de estados do Modelo 3, melhorando de uma forma significativa a eficiência do algoritmo de programação dinâmica.

Como já foi referido, o espaço de soluções do Modelo 3 inclui sequências de afectação acíclicas e cíclicas, sendo que as sequências de afectação cíclicas possuem ciclos de ordens superiores a 2.

No processo de geração de colunas, dizemos que uma sequência de afectação, acíclica ou cíclica, é atractiva quando o seu custo reduzido é negativo e, como tal, pode ser adicionada ao PPR como uma nova coluna. Para cada máquina, pretendemos encontrar a sequência de afectação de custo reduzido mais negativo. Ou seja, vamos enumerar todas as soluções do espaço de soluções do modelo 3 com o objectivo de identificar a sequência de afectação de menor custo reduzido.

Contudo, uma solução inteira para a formulação PP somente incluirá sequências de afectação acíclicas porque qualquer sequência de afectação que inclua mais do que uma vez o mesmo trabalho, ao tomar um valor igual a 1, viola a restrição relativa ao processamento desse trabalho. No entanto, as sequências de afectação cíclicas são admissíveis para a relaxação linear, porque as variáveis de decisão podem tomar valores fraccionários.

Daí que, se em qualquer momento do processo de geração de colunas soubéssemos da não existência de mais sequências de afectação acíclicas com custos reduzidos negativos, poderíamos terminar o processo já que a solução actual da relaxação linear do PPR seria a melhor solução para a relaxação linear do PP. A questão é que, usando o Modelo 3, não sabemos da não existência de mais sequências de afectação acíclicas com custos reduzidos negativos, e, como tal, só podemos parar o processo de geração de colunas quando não conseguirmos gerar mais sequências de afectação, acíclicas ou cíclicas, com custos reduzidos negativos.

No estudo que a seguir desenvolvemos, iremos identificar propriedades dos subproblemas que nos permitem não só saber se não existem mais sequências de afectação acíclicas com custos reduzidos negativos como também reduzir significativamente o espaço de estados do Modelo 3.

Definição 5.1

Uma sequência de afectação é chamada de sequência de afectação de custo reduzido decrescente (CRD) se os custos reduzidos dos seus arcos (j_h,j_{h+1}), $\bar{c}^k_{j_h,j_{h+1}}$, *para todos os h tal que* $1 \le h \le H$ *e* $H \ge 2$, *e também para quando* $0 \le h \le H$ *e* $H=1$, *são negativos. Caso contrário, é chamada de sequência de afectação de custo reduzido não-decrescente.*

Proposição 5.1

Para funções objectivo regulares e tarefas independentes, as sequências de afectação acíclicas de custo reduzido mais negativo são sequências de afectação de custo reduzido decrescente.

Prova. O custo reduzido de uma sequência de afectação, $\bar{c}^k_p$, é uma função do custo reduzido dos seus arcos (j_h,j_{h+1}), $\bar{c}^k_{j_h,j_{h+1}}$, que, para funções objectivo regulares, são também função dos tempos de conclusão $C^k_{j_1},C^k_{j2},...,C^k_{j_H}$. Como os tempos de preparação

das máquinas e os tempos de processamento são maiores do que zero, temos que $C^k_{j_h} < C^k_{j_{h+1}}$, e, de uma forma geral, $C^k_{j_1} < C^k_{j2} < ... < C^k_{j_H}$.

A prova do resultado considera dois casos:

Caso 1. $1 \leq h \leq H$ e $H \geq 2$ - Consideremos uma sequência de afectação acíclica *de custo reduzido não-decrescente p*, que incluí um único arco (j_h, j_{h+1}), para todos os *h* tal que $1 \leq h \leq H$ e $H \geq 2$, com um custo reduzido não negativo, $\bar{c}^k_{j_h, j_{h+1}} \geq 0$. Como as tarefas são independentes (não existem relações de precedência entre elas), se o único arco de custo reduzido não negativo (j_h, j_{h+1}) e, consequentemente, (j_{h+1}, j_{h+2}), forem removidos da sequência de afectação *p* e o arco (j_h, j_{h+2}) for adicionado, obteremos uma sequência de afectação acíclica *p'*, em que, como os tempos de preparação das máquinas satisfazem o princípio das desigualdades triangulares, $s_{ij} + s_{jv} \geq s_{iv}$, para todos os $i,j,v \in N$, temos que $C^{k'}_{j_{h+2}} \leq C^k_{j_{h+2}}, ..., C^{k'}_{j_H} \leq C^k_{j_H}$ (note-se que a igualdade é possível pela existência de datas de disponibilidade para as tarefas) e, consequentemente, $c^k_{j_h, j_{h+2}} \leq c^k_{j_{h+1}, j_{h+2}}$. Mais ainda, como os custos dos arcos são por definição não negativos, temos que $c^k_{j_h, j_{h+2}} \leq c^k_{j_h, j_{h+1}} + c^k_{j_{h+1}, j_{h+2}}$.

O custo reduzido da sequência de afectação *p'*, pode ser obtido pela seguinte expressão:

$$\bar{c}^k_{p'} = \bar{c}^k_p + \left(c^k_{j_h, j_{h+2}} - \pi_{j_{h+2}}\right) - \left(c^k_{j_h, j_{h+1}} - \pi_{j_{h+1}}\right) - \left(c^k_{j_{h+1}, j_{h+2}} - \pi_{j_{h+2}}\right) .$$

Simplificando e rearranjando os termos temos,

$$\bar{c}^k_{p'} = \bar{c}^k_p + c^k_{j_h, j_{h+2}} - \left(c^k_{j_h, j_{h+1}} + c^k_{j_{h+1}, j_{h+2}}\right) + \pi_{j_{h+1}} .$$

Como $c^k_{j_h, j_{h+2}} \leq c^k_{j_h, j_{h+1}} + c^k_{j_{h+1}, j_{h+2}}$ logo $\bar{c}^k_{p'} \leq \bar{c}^k_p + \pi_{j_{h+1}}$. Dado que o custo reduzido do arco (j_h, j_{h+1}) é não negativo, $\left(c^k_{j_h, j_{h+1}} - \pi_{j_{h+1}}\right) \geq 0$, e $c^k_{j_h, j_{h+1}} \geq 0$, temos que $\pi_{j_{h+1}} \geq 0$ e, consequentemente, $\bar{c}^k_{p'} \leq \bar{c}^k_p$.

Temos também que o custo reduzido do arco (j_h, j_{h+2}) é negativo, $\left(c^k_{j_{h+1}, j_{h+2}} - \pi_{j_{h+2}}\right) < 0$, e sendo $c^k_{j_h, j_{h+2}} \leq c^k_{j_{h+1}, j_{h+2}}$, temos que $c^k_{j_h, j_{h+2}} - \pi_{j_{h+2}} \leq c^k_{j_{h+1}, j_{h+2}} - \pi_{j_{h+2}} < 0$, ou seja, o custo

reduzido do arco (j_{h+1},j_{h+2}) é negativo, o que faz com que todos os arcos da sequência de afectação *p'* tenham custos reduzidos negativos e que esta seja uma *sequência de afectação de custo reduzido decrescente*, o que prova que, para funções objectivo regulares e tarefas independentes, é sempre possível obter, a partir de uma sequência de afectação acíclica *de custo reduzido não-decrescente*, uma sequência de afectação acíclica *de custo reduzido decrescente* cujo custo reduzido é menor ou igual ao da sequência de afectação original.

Note-se que quando o primeiro trabalho a processar for igual à configuração inicial da máquina, $j_1=l_k$, o tempo de preparação da máquina é nulo, $s_{lkj1}=0$, possibilitando que $s_{0j}+s_{jv}<s_{0v}$. Nestes casos, o arco (j_0,j_1), da sequência de afectação acíclica de custo reduzido mais negativo, pode ter um custo reduzido não negativo. Esta é a razão que leva a que o arco (j_0,j_1) não esteja considerado na Definição 5.1 para sequências de afectação em que $H\geq 2$, isto é, quando a sequência de afectação inclui pelos menos dois trabalhos.

Caso 2. $0\leq h\leq H$ e $H=1$ – Este caso considera as sequências de afectação que incluem um único trabalho e, como tal, são acíclicas. Estas sequências de afectação são constituídas por apenas dois arcos, (j_0,j_1) e (j_1,j_{n+1}) e, sendo por definição $\bar{c}^k_{j_1,j_{n+1}}=0$, é fácil de verificar que qualquer sequência de afectação em que o custo reduzido do arco fonte (j_0,j_1) tenha um valor negativo, $\bar{c}^k_{j_0,j_1}<0$, tem um custo reduzido, $\bar{c}^k_p$, de menor valor do que no caso contrário.□

Exemplo 5.6 – Sequência de afectação de custo reduzido decrescente

Na Figura 5.8 está representada uma sequência de afectação acíclica, $(0,1,2,3,n+1)$, para uma instância com três trabalhos.

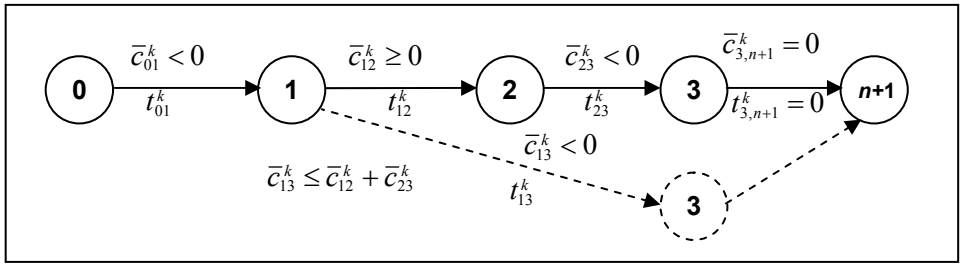


Figura 5.8 – Sequência de afectação de custo reduzido decrescente

É fácil de verificar que ao removermos da sequência de afectação o arco de custo reduzido não negativo (1,2), obtemos uma nova sequência de afectação também acíclica (0,1,3,*n*+1). Como os tempos de preparação satisfazem o princípio das desigualdades triangulares temos que $t_{13}^k \leq t_{12}^k + t_{23}^k$ o que implica que $\bar{c}_{13}^k \leq \bar{c}_{23}^k$. Então, a nova sequência de afectação é de custo reduzido decrescente e tem um custo reduzido menor ou igual ao custo reduzido da sequência de afectação original.□

De referir que a condição das tarefas serem independentes deixa de ser satisfeita aquando da introdução das restrições de partição e, como tal, este resultado não é directamente aplicável na fase de partição e geração de colunas. Esta questão será mais desenvolvida na Subsecção 5.6.1 onde iremos descrever as alterações necessárias a este modelo.

Proposição 5.2

Para funções objectivo regulares e tarefas independentes, se não existirem sequências de afectação de custo reduzido decrescente, com $\bar{c}_p^k$ *<0, então não existem sequências de afectação acíclicas com custo reduzido negativo.*

Prova. É fácil de verificar, a partir da Proposição 5.1, que se as sequências de afectação acíclicas de custo reduzido mais negativo são sempre *sequências de afectação de custo reduzido decrescente*, então se não existirem *sequências de afectação de custo reduzido decrescente*, também não existem sequências de afectação acíclicas de custo reduzido negativo.□

Estas são propriedades muito interessantes dos subproblemas que nos permitem que, durante a resolução da relaxação linear do PP, em qualquer estado rotulado (*i*,*t'*) de uma sequência de afectação parcial em geração no subproblema, só necessitemos de considerar os estados subsequentes (*j*,*t*) com custo reduzido negativo, para obter *sequências de afectação de custo reduzido decrescente*, reduzindo os espaços de soluções dos subproblemas. Isto equivale a remover das redes de programação dinâmica todos os arcos com custos reduzidos não negativos, com a excepção já referida para os arcos fonte (0, *j*) em que os tempos de preparação das máquinas sejam nulos.

O novo modelo de programação dinâmica é o seguinte:

$$H^k(0,a_k) = -\upsilon_k;$$
$$H^k(j,t) = \min_{(i,j)\in A}\{[H^k(i,t') + \bar{c}_{ij}^k \mid \bar{c}_{ij}^k < 0 \text{ or } s_{ij} = 0, j \neq h(i,t'), t' \leq t - p_{jk} - s_{ij}],$$
$$[H_2^k(i,t') + \bar{c}_{ij}^k \mid \bar{c}_{ij}^k < 0 \text{ or } s_{ij} = 0, j = h(i,t'), t' \leq t - p_{jk} - s_{ij}]\}$$

$$H_2^k(j,t) = \min_{(i,j)\in A}\{[H^k(i,t') + \bar{c}_{ij}^k \mid \bar{c}_{ij}^k < 0 \text{ or } s_{ij} = 0, j \neq h(i,t'), i \neq h(j,t), t' \leq t - p_{jk} - s_{ij}],$$
$$[H_2^k(i,t') + \bar{c}_{ij}^k \mid \bar{c}_{ij}^k < 0 \text{ or } s_{ij} = 0, j \neq h(i,t'), i = h(j,t), t' \leq t - p_{jk} - s_{ij}]\},$$

para todos os j,k,t tal que $j \in N, k \in M$ e $r_j + p_{jk} \leq t \leq T^k$

Os resultados computacionais preliminares mostram que este método reduz os tempos computacionais do Modelo 3 num factor superior a 7,0.

5.5.5. Um Algoritmo de Programação Dinâmica para a Resolução dos Subproblemas

Para a implementação do novo modelo de programação dinâmica (descrito na subsecção anterior), vamos adoptar o método *reaching* para tirar vantagem do facto das soluções em que o trabalho *j* é concluído no tempo *t'* dominam as soluções em que o trabalho *j* é concluído num tempo *t>t'*.

Definamos S como o conjunto que regista os estados (j,t) rotulados mas ainda não resolvidos. Sejam também B_{jt} e $B2_{jt}$ estruturas que registam a informação acerca das sequências de afectação *best* e *next best*, respectivamente, que vão desde o nodo "0" ao nodo j, tendo concluído o trabalho j no instante de tempo t, e que nos permite reconstruir para trás (*backward*) as respectivas sequências de afectação.

Dado que as sequências de afectação podem conter ciclos, torna-se necessário definir um limite superior para a duração das sequências de afectação, T^k. Para o nosso problema, T^k depende de vários factores: da máquina, dos tempos de preparação, das datas de disponibilidade das máquinas e das datas de disponibilidade dos trabalhos. Isto torna o seu cálculo numa tarefa que não é simples quando pretendemos obter um limite superior de qualidade (apertado). Por este motivo, vamos usar um limite superior óbvio que é n, o número total de trabalhos a processar. A variável $g(j,t)$ regista o número de trabalhos processados na sequência de afectação que vai desde o nodo "0" até ao nodo j, tendo completado o processamento do trabalho j no instante de tempo t.

O algoritmo de programação dinâmica é o seguinte:

Inicialização

$H^k(0,a_k) = -\upsilon_k,\ g(0,a_k) = 0, S = \{(0,a_k)\}, \bar{c}_p^{k*} = +\infty, j^{k*} = 0, t^{k*} = 0, T^k = a_k;$

$H^k(j,t) = +\infty,\ H_2^k(j,t) = +\infty, B_{jt} = \text{Null}, B2_{jt} = \text{Null},$ para todo o $j \in N$ e $t \le T^k$

While $S \neq \{\ \}$

Select estado $(i,t') \in S$ com menor t' e $g(i,t') < n$

$S = S - \{(i,t')\};$

For $j = 1,...,n$

If $(i,j) \in A$

$t = \max(r_j, t' + s_{ij}) + p_{jk},\ \bar{c}_{ij} = \max(t - d_j, 0) w_j - \pi_j;$

if $t > T^k$

$H^k(j,t) = +\infty,\ H_2^k(j,t) = +\infty, B_{jt} = \text{Null}, B2_{jt} = \text{Null};$

$T^k = t;$

If $j = B_{it'} \to i$ (if 2 - cycle)

$\bar{c}_p = H_2^k(i,t') + \bar{c}_{ij}, q = g_2(i,t');$

$cycle2 = \text{true};$

else

$\bar{c}_p = H^k(i,t') + \bar{c}_{ij}, q = g(i,t');$

$cycle2 = \text{false};$

If $(\bar{c}_{ij} < 0$ or $s_{ij} = 0)$ and $q < n$

If $\bar{c}_p < H^k(j,t)$ (actualiza "best")

If $j \neq B_{jt} \to i$ (actualiza " next - best")

$H_2^k(j,t) = H^k(j,t), g_2(j,t) = g(j,t);$

$B2_{jt} \to i = B_{jt} \to i,\ B2_{jt} \to t = B_{jt} \to t;$

$B2_{jt} \to cycle2 = B_{jt} \to cycle2;$

If $H^k(j,t) = +\infty$ then (novo estado)

$S = S + \{(j,t)\};$ (adiciona aos estados rotulados)

$H^k(j,t) = \bar{c}_p, g(j,t) = q + 1;$

$B_{jt} \to i = i,\ B_{jt} \to t = t',\ B_{jt} \to cycle2 = cycle2;$

If $H^k(j,t) < H^{k*}$ then (regista estado de mínimo custo reduzido)

$\bar{c}_p^{k*} = H^k(j,t), j^{k*} = j, t^{k*} = t;$

Else If $\bar{c}_p < H_2^k(j,t)$ and $j \neq B_{jt} \to i$ (regista " next - best")

$H_2^k(j,t) = \bar{c}_p, g_2(j,t) = q + 1;$

$B2_{jt} \to i = i,\ B2_{jt} \to t = t',\ B2_{jt} \to cycle2 = cycle2;$

5.6. O Algoritmo de Partição e Avaliação

Nesta secção, propomos um algoritmo de partição e avaliação para determinar a solução óptima inteira do nosso problema.

A solução óptima da relaxação linear do PP é obtida quando os subproblemas não são capazes de gerar mais colunas (sequências de afectação) com custo reduzido negativo. Se os valores das variáveis de decisão y_p^k são todos inteiros, então a solução óptima da relaxação linear do PP é também a solução óptima do problema original (PI-I). Caso contrário, alguns dos valores das variáveis de decisão y_p^k são fraccionários, sendo necessário explorar uma árvore de pesquisa para encontrar a solução óptima inteira.

O desenho do esquema de partição e avaliação é de grande importância num processo de geração de colunas porque a partição é feita a partir das colunas com soluções fraccionárias, e estas decisões têm que ser compatíveis com a estrutura do subproblema para a geração de novas colunas.

Em cada nodo da árvore de pesquisa resolvemos uma relaxação linear do PPR, com restrições adicionais impostas aos *m* subproblemas, usando o algoritmo de geração de colunas.

5.6.1. Estratégia de Partição

A estratégia apresentada faz partição sobre os arcos e não sobre as variáveis de decisão do PP. A estratégia tradicional de efectuar partição sobre as variáveis de decisão do PP torna-se impraticável para os ramos em que as variáveis de decisão tomam o valor zero.

Note-se que é perfeitamente possível fixar o valor de uma variável de decisão y_p^k com valor fraccionário no valor inteiro de 1, já que esta informação é facilmente transferida para o subproblema fixando o valor do fluxo em cada arco da sequência de afectação no valor 1, isto é, removendo da rede todos os outros arcos incidentes nos nodos visitados pela respectiva sequência de afectação.

Contudo, é impraticável fixar directamente o valor da variável y_p^k no valor 0 porque seria necessário explorar muitas das combinações dos valores de fluxo (0,1) para os arcos de uma sequência de afectação. Por este motivo, iremos efectuar partição sobre as variáveis de decisão x_{ij}^k do problema original (PI-I).

A estratégia de partição adoptada é baseada na representação em rede do problema. A raiz da árvore de pesquisa corresponde à rede *G* e os nodos descendentes são versões da rede *G* modificadas pela imposição de restrições sobre caminhos admissíveis. Cada iteração do processo de partição cria dois novos nodos na árvore de pesquisa que serão resolvidos através do algoritmo de geração de colunas.

Se o valor da solução óptima do nodo avaliado é maior ou igual ao valor do incumbente o nodo é abandonado porque a exploração dos seus descendentes nunca fornecerá uma solução melhor que o incumbente. Caso contrário, o processo de partição continua e são criados dois novos nodos na árvore de pesquisa.

O fluxo total no arco (i,j) das sequências de afectação da máquina k, $p \in P^k$, x_{ij}^k, é expresso em função das variáveis de decisão y_p^k, pela seguinte expressão:

$$x_{ij}^k = \sum_{p \in P^k} x_{ijp}^k y_p^k, \forall (i,j) \in A, k \in M \tag{5.29}$$

É fácil de verificar que:

- Se os valores das variáveis básicas não-nulas forem inteiros, então os valores de fluxo nos arcos, x_{ij}^k, das respectivas sequências de afectação, também são inteiros;
- Se existirem arcos com valores de fluxo fraccionários, então também existem variáveis básicas com valores fraccionários;
- Se todas as variáveis básicas apresentarem valores binários (0,1), então os valores de utilização das máquinas também são binários.

O seguinte resultado, derivado para os problemas de máquinas paralelas (idênticas, uniformes ou não-idênticas) com funções de custo aditivas $\sum w_j C_j$ e $\sum w_j U_j$, também se aplica ao problema em estudo.

Proposição 5.3 (Chen, Z.-L., and W. Powell, 1999[a])
Se todos os valores de fluxo nos arcos, x_{ij}^k, são binários, então os valores das variáveis básicas também são binários.

Este resultado deve-se ao facto das colunas geradas pelo algoritmo de geração de colunas serem sempre diferentes o que implica que, no caso de existirem colunas de valor fraccionário, pelo menos uma delas possui um arco que não está presente nas restantes colunas fraccionárias, sendo que o fluxo neste arco será também fraccionário.

Resulta do atrás exposto que, uma solução óptima em que todos os valores de fluxo nos arcos são binários é também uma solução óptima para o problema original (PI-I).

5.6.2. Regra de Selecção do Nodo

No algoritmo de partição e avaliação é, normalmente, necessário tomar dois tipos de decisões. Uma é chamada de selecção do nodo a explorar que define a forma de percorrer a árvore de pesquisa. A outra é a selecção da variável com valor fraccionário sobre a qual se irá efectuar a partição.

A selecção do próximo nodo a explorar é feita usando a regra *best-first*. Os nossos testes computacionais preliminares mostraram que esta regra tem um desempenho superior a outras regras de selecção do nodo (Figura 5.9).

A Figura 5.9 apresenta um gráfico comparativo do desempenho de três regras de selecção do nodo da árvore de pesquisa a explorar. A primeira é a regra *depth-first* (primeiro em profundidade) que é muito usada na prática. A segunda é uma combinação da regra *depth-first* com a regra *best-first* (primeiro o melhor). Nesta regra, se o nodo avaliado não é abandonado, é aplicada a regra de primeiro em profundidade. Caso

contrário, é aplicada a regra de o melhor primeiro. Por último, a terceira regra é a regra *best-first*.

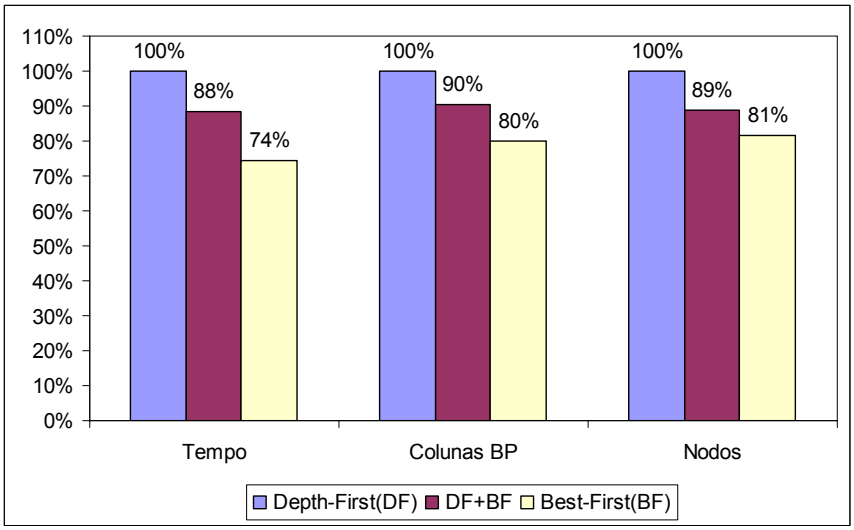


Figura 5.9 – Comparação de regras de selecção do nodo

Foram avaliadas três medidas de desempenho: o tempo computacional, o número de colunas geradas na fase de partição e geração de colunas e o número de nodos avaliados na árvore de pesquisa. Os resultados mostram que o desempenho da regra *best-first* é superior às outras duas tanto em tempo computacional como também no número de colunas e nodos avaliados.

5.6.3. Regra de Selecção da Variável

Uma das regras de selecção da variável para partição mais referida na literatura revista na Secção 4.3.3, e uma das mais usadas na prática, é a regra do arco com valor total de fluxo mais fraccionário - $\min|x_{ij}^k - 0{,}5|$. Contudo, para algumas das instâncias do nosso problema, esta regra não produz bons resultados já que são geradas muitas partições na árvore de pesquisa sem que o valor da solução nos nodos descendentes se altere de forma a permitir um rápido progresso para a solução óptima. Este fenómeno de "mergulho na árvore de pesquisa" resulta em tempos computacionais incomportáveis.

Conforme foi referido na Secção 3.4, um princípio geral a que as regras de partição devem obedecer é a de que as decisões mais fortes devem ser tomadas no início. A análise da estrutura das soluções para este problema revelou uma certa hierarquia de decisões ao longo da sequência de afectação em que as decisões mais importantes são tomadas no início, começando pela decisão entre afectar um trabalho de tempo de preparação nulo ou um outro, até ao momento em que os trabalhos começam a apresentar atrasos.

A partir daqui, a afectação dos trabalhos à máquina segue, na maior parte dos casos, uma lógica de prioridade decrescente, isto é, tipicamente os trabalhos menos prioritários ficam na cauda da sequência de afectação. Por outro lado, é fácil de perceber que, numa sequência de afectação, os trabalhos a montante afectam todos os trabalhos subsequentes ou, dito de outra forma, quanto mais a montante da sequência de afectação um trabalho se encontrar, maior será a sua influência no resultado global da sequência de afectação.

Com base no atrás exposto, desenvolvemos uma regra de partição multicritério, que chamamos de regra MFMC (Mais Fraccionário e Mais Cedo), que passamos a descrever.

A regra de partição MFMC, pondera dois valores para cada arco (i,j) com valor de fluxo não nulo:

1. O valor do desvio à integralidade - $\min|x_{ij}^k - 0{,}5|$;

2. A menor posição que o arco (i,j) ocupa na sequência de afectação da máquina k.

Relativamente ao ponto 2, ponderamos de forma diferente os chamados arcos fonte $(0,j)$. Estes arcos ao serem seleccionados para partição apresentam uma vantagem adicional, para além da sua posição mais a montante, que é o facto de definirem uma posição fixa na sequência de afectação, isto é, definem que o primeiro trabalho a ser processado na máquina k é o trabalho j. Mais ainda, definem implicitamente as posições dos arcos de ligação subsequentes na sequência de afectação.

Se o arco $(0,j)$ tem a posição 0, o arco (j,v) terá obrigatoriamente a posição 1 na sequência de afectação, e assim sucessivamente. Pelo contrário, se a restrição de partição for sobre um arco $(i,j)|i\neq 0$ e $j\neq n+1$, este arco poderá aparecer em qualquer posição da sequência de afectação, excepto nas posições 0 e H.

A Figura 5.10 apresenta um gráfico comparativo do desempenho das duas regras de partição acima referidas: mais fraccionário e MFMC. Foram avaliadas três medidas de desempenho: o tempo computacional, o número de colunas geradas na fase de aplicação do método de partição e geração de colunas e o número de nodos avaliados na árvore de pesquisa.

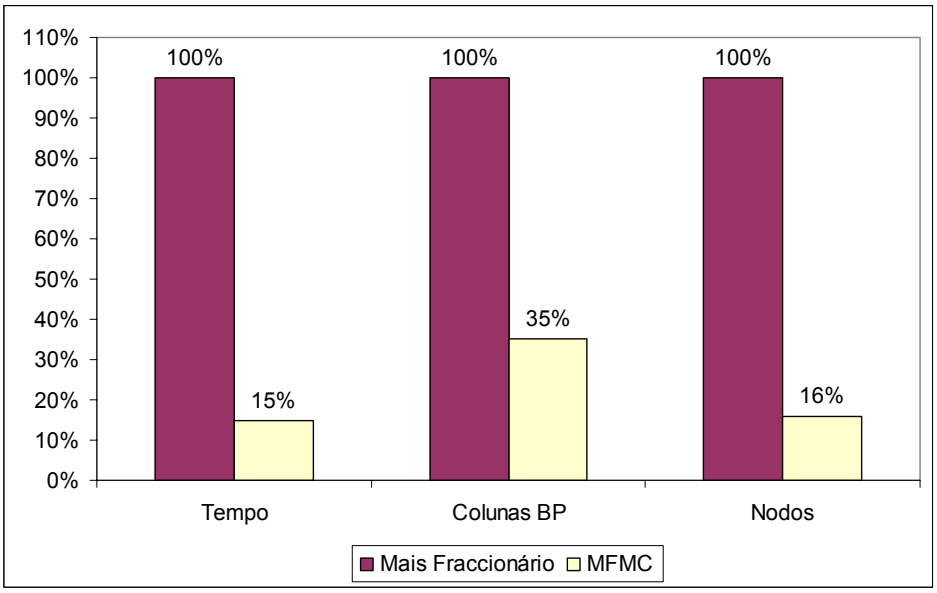


Figura 5.10 – Comparação de regras de partição

Os resultados mostram que o desempenho da regra MFMC é claramente superior à regra do mais fraccionário tanto em tempo computacional como também no número de colunas geradas e nodos avaliados.

5.6.4. Implementação da Estratégia de Partição

A estratégia de partição é implementada no valor de fluxo nos arcos, x_{ij}^k, da rede que representa as sequências de afectação para a máquina *k*. Para as variáveis básicas não-nulas ($y_p^k > 0$), somamos o valor do fluxo nos respectivos arcos, x_{ij}^k. Depois, atribuímos uma pontuação aos arcos de acordo com a seguinte expressão:

$$Pontuação = 0{,}25\beta - \left| x_{ij}^k - 0{,}5 \right| - \rho\alpha \mid 0 < x_{ij}^k < 1, \alpha = 0{,}3\left(\frac{m}{n}\right) \tag{5.30}$$

A variável ρ=0,1,…,*H*, representa a posição do arco na sequência de afectação e a variável β é uma variável binária que toma o valor 1 caso se trate de um arco fonte (0,*j*) e 0 caso contrário. A constante α representa um factor de correcção ao número de posições existentes numa sequência de afectação que é função da relação entre o número de trabalhos e o número de máquinas.

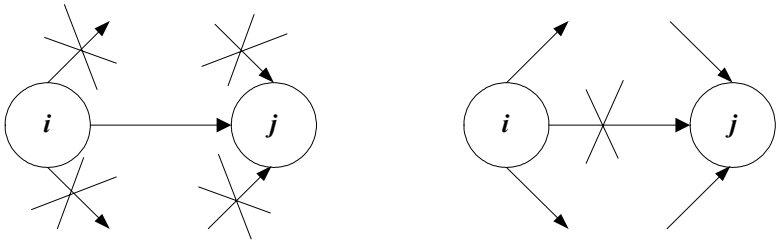
A variável fraccionária com a maior pontuação é seleccionada para partição.

Para a variável seleccionada para partição são criados dois ramos na árvore de pesquisa:

- Um impondo que o fluxo total no arco (*i*,*j*), na máquina *k*, é igual a um - $x_{ij}^k = 1$, i.e., estabelece uma relação de precedência $i \prec j$ entre os trabalhos *i* e *j* e estabelece que o arco (*i*,*j*) só pode ser usado na rede *G* do subproblema que representa a máquina *k*;
- O outro impondo que o fluxo total no arco (*i*,*j*), na máquina *k*, é igual a zero - $x_{ij}^k = 0$, i.e., o arco (*i*,*j*) não pode ser usado na rede *G* do subproblema que representa a máquina *k*.

Como já foi referido, as restrições de partição têm de ser compatíveis com a estrutura do subproblema. Isto implica que a rede *G* que define o subproblema seja modificada para fazer reflectir as restrições de partição impostas:

- Se x_{ij}^k toma o valor zero, o arco (i,j) é simplesmente removido da rede G do subproblema que representa a máquina k, eliminando a geração de sequências de afectação para a máquina k que incluam o arco (i,j) - Figura 5.11 b);
- Se x_{ij}^k toma o valor 1, os arcos $(i,v) \in A|v \neq j$ e $(l,j) \in A|l \neq i$ são removidos da rede G do subproblema que representa a máquina k- Figura 5.11 a).



a) b)

Figura 5.11 – Restrições de partição na rede do subproblema

Exemplo 5.7 – Restrições de partição sobre os arcos

Na Figura 5.12 estão representadas as modificações introduzidas para compatibilizar o nosso subproblema com o problema primal restrito, pelas restrições de partição sobre arcos não fonte, $x_{12}^k = 1$ e $x_{23}^k = 1$, na rede da máquina k para um problema com três trabalhos.

Restrição de partição $x_{12}^k = 1$

Para compatibilizar o nosso subproblema com o problema primal restrito, vamos remover os arcos incidentes no nodo 2 com origem em nodos diferentes do nodo 1 – arcos (0,2) e (3,2), e os arcos com origem no nodo 1 que incidem em nodos diferentes do nodo 2 – arcos (1,3) e (1,n+1). Note-se que a posição do arco (1,2) nas sequências de afectação (caminhos) na rede modificada a) não fica estabelecida, já que, na sequência de afectação (0,1,2,3,n+1) aparece na posição 1 enquanto que na sequência de afectação (0,3,1,2,n+1) aparece na posição 2. De referir ainda que esta restrição de partição não

obriga a utilização do arco (1,2) em todas as sequências de afectação na rede modificada a); a sequência de afectação (0,3,*n*+1) é admissível e não inclui o arco (1,2).

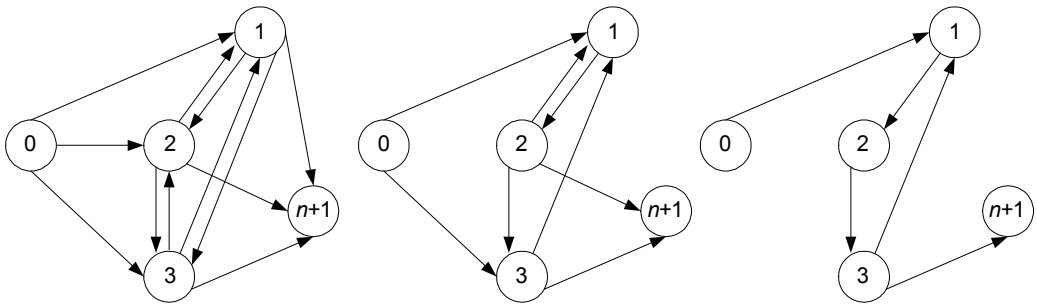


Figura 5.12 – Restrições de partição sobre arcos não fonte

Restrição de partição $x_{23}^k = 1$

Da mesma forma, vamos remover os arcos incidentes no nodo 3 com origem em nodos diferentes do nodo 2 – arco (0,3), e os arcos com origem no nodo 2 que incidem em nodos diferentes do nodo 3 – arcos (2,1) e (2,*n*+1).

Na Figura 5.13 estão representadas as modificações introduzidas pelas restrições de partição $x_{01}^k = 1$ e $x_{12}^k = 1$, sendo que a primeira é sobre um arco fonte, na rede da máquina *k* para um problema com três trabalhos. A rede modificada é apresentada na Figura 5.12 b). Podemos verificar agora que qualquer sequência de afectação inclui os arcos (1,2) e (2,3).

Restrição de partição $x_{01}^k = 1$

Para compatibilizar o nosso subproblema com o problema primal restrito, vamos remover os arcos incidentes no nodo 1 com origem em nodos diferentes do nodo 0 – arcos (2,1) e (3,1), e os arcos com origem no nodo 0 que incidem em nodos diferentes do nodo 1 – arcos (0,3) e (0,2). Note-se agora que a posição do arco (0,1) fica estabelecida (posição 0), e que a utilização do arco (0,1) é obrigatória para todas as sequências de afectação (caminhos) na rede modificada b).

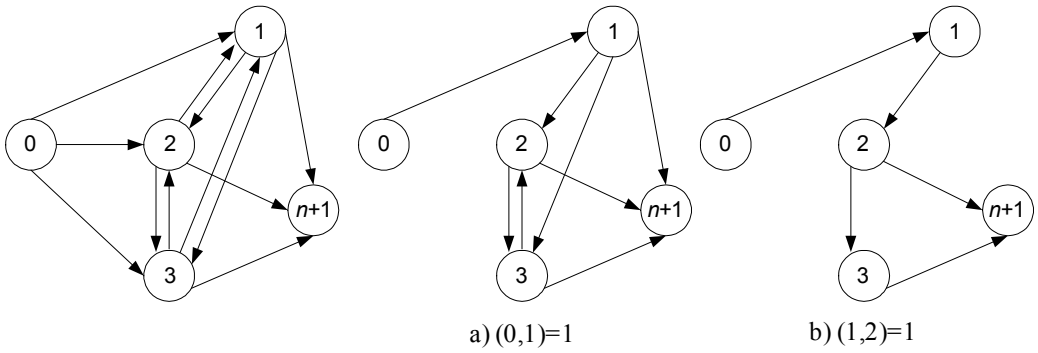


Figura 5.13 – Restrições de partição sobre arcos fonte

Restrição de partição $x_{12}^k = 1$

Da mesma forma, vamos remover os arcos incidentes no nodo 2 com origem em nodos diferentes do nodo 1 – arco (3,2), e os arcos com origem no nodo 1 que incidem em nodos diferentes do nodo 2 – arcos (1,3) e (1,n+1). □

Finalmente, se x_{ij}^k toma o valor zero, o custo de todas as colunas do PPR que usam o arco (i,j) e a máquina k são penalizados. Se x_{ij}^k toma o valor 1, o custo de todas as colunas do PPR que usam a máquina k, e os arcos $(i,v) \in A | v \neq j$ e $(l,j) \in A, l \neq i$, são penalizados. Em ambos os casos, a solução actual do PPR continua válida mas apresenta agora um custo muito elevado. Após a reoptimização do PPR, novas colunas são geradas pelo algoritmo de geração de colunas até que seja obtida uma nova solução óptima para o PPR.

5.6.5. Relaxação das Sequências de Afectação de Custo Reduzido Decrescente

Como já foi mencionado na Subsecção 5.5.4, após a introdução de restrições de partição, a Proposição 5.2 perde validade devido ao facto das restrições de partição imporem relações de precedência aos trabalhos sobre os quais incidem, logo, a condição de independência dos trabalhos deixa de ser satisfeita. De facto, pode existir uma única sequência de afectação acíclica atractiva que inclui um arco de custo reduzido não

negativo, sobre o qual incide uma restrição de partição, que obriga a que este arco tenha de ser considerado. As seguintes situações podem acontecer:

1. O arco (i,j) da rede que representa a máquina k toma o valor 1;

   O arco (i,j) tem um custo reduzido não negativo mas tem de ser considerado porque todos os arcos $(i,v) \in A|v \neq j$ são removidos da rede que representa a máquina k e existe um sequência de afectação acíclica $(0,\ldots,i,j,v,\ldots,n+1)$ com custo reduzido negativo que inclui a sequência parcial de trabalhos (i,j,v). Esta situação está representada na Figura 5.14. Podemos verificar que, como a ligação entre i e v não é possível de estabelecer devido à restrição de partição sobre o arco (i,j), se apenas forem geradas sequências de afectação de custo reduzido decrescente, o arco arco (i,j) não é considerado e, consequentemente, o arco de custo reduzido negativo (j,v) também não o é, podendo, assim, eliminar sequências de afectação acíclicas de custo reduzido negativo quando $\bar{c}_{ij}^{k} + \bar{c}_{jv}^{k} < 0$;

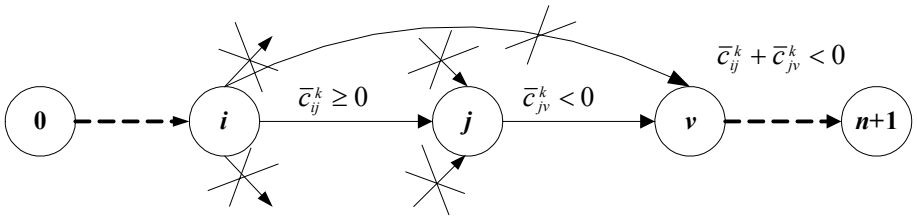


Figura 5.14 – Situação 1-a)

   O arco (i,j) tem um custo reduzido negativo mas não é considerado porque todos os arcos $(v,j) \in A|v \neq i$ são removidos da rede que representa a máquina k e existe um sequência de afectação acíclica $(0,\ldots,v,i,j,\ldots,n+1)$ com custo reduzido negativo que inclui a sequência parcial de trabalhos (v,i,j). Esta situação está representada na Figura 5.15. Verificamos que, como a ligação entre v e j não é possível de estabelecer devido à restrição de partição sobre o arco (i,j), se apenas forem geradas

sequências de afectação de custo reduzido decrescente, o arco arco (v,i) não é considerado e, consequentemente, o arco de custo reduzido negativo (i,j) também não o é, podendo, assim, eliminar sequências de afectação acíclicas de custo reduzido negativo quando $\bar{c}_{vi}^{k} + \bar{c}_{ij}^{k} < 0$;

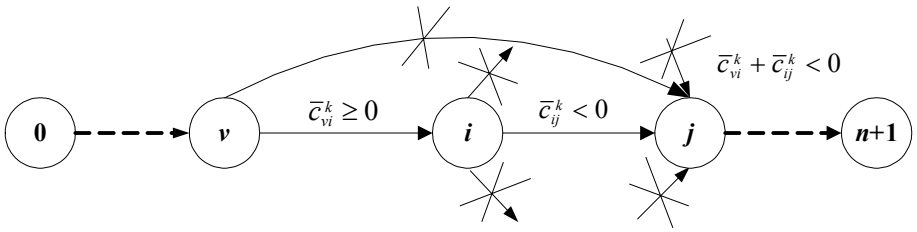


Figura 5.15 – Situação 1-b)

2. O arco (i,j) da rede que representa a máquina k toma o valor 0. Neste caso, os arcos (i,v) $\in A|v \neq j$ com um custo reduzido não negativo devem ser considerados porque o arco (i,j) é removido da rede que representa a máquina k e existe um sequência de afectação acíclica (0,…,i,v,j,…,n+1) com custo reduzido negativo que inclui a sequência parcial de trabalhos (i,v,j). Esta situação está representada na Figura 5.16 onde podemos verificar que, como a ligação entre i e j não é possível de estabelecer devido à restrição de partição sobre o arco (i,j), se apenas forem geradas sequências de afectação de custo reduzido decrescente, o arco arco (i,v) não é considerado e, consequentemente, o arco de custo reduzido negativo (v,j) também não o é, podendo, assim, eliminar sequências de afectação acíclicas de custo reduzido negativo quando $\bar{c}_{iv}^{k} + \bar{c}_{vj}^{k} < 0$;

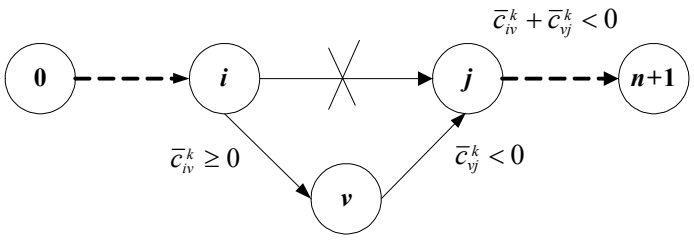


Figura 5.16 – Situação 2

Nas situações atrás descritas, a condição expressa na Proposição 5.2, de gerar apenas sequências de afectação com custo reduzido decrescente, é relaxada. Para além dos

estados (j,t) subsequentes aos estados rotulados (i,t') e com custo reduzido negativo, são também considerados os estados (j,t) com qualquer valor de custo reduzido quando existem restrições de partição sobre o trabalhos i ou j. É fácil de verificar que esta relaxação resolve qualquer uma das situações atrás referidas; na situação 1 a), o arco (i,j) passa a ser considerado porque tanto o trabalho i como o trabalho j são objecto de uma restrição de partição; na situação 1 b), o arco (v,i) passa a ser considerado porque o trabalho i é objecto de uma restrição de partição; na situação 2, o arco (i,v) passa a ser considerado porque o trabalho i é objecto de uma restrição de partição.

Isto implica que tenhamos de gerar nos subproblemas mais do que sequências de afectação de custo reduzido decrescente para não eliminar sequências de afectação acíclicas atractivas.

5.6.6. Convergência do Método

O número de caminhos e circuitos de uma rede (sequências de afectação acíclicas e cíclicas) é finito, embora exponencialmente elevado relativamente aos nodos da rede. Por outro lado, as restrições de partição sobre os arcos garantem que, na solução inteira, não vão existir caminhos com ciclos (sequências de afectação com trabalhos a serem processados mais do que uma vez) e, sendo o número destas restrições de partição também finito, existe a garantia de o método obter a solução óptima num número finito de passos, que pode ser exponencial.

Capítulo 6

Estabilização e Aceleração da Geração de Colunas

Neste capítulo, fazemos uma apresentação global de estratégias de estabilização e aceleração do algoritmo de geração de colunas e estudamos a implementação daquelas que nos pareceram mais adequadas ao problema em estudo. Por último, propomos um método de aceleração do algoritmo de geração de colunas baseado na restrição do espaço primal.

6.1. Introdução

O algoritmo de geração de colunas é conhecido pela sua lenta convergência. Enquanto que soluções próximas do valor óptimo são atingidas de uma forma relativamente rápida, a partir daqui, a progressão do algoritmo até à solução óptima é feita de uma forma lenta, exibindo uma longa cauda. Este fenómeno, conhecido na literatura por efeito de cauda longa (*tailing off effect*), é induzido por comportamentos indesejáveis tais como a degenerescência do primal ou a excessiva oscilação das variáveis duais. Este tópico foi objecto de muito trabalho de investigação nos últimos anos e foram propostas algumas estratégias para a sua correcção. Globalmente, estas estratégias beneficiam da sua adaptação às características específicas do problema.

6.2. Limites Inferiores

Uma das ideias mais simples para corrigir o efeito de cauda longa é a de terminar a execução do algoritmo antes de este fenómeno ocorrer. Mais precisamente, o algoritmo de geração de colunas é interrompido quando o desvio da solução actual do problema de programação linear relativamente a um limite inferior calculado (para problemas de minimização), é inferior a um determinado valor. Infelizmente, o algoritmo de geração de colunas faz uma aproximação ao valor óptimo por cima, isto é, em cada iteração gera um novo limite superior para o valor óptimo. Por este motivo, terminar prematuramente o algoritmo de geração de colunas pode conduzir a um valor da solução da relaxação linear que não é um limite inferior para a solução óptima inteira.

O método proposto originalmente por Gilmore e Gomory (1963) de terminar o algoritmo de geração de colunas quando o decréscimo relativo da função objectivo, após um determinado número de iterações, fosse inferior a um dado valor percentual, pode também conduzir a valores de solução não óptimos pela ocorrência de patamares temporários de estagnação do valor da função objectivo.

Contudo, Lasdon (1970) apresenta um método simples e fácil de implementar para o cálculo de um limite inferior da relaxação linear em cada iteração do algoritmo de geração de colunas. Este método e uma sua extensão, mais geral, são apresentados na Subsecção 6.2.1. Na Subsecção 6.2.2, alargamos a utilização destes limites inferiores aos problemas de programação inteira.

6.2.1. Limites Inferiores na Programação Linear

Seja $\overline{Z}_{PPR}$ o valor da solução óptima do problema primal restrito e $\overline{c}_k^*$ o valor do menor custo reduzido do subproblema correspondente à máquina *k*, para uma dada iteração do algoritmo de geração de colunas. Seja também Z_{RLPP}^* o valor da solução óptima da relaxação linear do problema principal (PP). Um limite inferior para Z_{RLPP}^* pode ser obtido a partir das expressões gerais propostas por Lasdon (1970).

O modelo da relaxação linear do problema principal (RLPP) é o seguinte

$$\text{RLPP}:$$
$$\min\ Z = \sum_{k\in M}\ \sum_{p\in P^k} c_p^k y_p^k, \tag{6.1}$$
$$\text{sujeito a}:$$
$$\sum_{k\in M}\ \sum_{p\in P^k} a_{jp}^k y_p^k = 1, \forall j \in N \tag{6.2}$$
$$\sum_{p\in P^k} y_p^k \le 1, \forall k \in M \tag{6.3}$$
$$y_p^k \ge 0, \forall p \in P^k, k \in M \tag{6.4}$$

e na sua forma aumentada,

$$\text{RLPP}:$$
$$\min\ Z = \sum_{k\in M}\ \sum_{p\in P^k} c_p^k y_p^k, \tag{6.5}$$
$$\text{sujeito a}:$$
$$\sum_{k\in M}\ \sum_{p\in P^k} a_{jp}^k y_p^k = 1, \forall j \in N \tag{6.6}$$
$$\sum_{p\in P^k} y_p^k + s_k = 1, \forall k \in M \tag{6.7}$$
$$y_p^k, s_k \ge 0, \forall p \in P^k, k \in M \tag{6.8}$$

onde s_k representa as variáveis de folga.

Multiplicando as restrições pelas respectivas variáveis duais e efectuando a sua soma fica:

$$\sum_{j\in N}\ \sum_{k\in M}\ \sum_{p\in P^k} a_{jp}^k y_p^k \pi_j = \sum_{j\in N} \pi_j \tag{6.9}$$
$$\sum_{k\in M}\ \sum_{p\in P^k} y_p^k \upsilon_k + \sum_{k\in M} s_k \upsilon_k = \sum_{k\in M} \upsilon_k \tag{6.10}$$

Subtraindo estas expressões à função objectivo, obtemos

$$Z - \sum_{j\in N}\pi_j - \sum_{k\in M}\upsilon_k = \sum_{k\in M}\sum_{p\in P^k} c_p^k y_p^k - \sum_{j\in N}\sum_{k\in M}\sum_{p\in P^k} a_{jp}^k y_p^k \pi_j - \sum_{k\in M}\sum_{p\in P^k} y_p^k \upsilon_k - \sum_{k\in M} s_k \upsilon_k \qquad (6.11)$$

ou

$$Z - \sum_{j\in N}\pi_j - \sum_{k\in M}\upsilon_k = \sum_{k\in M}\sum_{p\in P^k} y_p^k \left(c_p^k - \sum_{j\in N} a_{jp}^k y_p^k \pi_j - \upsilon_k \right) - \sum_{k\in M} s_k \upsilon_k \qquad (6.12)$$

onde podemos verificar que a expressão entre parêntesis representa o custo reduzido associado à coluna p da máquina k, $\bar{c}_p^k$. Como y_k^p é não negativo, $\bar{c}_p^k$ pode ser substituído pelo seu valor mínimo $\bar{c}_k^*$, ficando

$$Z \geq \sum_{j\in N}\pi_j + \sum_{k\in M}\upsilon_k + \sum_{k\in M}\left(\sum_{p\in P^k} y_p^k \right)\bar{c}_k^* - \sum_{k\in M} s_k \upsilon_k \qquad (6.13)$$

Da mesma forma, como $\sum_{p\in P^k} y_p^k \leq 1, \forall k \in M$ podemos simplificar a expressão (6.13),

$$Z \geq \sum_{j\in N}\pi_j + \sum_{k\in M}\upsilon_k + \sum_{k\in M}\bar{c}_k^* - \sum_{k\in M} s_k \upsilon_k \qquad (6.14)$$

Da teoria da dualidade sabemos que o valor da solução óptima actual do problema primal restrito, $\bar{Z}_{PPR}$, é igual ao valor da solução dual correspondente $\bar{W}_{PPR} = \sum_{j\in N}\pi_j + \sum_{k\in M}\upsilon_k$. Substituindo esta expressão na expressão (6.14) obtemos um limite inferior para a relaxação linear do problema primal restrito,

$$Z_{RLPP}^* \geq \bar{Z}_{PPR} + \sum_{k\in M}\bar{c}_k^* - \sum_{k\in M} s_k \upsilon_k \qquad (6.15)$$

Os resultados computacionais preliminares da aplicação deste método ao nosso problema mostram que, na prática, não reduz ou reduz marginalmente o número de iterações do algoritmo de geração de colunas.

Este limite inferior pode também ser usado para calcular, em cada iteração, o desvio percentual relativamente à solução óptima e, desta forma, terminar o algoritmo quando for atingido um determinado valor máximo para esse desvio.

Quando $c \geq 0$, podemos usar um método mais geral (Farley, 1990) para o cálculo de um limite inferior da relaxação linear em cada iteração do algoritmo de geração de colunas que não depende de $\bar{c}_k^*$.

Os modelos primal e dual da relaxação linear do problema principal, na forma matricial, são os seguintes:

RLPP:	DRLPP:	
$\min\ Z = cy,$	$\max\ W = \pi + \upsilon,$	(6.16)
sujeito a:	sujeito a:	
$ay = 1$	$a\pi + \upsilon \leq c$	(6.17)
$y \leq 1$		(6.18)
$y, c \geq 0$	π urs, $\upsilon \leq 0, c \geq 0$	(6.19)

Seja $(\bar{\pi}, \bar{\upsilon})$ a solução óptima dual para uma dada iteração do algoritmo de geração de colunas. Definamos $\lambda_p^k = a_p^k \bar{\pi} + \bar{\upsilon}_k, \forall p \in P^k, \forall k \in M$. Se $\lambda_p^k \leq 0, \forall p \in P^k, \forall k \in M$ então $(\bar{\pi}, \bar{\upsilon})$ é a solução óptima do dual completo e, pela teoria da dualidade, $\bar{y}$ é a solução óptima do primal completo. Caso contrário, seja $c_u / \lambda_u = \min_{k \in M, p \in P^k} \left\{ c_p^k / \lambda_p^k \mid \lambda_p^k > 0 \right\}$. Como $c_u \geq 0$, resulta que $c_u / \lambda_u \geq 0$.

Teorema 6.1 – *Para o problema RLPP $c\bar{y}c_u / \lambda_u$ é um limite inferior para o valor óptimo da função objectivo.*

Prova. (a prova deste teorema decorre da que foi feita por Farley, 1990, para a classe de problemas $\min\{cx : Ax \geq b, x \geq 0, c \geq 0\}$).

Sabemos da teoria da dualidade que uma solução admissível para o problema dual (DRLPP) é um limite inferior para a solução óptima do problema primal (RLPP). Em particular, se $(\pi, \upsilon) c_u / \lambda_u$ é uma solução admissível para o modelo DRLPP, então o seu valor é um limite inferior para a solução óptima do modelo RLPP. Para estabelecer a

validade de $(\pi,\upsilon)c_u/\lambda_u$ no modelo DRLPP é necessário demonstrar que $(a\pi,\upsilon)c_u/\lambda_u \le c$ e que $\upsilon c_u/\lambda_u \le 0$. Para a restrição *p* da máquina *k* a primeira condição fica $(v_p^k\pi+\upsilon_k)c_u/\lambda_u \le c_p^k$ ou $\lambda_p^k c_u/\lambda_u \le c_p^k$. Se $\lambda_p^k \le 0$ então $\lambda_p^k c_u/\lambda_u \le 0 \le c_p^k$ já que $c_p^k \ge 0$. Se $\lambda_p^k > 0$ então é necessário que $c_u/\lambda_u \le c_p^k/\lambda_p^k$ o que é verdadeiro pela definição de c_u/λ_u. Como $c_u \ge 0$ resulta que $c_u/\lambda_u \ge 0$ e sendo $\upsilon \le 0$ a segunda condição $\upsilon c_u/\lambda_u \le 0$ é satisfeita, provando o resultado.□

No entanto, este método necessita de um maior esforço computacional para ser obtido, e por este motivo não foi utilizado.

6.2.2. Limites Inferiores na Programação Inteira

Quando pretendemos uma solução inteira, o único propósito da resolução da relaxação linear do problema de programação inteira é o de obter um limite inferior para a solução óptima do problema de programação inteira. Se as colunas apresentam valores inteiros para os custos (o que pode ser obtido através de um factor de escala apropriado), o processo de geração colunas pode ser terminado quando $\lceil Z_{PPR}^* \rceil = \lceil LI \rceil$, sendo *LI* um dos limites inferiores apresentados na Secção anterior. Este valor garante que a solução óptima inteira do problema original $\lceil Z^* \rceil = \lceil LI \rceil$.

Este limite inferior pode ser usado no esquema de partição e avaliação permitindo que um nodo da árvore de pesquisa seja abandonado quando $\overline{Z} = \lceil LI \rceil$, em que $\overline{Z}$ representa o valor actual do incumbente.

6.3. Estabilização do Algoritmo de Geração de Colunas

Uma outra forma de acelerar a convergência do algoritmo de geração de colunas é pela estabilização dos valores das variáveis duais. Os processos de geração de colunas são algoritmos de cortes duais (Lasdon, 1970). Da perspectiva do dual, os processos de

geração de colunas podem ser vistos da seguinte forma: em cada iteração, são adicionados cortes duais para eliminar a anterior solução dual não admissível. O espaço dual é sucessivamente restringido até que uma solução dual admissível seja encontrada.

A estabilização dos valores das variáveis duais é normalmente obtida através da restrição do espaço dual admissível ou, alternativamente, guiando os valores das variáveis duais. Esta é a estratégia seguida pelos métodos que a seguir descrevemos.

O método *boxstep* (Marsten, Hogan e Blankenship, 1975) segue a estratégia de guiar os valores das variáveis duais criando caixas (*boxes*) de dimensão fixa à volta dos valores da solução dual do problema primal restrito que funcionam como limites inferiores e superiores aos valores das variáveis duais. Este novo problema primal restrito é reoptimizado. Se a nova solução dual se encontrar na fronteira da caixa, esta é deslocada na direcção do limite atingido. Caso contrário, a solução óptima dual encontra-se no interior da caixa e esta é também a solução óptima do PPR original.

O *trust region method* de Kallehauge, Larsen e Madsen (2001) usa um conceito similar mas onde os intervalos (caixas) para os valores das variáveis duais podem ser ajustados automaticamente dependendo da forma como o dual do PPR se aproxima do dual Lagrangiano.

Du Merle, Villeneuve, Desrosiers e Hansen (1999), propuseram uma estratégia de estabilização que permite que, ao longo do processo de geração de colunas, haja soluções fora da caixa definida. A essas soluções é atribuída uma penalidade, definida por uma função linear por troços, que depende da distância à caixa. Além disso, a dimensão e o centro das caixas são actualizadas de uma forma dinâmica e é também introduzida uma pequena perturbação do lado direito do problema primal para ultrapassar problemas de degenerescência.

Mais recentemente Ben Amor, Desrosiers e Valério de Carvalho (2003) mostraram que, se conhecermos a priori uma solução dual óptima para o problema, esta informação pode ser usada para guiar o processo de geração de colunas e obter uma solução óptima muito mais rapidamente. Valério de Carvalho (2004), derivou cortes duais óptimos para

o problema de corte e mostrou que a introdução destes cortes duais antes do início do algoritmo de geração de colunas, o que corresponde a restringir o espaço dual durante todo o processo de geração de colunas, acelera o algoritmo.

Outros métodos, como o *bundle* (Urruty e Lemaréchal, 1993), *analytic center cutting plane* (du Merle, Goffin e Vial, 1998), foram usados para prevenir a variação excessiva das variáveis duais.

Nas Subsecções 6.3.1 e 6.3.2 estudamos a aplicação de dois métodos de estabilização do algoritmo de geração de colunas ao problema em estudo e na Subsecção 6.4 propomos um método de restrição do espaço primal que permite a resolução dos subproblemas de uma forma muito mais eficiente.

6.3.1. O Modelo de Cobertura de Conjuntos

O problema principal do nosso problema de programação de máquinas paralelas foi formulado, na Secção 5.4, com base no modelo de partição de conjuntos (ISP). Contudo, na implementação do algoritmo e durante a fase de relaxação linear, vamos usar uma formulação baseada no modelo de cobertura de conjuntos (ISC).

A diferença entre as duas formulações, do ponto de vista do dual, é a seguinte: no modelo ISP, os valores das variáveis duais, correspondentes a restrições do tipo igualdade, são não limitados em sinal, enquanto, no modelo ISC, as variáveis duais são não negativas. Portanto, ao adoptar o modelo ISC, estamos a restringir o espaço dual, o que influencia positivamente a convergência do processo de geração de colunas (Valério de Carvalho, 2004). Em consequência disso, a resolução da relaxação linear da formulação baseada no modelo de cobertura de conjuntos é numericamente muito mais estável do que a formulação baseada no modelo de partição de conjuntos.

A formulação do nosso problema com base no modelo ISC é a seguinte:

ISC:

$$\min \sum_{k\in M}\sum_{p\in P^k} c_p^k y_p^k, \tag{6.20}$$

sujeito a:

$$\sum_{k\in M}\sum_{p\in P^k} a_{jp}^k y_p^k \geq 1, \forall j \in N \tag{6.21}$$

$$\sum_{p\in P^k} y_p^k \leq 1, \forall k \in M \tag{6.22}$$

$$y_p^k \in \{0,1\}, \forall p \in P^k, k \in M \tag{6.23}$$

onde

$$a_{jp}^k = \sum_{i\in N\cup\{0\}|i\neq j} x_{ijp}^k, \forall j \in N, p \in P^k, k \in M \tag{6.24}$$

$$x_{ij}^k = \sum_{p\in P^k} x_{ijp}^k y_p^k, \forall (i,j) \in A, k \in M \tag{6.25}$$

O primeiro conjunto de restrições (6.20) garante que cada trabalho é processado pelo menos uma vez; o segundo conjunto de restrições (6.21) garante que cada máquina é utilizada no máximo uma vez. Tal como na formulação ISP, quando relaxamos a integralidade de y_p^k, (6.23) também podemos relaxar a restrição de ligação (6.25) entre y_p^k e x_{ij}^k.

Exemplo 6.1 – Convergência dos valores das variáveis duais

A Figura 6.1 e a Figura 6.2 mostram a convergência dos valores das variáveis duais das formulações baseadas nos modelos ISP e ISC, respectivamente, para uma instância de 10 máquinas e 70 trabalhos. No eixo das ordenadas estão representados os valores das variáveis duais referentes aos trabalhos e no eixo das abcissas está representado o número de colunas geradas.

Pela observação destas duas figuras, podemos constatar o problema referido no ponto 2 acima (os valores das variáveis duais oscilam entre valores negativos e positivos para o modelo ISP enquanto para o modelo ISC estão restringidos a valores positivos) e verificar que a convergência do processo de geração de colunas é aproximadamente 15% mais rápida na formulação ISC.

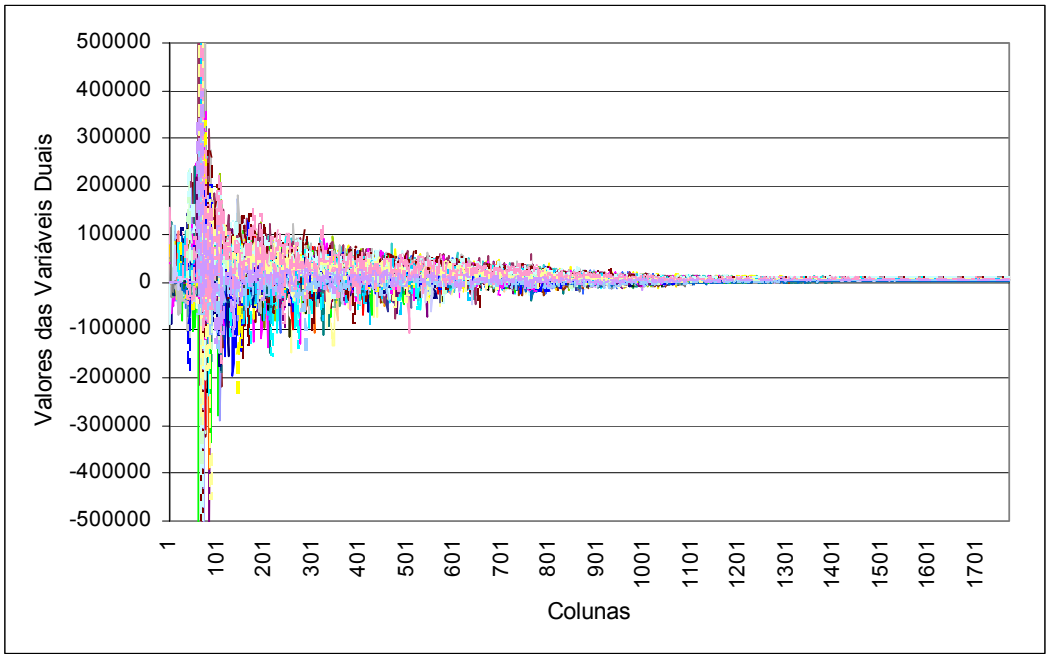


Figura 6.1 – Convergência dos valores das variáveis duais (ISP)

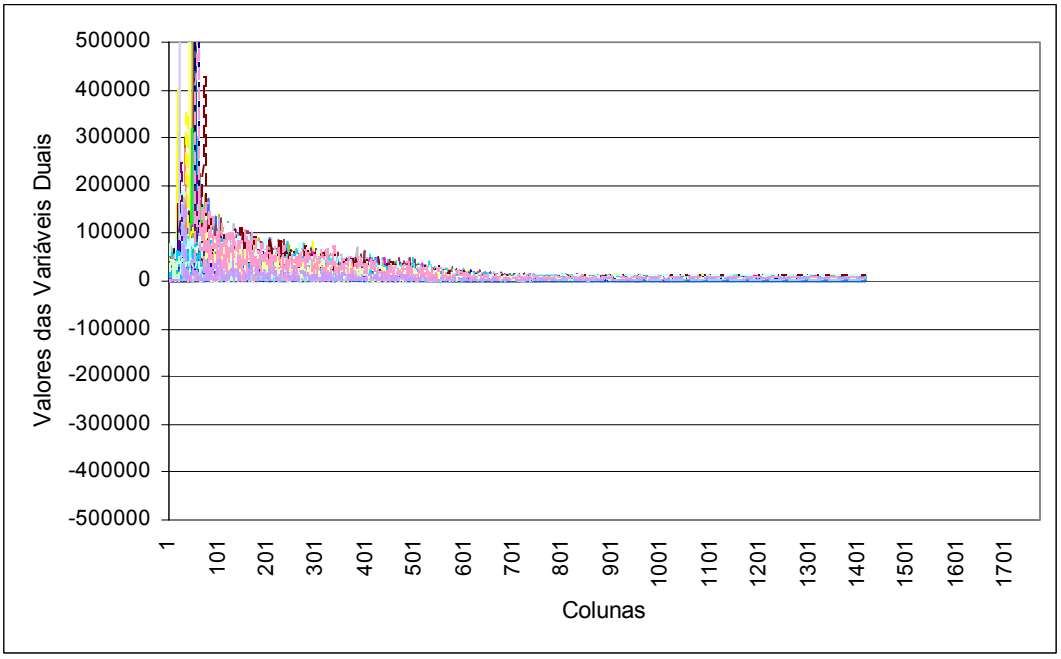


Figura 6.2 – Convergência dos valores das variáveis duais (ISC)

A formulação ISC constitui uma relaxação da formulação ISP. Em geral, sempre que uma subcoluna de uma coluna admissível define uma outra coluna também admissível e com menor custo, então uma solução óptima do problema de cobertura de conjuntos será também uma solução óptima do problema de partição de conjuntos.

Contudo, o nosso problema não se enquadra nesta definição por duas razões:

1. O custo associado a uma coluna (sequência de afectação) pode ser nulo;

2. Uma solução óptima para a formulação ISC pode ter um valor inferior à solução óptima para formulação ISP. Isto pode acontecer quando máquinas diferentes possuem configurações iniciais, l_k, iguais podendo ambas beneficiar de tempos de preparação nulos caso a primeira tarefa a ser processada não implique uma alteração na sua configuração, isto é, $j_1=l_k$. Nestes casos, o princípio das desigualdades triangulares, aplicado aos tempos de preparação das máquinas, pode ser violado e permitir soluções de menor custo quando a mesma tarefa é afectada a máquinas diferentes, com tempos de preparação nulos.

No entanto, a formulação ISC pode ser usada na resolução da relaxação linear do PP com vantagens para a eficiência do algoritmo de geração de colunas já que se verifica que o processo de geração de colunas é mais lento na fase inicial pelo facto dos valores das variáveis duais oscilarem bastante (Figura 6.1), e durante esta fase a formulação ISC apresenta melhor desempenho (Figura 6.2).

A formulação ISC é utilizada num processo em duas fases: numa primeira fase, o PP assume a forma da formulação ISC até que a solução da sua relaxação linear seja encontrada, através do procedimento de geração de colunas; na segunda fase, o PP assume a forma da formulação original (ISP) e é novamente resolvida a sua relaxação linear através do mesmo procedimento.

Exemplo 6.2 – Evolução dos valores da função objectivo

A Figura 6.4 mostra a evolução do valor da função objectivo para uma instância de 10 máquinas e 70 trabalhos. No eixo das ordenadas estão representados os valores da

função objectivo e no eixo das abcissas está representado o número de colunas geradas. A linha de cima representa o processo original de geração de colunas e a linha de baixo representa o processo de geração de colunas com estabilização.

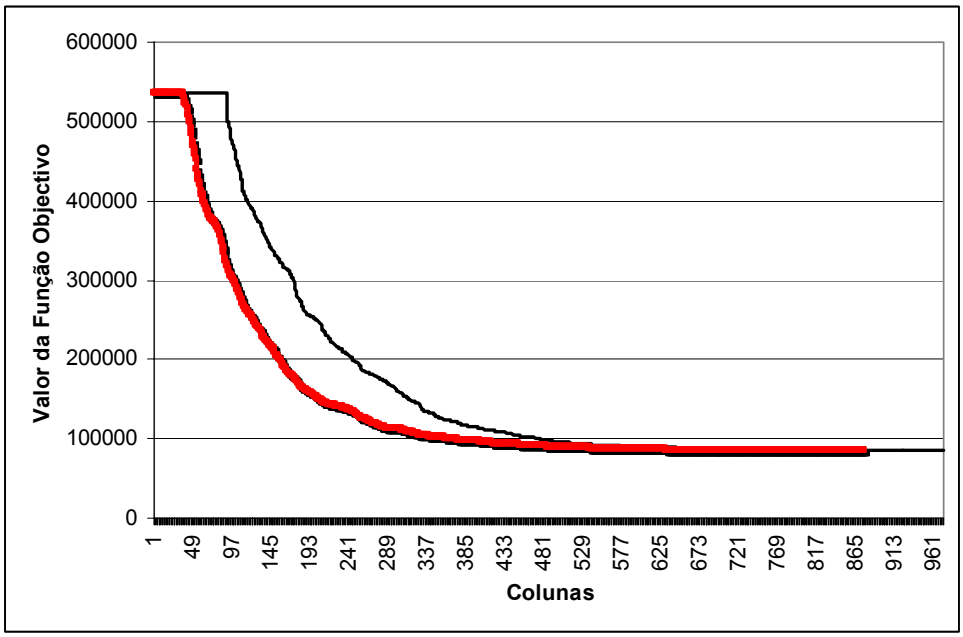


Figura 6.3 – Evolução dos valores da função objectivo

Mais importante que a visível redução do número de colunas geradas são os ganhos em tempo computacional para determinar a solução óptima da relaxação linear do PP. Estes ganhos são explicados não só pelo facto de serem geradas menos colunas mas também pelo facto dos subproblemas serem resolvidos mais rapidamente, já que as variáveis duais, ao assumirem valores mais baixos, proporcionam ganhos de eficiência ao método dos custos reduzidos decrescentes proposto na Secção 5.5.4.

Os resultados computacionais preliminares mostram que normalmente a solução óptima encontrada na primeira fase é também a solução óptima da relaxação linear da formulação original (ISP) e que este processo acelera o processo de geração de colunas em mais de 10%.

6.3.2. O Método de Estabilização

Baseados na ideia do método *boxstep* de Marsten, Hogan e Blankenship (1975), Du Merle, Villeneuve, Desrosiers e Hansen (1999), propuseram um conceito de caixa mais flexível e que pode ser usado directamente no algoritmo de geração de colunas. Definamos o problema primal $\tilde{P}$ e o seu dual $\tilde{D}$ da seguinte forma:

$$
\begin{array}{ll}
\tilde{P}: & \tilde{D}: \\
\min \quad c^T\tilde{x} - \delta_- y_- + \delta_+ y_+ & \max \quad b^T\tilde{\pi} - w_-\varepsilon_- - w_+\varepsilon_+ \\
\text{s. a: } A\tilde{x} - y_- + y_+ = b & \text{s. a: } A^T\tilde{\pi} \le c \\
\quad y_- \le \varepsilon_- & \quad -\tilde{\pi} - w_- \le -\delta_- \\
\quad y_+ \le \varepsilon_+ & \quad \tilde{\pi} - w_+ \le \delta_+ \\
\quad \tilde{x}, y_-, y_+ \ge 0 & \quad w_-, w_+ \ge 0
\end{array}
$$

No problema primal $\tilde{P}$, y_- e y_+ são vectores de variáveis de folga e excesso que introduzem uma perturbação do lado direito do primal, de valor situado no intervalo $[\varepsilon_-, \varepsilon_+]$, com o objectivo de ultrapassar os problemas de degenerescência. Estas variáveis são penalizadas na função objectivo pelos vectores δ_- e δ_+, respectivamente. No problema dual $\tilde{D}$, isto representa a penalização das variáveis duais $\tilde{\pi}$ quando estas se encontram fora do intervalo $[\delta_-, \delta_+]$, pelos montantes ε_- e ε_+, respectivamente.

Quando a solução óptima fornecida pelo modelo não é uma solução óptima do problema original, a dimensão das caixas é actualizada, o que equivale a redefinir as penalidades. Esta actualização é feita de uma forma dinâmica, tem um carácter heurístico e deve ter em conta as especificidades do problema em questão.

Na aplicação deste método ao nosso problema, seguimos uma estratégia semelhante à adoptada por Du Merle, Villeneuve, Desrosiers e Hansen (1999). O intervalo inicial de variação permitido às variáveis duais $\tilde{\pi}$ (que representam os trabalhos) é definido em função do valor da solução inicial do PPR e do número de trabalhos do problema:

$\overline{Z}/n \leq \pi_j \leq \overline{Z}/n + 500, \forall j \in N$. Os vectores dos parâmetros ε_- e ε_+ são seleccionados aleatoriamente nos intervalos [0,2] e [0,10^{-4}], respectivamente.

Sempre que um valor da solução actual das variáveis duais se encontra na fronteira do intervalo ou para além deste, o intervalo da respectiva variável é actualizado na direcção do limite atingido ou ultrapassado, pelo valor constante de 500, i.e., é somado ou subtraído o valor de 500 a ambos os limites do intervalo.

Quando o algoritmo atinge a solução óptima, as colunas relativas às variáveis y_- e y_+ são removidas do PPR modificado de forma a regressar à formulação original do PPR. O PPR original é então reoptimizado e a sua solução óptima obtida através do algoritmo de geração de colunas. Foram feitas algumas experiências de avaliação do impacte da aplicação desta estratégia ao problema em estudo, cujos resultados são apresentados de seguida.

A Figura 6.4 mostra a evolução do valor da função objectivo para uma instância de 10 máquinas e 70 trabalhos. No eixo das ordenadas estão representados os valores da função objectivo e no eixo das abcissas está representado o número de colunas geradas. A linha de cima representa o processo original de geração de colunas e a linha de baixo representa o processo de geração de colunas com estabilização.

É claramente visível uma redução significativa do número de colunas geradas para determinar a solução óptima da relaxação linear do PP. Em particular, o patamar inicial, constituído por uma sequência de soluções de igual valor, o que indicia uma forte degenerescência do problema primal, é praticamente eliminado.

Contudo, os ganhos em tempo computacional não são tão significativos dado o maior esforço computacional exigido por este método, em resultado das necessárias actualizações de valores. De realçar ainda que para obter bons resultados os valores dos parâmetros devem ser definidos tendo em conta as especificidades do problema. Por estes motivos, decidimos adoptar o método proposto na Secção 6.3.1 - O Modelo de Cobertura de Conjuntos, para estabilização do processo de geração de colunas, dados os bons resultados obtidos e a sua simplicidade e facilidade de implementação.

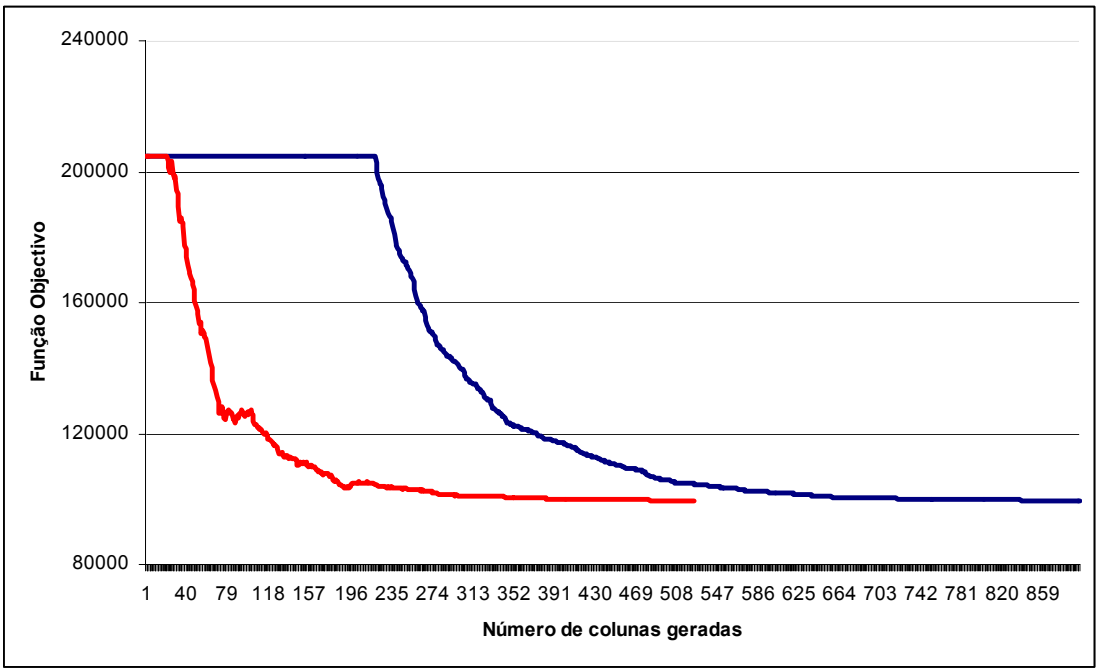


Figura 6.4 – Evolução dos valores da função objectivo

6.4. Estratégia de Restrição Primal

Neste trabalho introduzimos uma estratégia de restrição temporária do espaço de soluções primal que designamos por estratégia de restrição do primal (*primal boxstep*).

A ideia consiste em identificar características estruturais a que a solução óptima primal verosimilmente obedece, e traduzir essas características em restrições do espaço primal. Essas restrições definem um subdomínio do domínio original (uma caixa) no qual se determina uma solução óptima. Em seguida avalia-se se essa solução óptima encontrada no espaço restrito é solução óptima do domínio original. Caso não o seja, a restrição é progressivamente relaxada até a solução óptima do domínio original ser encontrada. Desejavelmente esta estratégia conduzirá a uma aceleração do algoritmo de geração de colunas.

As estratégias de restrição do espaço primal podem ser implementadas incluindo na formulação compacta as restrições encontradas. Contudo, este processo pode não se traduzir em melhorias significativas se implementado desta forma. A forma de implementar estas estratégias num processo de geração de colunas consiste em transferir essas restrições para o subproblema com o objectivo de tornar a sua resolução muito mais rápida.

Claramente, em processos de optimização deste tipo é sempre importante identificar propriedades da solução óptima primal que se possam traduzir em restrições do espaço primal que, quando transferidas para o subproblema, se possam traduzir numa aceleração da sua resolução. No método *primal boxstep* são derivadas características verosímeis das soluções óptimas que, não havendo garantia de ser observadas, podem contudo produzir um efeito semelhante. Em problemas menos estruturados onde a identificação dessas propriedades pode não ser fácil, o recurso a esta estratégia pode significar a possibilidade de resolver problemas de maior dimensão.

A motivação para o uso deste método é a seguinte. Muito possivelmente a solução encontrada no espaço primal restrito é uma solução óptima do problema original e todo o processo de geração de colunas decorre de uma forma muito mais rápida, não só por a resolução do subproblema ser mais rápida mas também por haver uma convergência mais rápida para o valor final atingido no processo de geração de colunas, dada a restrição das soluções que o subproblema pode enumerar.

Se a solução óptima não for encontrada, é pelo menos expectável que seja determinada uma solução de boa qualidade. Subsequentemente, quando se relaxa a restrição utilizada, garante-se que não é excluída nenhuma solução óptima. No entanto, são eliminadas implicitamente do processo de geração de colunas outras soluções com maior valor de função objectivo do que a solução encontrada que eventualmente seriam percorridas no processo normal.

Do ponto de vista do dual, este processo corresponde a uma relaxação do espaço dual. Conforme já foi referido, alguns autores verificaram que a restrição do espaço dual pode ser vantajosa. Não contrariando esse princípio, na prática, o que se passa é que, usando

este método, se encontra um espaço dual reduzido rapidamente, constituído por restrições duais válidas, não se enumerando muitas restrições que acabariam por ser redundantes.

É de salientar que este método é essencialmente diferente do método de geração rápida de colunas (*fast column generation*), usada por exemplo por Puchinger e Raidl (2004), em que se aplicam técnicas heurísticas para a geração de colunas numa primeira fase do processo, recorrendo-se depois à resolução exacta do subproblema para a determinação da solução óptima do problema original. No método de geração rápida de colunas não é possível identificar restrições do espaço primal. O que se faz é resolver o subproblema de uma forma aproximada procurando uma solução no domínio original completo. Além disso, quando se recorre a heurísticas, nem sempre é usada a informação dual que é gerada pelo problema primal restrito.

6.5. Aplicação da Estratégia de Restrição Primal ao Problema

A resolução dos subproblemas representa a maior parte do tempo computacional gasto na resolução do nosso problema de programação. O algoritmo de programação dinâmica é pseudopolinomial e a sua eficiência depende muito do valor de T (valor máximo para o tempo de conclusão das sequências de afectação geradas nos subproblemas). Os limites superiores (*upper bounds*) para o valor de T impõem limites à duração das sequências de afectação geradas pelos subproblemas. Um limite superior óbvio para T seria o valor do *makespan* da solução óptima para o problema de programação, caso este fosse conhecido *a priori.*

Uma forma diferente de definir limites superiores é limitando o tamanho das sequências de afectação geradas pelos subproblemas a um número máximo de trabalhos processados. Um limite superior trivial corresponde a limitar o tamanho das sequências de afectação geradas pelos subproblemas a um número máximo n (número total de trabalhos a processar) de trabalhos processados. Contudo, a este limite superior pode corresponder um elevado valor para T. Recorde-se que o facto das sequências de

afectação geradas nos subproblemas poderem conter ciclos obriga à definição de um limite superior para o seu tamanho.

Se bem que não conheçamos o valor do *makespan* da solução óptima *a priori*, é verosímil pensarmos que uma solução óptima para o nosso problema de programação deverá apresentar um certo balanceamento de cargas nas máquinas. O exemplo que se segue tenta ilustrar esta ideia.

Exemplo 6.3 – Programas balanceados

Vamos considerar um problema com duas máquinas e três trabalhos. Em cada um dos subproblemas, temos de resolver um problema de máquina única e três trabalhos. Na Figura 6.5, lado esquerdo, estão representados os diagramas de Gantt do espaço de soluções de um subproblema.

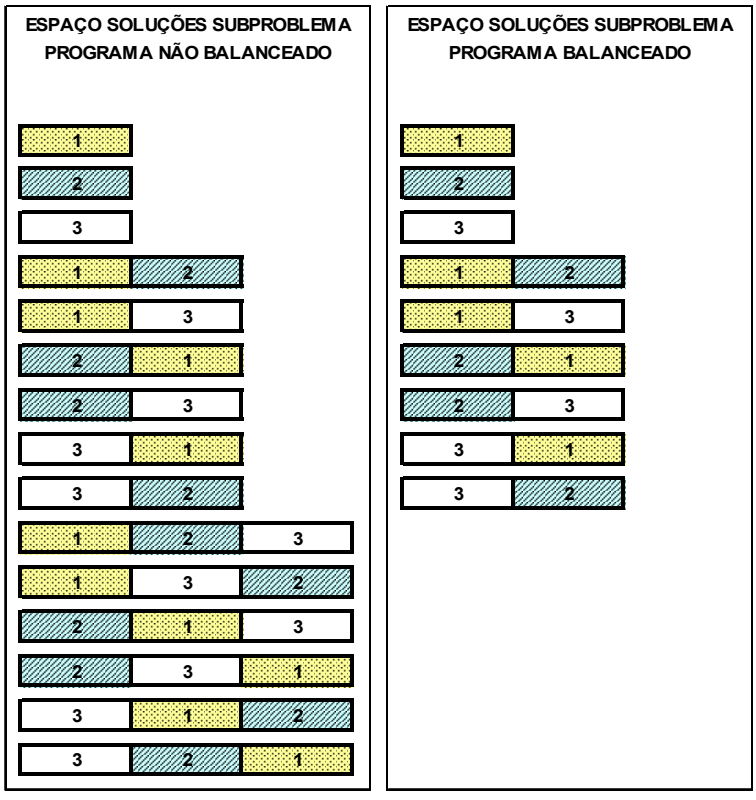


Figura 6.5 – Programas não balanceados/balanceados

Conforme foi referido, quando o problema apresenta mais do que uma máquina, parece razoável admitir que existirá um certo balanceamento de cargas entre as várias máquinas, isto é, os trabalhos são distribuídos pelas várias máquinas permitindo a sua conclusão mais cedo. Neste caso, o espaço de soluções de cada subproblema pode ser reduzido de uma forma significativa, conforme se tenta ilustrar no lado direito da Figura 6.5 onde está representado o espaço de soluções reduzido para um subproblema de um problema de programação com três trabalhos e duas máquinas.

Neste exemplo, o comprimento das sequências de afectação está reduzido de três trabalhos para dois trabalhos, permitindo eliminar as sequências de afectação com três trabalhos - (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2) e (3,2,1), que representam 40% do total. □

Esta ideia é ainda reforçada pela seguinte propriedade que mostra que a transferência de um trabalho de uma sequência de afectação mais longa para uma outra mais curta, dando origem a um programa mais balanceado, não aumenta global da solução nas condições a seguir referidas.

Propriedade 6.1

Seja i_H o último trabalho processado numa dada sequência de afectação da máquina k', com tempo de conclusão C'_{i_H}, e j_H o último trabalho processado numa dada sequência de afectação da máquina k, com tempo de conclusão C_{j_H}. Se o trabalho j_H pode ser transferido da última posição da sequência de afectação da máquina k para a última posição da sequência de afectação da máquina k', com tempo de conclusão C'_{j_H}, tal que $C_{j_H} \geq C'_{j_H} = \max(C'_{i_H} + s_{i_H j_H}, r_{j_H}) + p_{j_H k'}$ então, o custo total deste novo programa não é superior ao do anterior.

Prova. Como a função de custo é regular (não decrescente com os tempos de conclusão) e é função dos tempos de conclusão e a transferência do trabalho não afecta os restantes

trabalhos já que é feita da última posição de uma sequência para a última posição de uma outra, é óbvio que se $C'_{j_H} \leq C_{j_H}$ o custo total do novo programa não aumenta.□

Para pôr em prática esta ideia vamos, numa primeira fase, começar por restringir o conjunto de sequências de afectação que podem ser geradas nos subproblemas impondo um valor estimado para *T*. A solução óptima deste conjunto restrito de colunas é determinada. Se esta solução óptima não for a solução óptima do problema completo o valor de *T* é aumentado, permitindo gerar sequências de afectação de maior duração nos subproblemas (alargando o espaço de soluções dos subproblemas). Este processo é sucessivamente repetido até que a solução óptima do problema completo seja encontrada. Do ponto de vista do dual, significa utilizar um conjunto restrito de cortes duais para restringir o espaço dual e obter tentativamente uma solução dual de qualidade.

A verificação se, para um valor de *T* restringido, a solução óptima actual é também a solução óptima do problema completo, toma diferentes formas consoante se trate da fase de relaxação linear ou da fase de partição e geração de colunas.

6.5.1. Relaxação Linear

Seja $C^k_{j_H}$ o tempo de conclusão da sequência de afectação de maior duração da solução óptima para o actual valor de *T*. Para avaliar se esta solução óptima não é a solução óptima do problema completo o valor de *T* é aumentado para permitir a inclusão de mais um (qualquer) trabalho na sequência de afectação de maior duração.

Se o novo valor de *T* for $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$, passa a existir espaço para a inclusão de qualquer novo trabalho na sequência de afectação, porque este espaço é igual à soma do maior valor dos tempos de preparação com o maior valor dos tempos de processamento. Contudo, como as sequências de afectação devem ser de custos reduzidos decrescentes (ver Subsecção 5.5.4), este espaço adicional só pode ser usado por trabalhos com arcos de custo reduzido negativo.

O resultado que se segue mostra que, se os subproblemas não conseguem gerar novas sequências de afectação de custo reduzido decrescente para o novo valor de *T*, não existem mais colunas atractivas para o problema completo, para qualquer valor maior do que *T*.

Proposição 6.1

Durante a resolução da relaxação linear, se não existirem sequências de afectação atractivas de duração máxima T e os subproblemas não conseguirem gerar novas sequências de afectação de custo reduzido decrescente para um novo $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$*, então não existem mais colunas atractivas para qualquer valor maior do que T, e a solução óptima actual é a solução óptima para o problema completo*.

Prova. Como já foi referido, as sequências de afectação geradas na fase de relaxação linear são sequências de afectação de custo reduzido decrescente. O valor inicial (estimado) de *T* cobre o processamento de todos os trabalhos e, consequentemente, cobre todas as datas de disponibilidade dos trabalhos e das máquinas.

Seja Q^T o conjunto de sequências de afectação geradas nos subproblemas para o valor *T*, em cada iteração do algoritmo de geração de colunas. Quando todas as sequências de afectação $p \in Q^T$ tiverem custos reduzidos não negativos (colunas não atractivas) o valor de *T* é aumentado para $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$, permitindo a inclusão de qualquer trabalho $j \in N$.

Verifica-se facilmente que, como a função de custo é regular e aditiva, é preferível adicionar um novo trabalho no novo espaço de tempo disponibilizado do que em qualquer outro momento posterior a *T* e, consequentemente, se existir um trabalho *j* correspondente a um arco de custo reduzido negativo quando adicionado à sequência de afectação *p*, será adicionado a *p* gerando uma nova sequência de afectação de custo reduzido decrescente.□

A solução gerada pela heurística, descrita na Subsecção 7.1.2.2, é usada para determinar o valor inicial para T através da expressão, $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$, onde $C^k_{j_H}$ é o tempo de conclusão do último trabalho processado na sequência de afectação da máquina k gerada pela heurística. Note-se que o valor inicial de T permite a adição de qualquer trabalho $j \in N$ à sequência de afectação de maior duração gerada pela heurística.

Após a determinação do valor inicial para T, o processo de geração de colunas é efectuado em duas fases:

1. Na primeira fase, o valor estimado para T é usado para gerar sequências de afectação de custo reduzido decrescente nos subproblemas tal que $C^k_{j_H}$ é menor ou igual a T. Quando o algoritmo não conseguir gerar mais colunas de custo reduzido negativo, passamos á segunda fase;

2. Na segunda fase, o valor de T é ajustado para $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$ e o algoritmo de programação dinâmica é usado para tentar gerar mais colunas de custo reduzido negativo e três situações podem ocorrer:

   - Se forem geradas novas colunas de custo reduzido negativo, são adicionadas ao PPR, até que uma nova solução óptima seja encontrada e T é novamente ajustado repetindo-se a execução do ponto 2;
   - Se forem geradas novas colunas de custo reduzido decrescente mas não negativo, T é novamente ajustado repetindo-se a execução do ponto 2;
   - Se não forem geradas novas colunas o processo pára (observa-se a Proposição 6.1).

No limite, este processo pára quando for atingido o limite superior ao número de trabalhos que podem ser incluídos numa sequência de afectação que se mantém no valor n em todo este processo.

Este limite superior para o tempo reduz drasticamente o espaço de estados do modelo de programação dinâmica dos subproblemas e, durante o processo de geração de colunas, são tipicamente muito poucos os ajustes efectuados ao valor de *T*.

6.5.2. Partição e Geração de Colunas

Durante a fase de partição e geração de colunas, podem existir sequências de afectação acíclicas que incluem arcos (*i,j*), para $i,j \in N$, com custos reduzidos não negativos e, como tal, a Proposição 6.1 não se aplica. O resultado que se segue mostra que, se os subproblemas não conseguem gerar novas sequências de afectação para o novo valor de *T*, não existem mais colunas atractivas para o problema completo, para qualquer valor maior do que *T*.

Proposição 6.2

Durante a fase de partição e geração de colunas, se não existirem sequências de afectação atractivas de duração máxima T e os subproblemas não conseguirem gerar novas sequências de afectação para um novo $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$, *então não existem mais colunas atractivas para qualquer valor maior do que T, e a solução óptima actual é a solução óptima para o problema completo*.

Prova. A prova é semelhante à que foi efectuada para a Proposição 6.1. O novo valor de *T*, $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$, permite a inclusão de qualquer trabalho $j \in N$ e, como a função de custo é regular e aditiva, é preferível adicionar um novo trabalho no novo espaço de tempo disponibilizado do que em qualquer outro momento posterior a *T*.

A adição de um novo trabalho *j* só não ocorrerá em duas situações: quando o número de trabalhos incluídos na sequência de afectação for igual a *n* (limite superior baseado no número máximo de trabalhos que uma sequência de afectação pode conter), ou quando os trabalhos correspondentes ao arco a adicionar, (*i,j*), não estiverem abrangidos por

restrições de partição e o arco tiver um custo reduzido não negativo (ver Subsecção 5.6.1 - Estratégia de Partição). Caso estas duas situações não se verifiquem, um novo trabalho *j* é adicionado gerando uma nova sequência de afectação.□

O processo proposto na Subsecção anterior é alterado tomando a seguinte forma:

1. Na primeira fase, o valor estimado para *T* é usado para gerar sequências de afectação de custo reduzido decrescente nos subproblemas tal que $C^k_{j_H}$ é menor ou igual a *T*. Quando o algoritmo não conseguir gerar mais colunas de custo reduzido negativo, passamos á segunda fase;

2. Na segunda fase, o valor de *T* é ajustado para $T = \max_{k \in M} \{C^k_{j_H}\} + \max_{i,j \in N} \{s_{ij}\} + \max_{k \in M, j \in N} \{p_{jk}\}$ e o algoritmo de programação dinâmica é usado para tentar gerar mais colunas de custo reduzido negativo e três situações podem ocorrer:

   - Se forem geradas novas colunas de custo reduzido negativo, são adicionadas ao PPR, até que uma nova solução óptima seja encontrada e *T* é novamente ajustado repetindo-se a execução do ponto 2;
   - Se forem geradas novas colunas mas de custo reduzido não negativo, *T* é novamente ajustado repetindo-se a execução do ponto 2;
   - Se não forem geradas novas colunas nos subproblemas, a Proposição 6.2 é observada e o processo pára.

Os resultados computacionais preliminares mostram que esta estratégia de aceleração do algoritmo de geração de colunas, tendo em conta o conjunto das duas fases (relaxação linear e partição e geração de colunas), acelera o algoritmo num factor superior a 4.0, quando comparada com a estratégia de utilização do limite superior *n* para o número máximo de trabalhos que podem ser incluídos numa sequência de afectação.

Capítulo 7

Implementação Computacional e Resultados

Neste capítulo refere-se a implementação efectuada e apresenta-se a estrutura do programa desenvolvido. Para a obtenção de uma solução inicial, são descritos métodos baseados em soluções artificiais e é proposta uma heurística que foi desenvolvida tendo em conta a especificidade do problema. Apresentam-se ainda os resultados dos testes computacionais efectuados em instâncias com diferentes características e uma análise de sensibilidade do modelo desenvolvido a variações nos principais parâmetros do problema. Por último, é estudada a extensão do modelo a outras funções de custo regulares.

7.1. Implementação Computacional

7.1.1. Introdução

O programa que corresponde à implementação do método apresentado neste trabalho foi desenvolvido na linguagem de programação Microsoft Visual C++ Versão 6.0 e serve-se da biblioteca de funções CPLEX ("CPLEX Callable Library Version 6.0") ILOG (1998) para resolver os problemas de programação linear. A esse programa foi dado o nome de PMPNI. A descrição da sua estrutura é apresentada no Anexo A. Uma descrição sucinta do funcionamento da biblioteca de funções CPLEX é apresentada no

Anexo B. Os testes computacionais foram efectuados num computador portátil com um processador 1.8 GHz Pentium IV Mobile e 512 Mb de RAM.

7.1.2. Obtenção do Primeiro Problema Primal Restrito

Para inicializar o método de geração de colunas é necessário um problema primal restrito que seja válido, no sentido de permitir o cálculo do valor das variáveis duais que serão utilizadas no subproblema para avaliação da atractividade das colunas. Nesta Secção vamos apresentar dois modelos para a obtenção de uma solução inicial admissível baseados em variáveis artificiais e uma heurística desenvolvida para o mesmo efeito.

7.1.2.1. Solução Inicial Artificial

Uma forma de obter um problema primal restrito válido é incluir n variáveis artificiais que correspondem a n sequências de afectação em que a sequência de afectação artificial A_j processa o trabalho j, para j=1,2,..,n, e com um coeficiente na função objectivo que torne qualquer solução que inclua estas variáveis artificiais pior do que qualquer outra solução que não as inclua, constituindo uma matriz identidade $n \times n$ no quadro do problema primal restrito. Desta forma, garantimos que existe sempre uma solução que satisfaz todas as restrições relativas aos trabalhos (=1), isto é, todos os trabalhos são processados uma e uma só vez. Os coeficientes nulos das variáveis artificiais para as linhas das restrições relativas às máquinas (≤1), garantem, por sua vez, que estas restrições são sempre satisfeitas.

O quadro do problema primal restrito, para as variáveis artificiais, teria a seguinte configuração:

	A_1	A_2	…	A_n	
T_1	1		…		=1
T_2		1	…		=1
…	…	…	…	…	…
T_n			…	1	=1
M_1	0	0	…	0	<=1
M_2	0	0	…	0	<=1
…	…	…	…	…	…
M_m	0	0	…	0	<=1
C	M	M	…	M	

Figura 7.1 – Matriz identidade das variáveis artificiais

O problema primal restrito, baseado no modelo de partição de conjuntos seria então,

ISPA - I :

$$\min \sum_{k \in M} \sum_{p \in P^k} c_p^k y_p^k + MA_j, \tag{7.1}$$

sujeito a :

$$\sum_{k \in M} \sum_{p \in P^k} a_{jp}^k y_p^k + A_j = 1, \forall j \in N \tag{7.2}$$

$$\sum_{p \in P^k} y_p^k \leq 1, \forall k \in M \tag{7.3}$$

$$y_p^k \in \{0,1\}, \forall p \in P^k, \forall k \in M \tag{7.4}$$

onde a variável A_j é uma variável artificial e M é uma constante de valor muito elevado em comparação com os custos unitários das outras sequências de afectação.

Note-se que uma solução válida para o modelo de partição de conjuntos é também válida para a formulação baseada no modelo de cobertura de conjuntos.

Outra forma, mais simples, de obter um problema primal restrito válido é incluir uma única variável artificial que corresponda a uma sequência de afectação que use todos os trabalhos e com um coeficiente na função objectivo que torne qualquer solução que inclua esta variável artificial pior do que qualquer outra solução que não a inclua.

Também é fácil verificar que, desta forma, garantimos que existe sempre uma solução válida para o problema primal restrito.

O quadro do problema primal restrito, para as variáveis artificiais, teria a seguinte configuração:

	A	
T_1	1	=1
T_2	1	=1
…	…	…
T_n	1	=1
M_1	0	<=1
M_2	0	<=1
…	…	…
M_m	0	<=1
C	M	

Figura 7.2 – Matriz das variáveis artificiais

Para esta outra forma, o primeiro problema primal restrito, baseado no modelo de partição de conjuntos, seria então,

$$\text{ISPA - II}:$$

$$\min \sum_{k\in M}\sum_{p\in P^k} c_p^k y_p^k + MA, \tag{7.5}$$

$$\text{sujeito a}:$$

$$\sum_{k\in M}\sum_{p\in P^k} a_{jp}^k y_p^k + A = 1, \forall j \in N \tag{7.6}$$

$$\sum_{p\in P^k} y_p^k \le 1, \forall k \in M \tag{7.7}$$

$$y_p^k \in \{0,1\}, \forall p \in P^k, \forall k \in M \tag{7.8}$$

onde a variável A é uma variável artificial e M é uma constante de valor muito elevado em comparação com os custos unitários das outras sequências de afectação.

A desvantagem desta forma relativamente à primeira é que, quando a solução for impossível (o que não é o caso no nosso problema já que a solução é sempre possível), a primeira forma permite saber quais os trabalhos que inviabilizam uma solução válida para o problema original (através do respectivo valor das variáveis artificiais), enquanto que a última apenas nos informa de que a solução para o problema original é impossível.

Se bem que a implementação dos dois modelos apresentados seja simples, os resultados computacionais preliminares mostram que a sua utilização provoca uma grande instabilidade inicial no processo de geração de colunas, o que se reflecte negativamente no desempenho do algoritmo. Uma possível justificação para este comportamento é o facto de estes modelos não incorporarem qualquer informação sobre a estrutura de uma possível solução óptima para o problema, produzindo valores para as variáveis duais que não direccionam o algoritmo de geração de colunas de uma forma eficiente. Esta situação justifica a determinação de uma solução inicial através de uma heurística, que é apresentada na subsecção seguinte.

7.1.2.2. Solução Inicial Heurística

O objectivo a que nos propusemos foi o de desenvolver uma heurística que, tendo em conta a especificidade do problema, fosse capaz de produzir soluções iniciais admissíveis de qualidade razoável em tempo computacional considerado irrelevante (inferior a 1 segundo). Este tipo de solução pareceu-nos a mais indicada tendo em conta que, por um lado, o principal objectivo deste trabalho é o estudo da aplicação de um algoritmo de solução exacta a um problema complexo de programação de máquinas paralelas e, por outro lado, estávamos interessados em estudar o comportamento do algoritmo de geração de colunas nas mais variadas situações incluindo aquelas em que a solução inicial é admissível mas se encontra claramente afastada da solução óptima.

A heurística desenvolvida é baseada no princípio de dominância para o problema $1||\sum w_j T_j$ apresentado na Secção 4.2: se existirem dois trabalhos i e j com $d_i \leq d_j$, $p_i \leq p_j$ e $w_i \geq w_j$, então existe uma sequência óptima em que o trabalho i aparece antes do trabalho

j. Se bem que este princípio de dominância não se aplique ao problema em estudo, é esperado que a sua utilização produza boas soluções.

O princípio de funcionamento desta heurística é muito simples:

1. Construir uma lista ordenada dos trabalhos por ordem crescente do rácio d_j/w_j (os trabalhos com datas de entrega curtas e prioridades elevadas têm um menor valor do rácio);

2. Seleccionar o primeiro trabalho da lista ordenada e encontrar, no conjunto das máquinas, a que representa o menor custo de afectação para o trabalho seleccionado; em caso de empate, seleccionar a máquina que representa o menor tempo de afectação (tempo de preparação + tempo de processamento), isto é, o menor valor de $(p_{jk} + s_{ij})$. Remover o trabalho afectado da lista ordenada e registar a afectação na máquina seleccionada;

3. Se a lista ordenada ainda contém trabalhos reiniciar o processo no passo 2; caso contrário, está encontrada uma solução inicial admissível para o problema e o processo termina.

Um pseudo código para esta heurística é apresentado na Figura 7.3.

X representa o vector dos trabalhos ainda não afectados, C_i^k o tempo de conclusão do último trabalho (*i*) processado na máquina *k*; *l* representa o trabalho seleccionado para ser afectado à máquina *m*. O vector Y^k regista a sequência de afectação na máquina *k*.

Esta heurística é, no pior dos casos, de complexidade $O(n^2+nm)$ já que contém um primeiro ciclo de ordenação dos *n* trabalhos ($n*n$) e um segundo ciclo de afectação dos *n* trabalhos às *m* máquinas ($n*m$).

O Exemplo que se segue ilustra a aplicação desta heurística a uma pequena instância com duas máquinas e quatro trabalhos.

Begin

$X = \{\ \}; Y^k = \{\ \} \forall k \in M$

$while(X \neq N)$

$l = \arg\min_{j \in N \setminus X} \left\{ \frac{d_j}{w_j} \right\};$

$X = X + \{l\};$

$while(X \neq \{\ \})$

$l = X_1;$

$m = \arg\min_{k \in M} \left\{ \left(\max\left(\max(C_i^k + s_{il}, r_l) + p_{lk} \right) - d_l, 0 \right) * w_l \right\};$

$Y^m = Y^m + \{l\}$

$X = X - \{l\}$

End

Figura 7.3 – Pseudo código para a heurística de solução inicial

Exemplo **7.1** – Heurística de solução inicial

Os dados da instância gerada aleatoriamente são os seguintes:

J_j	r_j	d_j	p_{j1}	p_{j2}	w_j
1	35	66	31	51	95
2	13	70	30	31	62
3	15	110	72	49	55
4	55	81	26	60	68

M_k	a_k	l_k
1	0	1
2	45	2

s_{ij}	1	2	3	4
1	-	38	38	31
2	26	-	25	24
3	27	30	-	28
4	30	30	26	-

1ª Iteração

$X = \{\ \}; Y^k = \{\ \} \forall k \in M;$

$X = \{1,2,4,3\};$

$l = X_1 = 1;$

$$c_1^1 = \left(\max\left(\max(C_0^1 + s_{01}, r_1) + p_{11}\right) - d_1, 0\right) * w_1 =$$

$$= \left(\max\left(\max(0 + 0{,}35) + 31\right) - 66{,}0\right) * 95 = \max(66 - 66{,}0) * 95 = 0;$$

$$c_1^2 = \left(\max\left(\max(45 + 26{,}35) + 51\right) - 66{,}0\right) * 95 = \max(122 - 66{,}0) * 95 = 5320;$$

$$m = 1;$$
$$Y^1 = Y^1 + \{1\}; X = X - \{1\};$$

Afectamos o trabalho *l*=1 à máquina *m*=1 com um custo nulo.

2ª Iteração

$$l = X_1 = 2;$$
$$c_2^1 = \left(\max\left(\max(C_1^1 + s_{12}, r_2) + p_{21}\right) - d_2, 0\right) * w_2 =$$

$$c_2^1 = \left(\max\left(\max(66 + 38{,}13) + 30\right) - 70{,}0\right) * 62 = \max(134 - 70{,}0) * 62 = 3968;$$

$$c_2^2 = \left(\max\left(\max(45 + 0{,}13) + 31\right) - 70{,}0\right) * 62 = \max(76 - 70{,}0) * 62 = 372;$$

$$m = 2;$$
$$Y^2 = Y^2 + \{2\}; X = X - \{2\}.$$

Afectamos o trabalho *l*=2 à máquina *m*=2 com um custo=372.

3ª Iteração

$$l = X_1 = 4;$$
$$c_4^1 = \left(\max\left(\max(66 + 31{,}55) + 26\right) - 81{,}0\right) * 68 = \max(123 - 81{,}0) * 68 = 2856;$$

$$c_4^2 = \left(\max\left(\max(76 + 24{,}55) + 60\right) - 81{,}0\right) * 68 = \max(160 - 81{,}0) * 68 = 5372;$$

$$m = 1;$$
$$Y^1 = Y^1 + \{4\}; X = X - \{4\}.$$

Afectamos o trabalho *l*=4 à máquina *m*=1 com um custo=2856.

4ª Iteração

$l = X_1 = 3;$

$c_3^1 = \left(\max\left(\max(123 + 26,15) + 72\right) - 110,0\right) * 55 = \max(221 - 110,0) * 55 = 6105;$

$c_3^2 = \left(\max\left(\max(76 + 25,15) + 49\right) - 110,0\right) * 55 = \max(150 - 110,0) * 55 = 2200;$

$m = 2;$

$Y^2 = Y^2 + \{3\}; X = X - \{3\}.$

Afectamos o trabalho l=3 à máquina m=2 com um custo=2200.

O custo total da solução heurística obtida é c=0+372+2856+2200=5428. A aplicação do método exacto a esta instância permitiu verificar que esta solução é óptima.

As sequências de afectação resultantes da aplicação desta heurística ao Exemplo estão representadas na Figura 7.4.

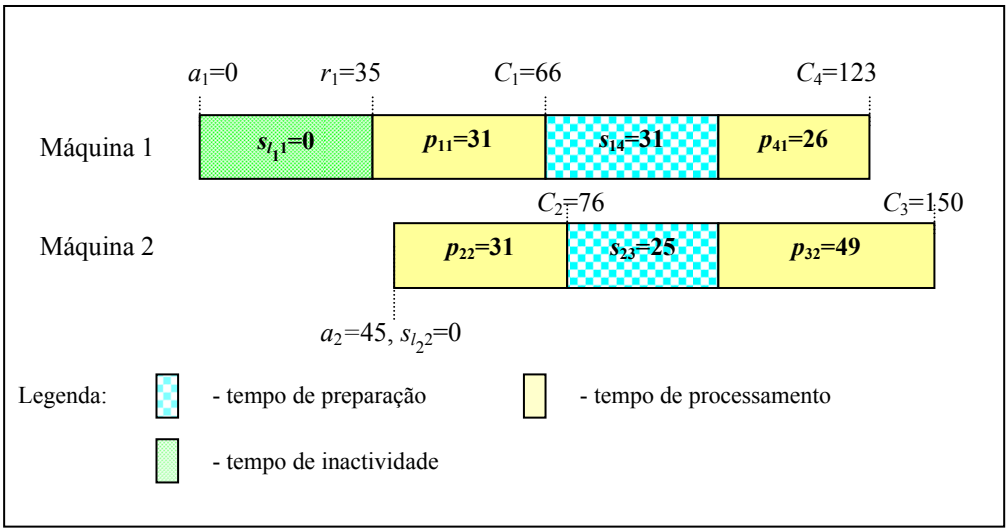


Figura 7.4 – Diagrama de Gantt da solução inicial heurística

7.2. Geração das Instâncias de Teste

Os testes computacionais foram desenhados no sentido de observar o comportamento do modelo desenvolvido em diferentes situações. Assim, para além da situação clássica de estudar o comportamento do modelo para diferentes intervalos dos tempos de processamento e dos tempos de preparação, e diferentes níveis de congestionamento do sistema, foi também estudada a sensibilidade do modelo a diferentes valores dos parâmetros das datas de disponibilidade dos trabalhos.

Para os vários conjuntos de testes efectuados, houve sempre a preocupação de tentar alinhar os valores dos parâmetros usados para gerar as instâncias com o que é mais comum encontrar na literatura para problemas de programação de máquinas paralelas. Em particular, é frequentemente referido na literatura (ver por exemplo, Van den Akker, Hoogeveen e Van de Velde, 1999, Chen e Powell, 1999b, e Chen e Lee, 2002) que, para um mesmo número de máquinas e trabalhos, o tempo total de processamento dos trabalhos mais o tempo total de preparação das máquinas, têm uma influência com significado no desempenho do algoritmo; maiores valores destes tempos penalizam o desempenho do algoritmo devido a aumentarem o horizonte temporal de programação.

Por este motivo, as instâncias para o nosso principal conjunto de testes foram geradas para o caso mais desfavorável (esta questão vai ser analisada com mais detalhe na Subsecção 7.4.1) em que os tempos de processamento dos trabalhos e os tempos de preparação das máquinas foram definidos de forma a concretizar um horizonte temporal de programação equivalente ao valor mais desfavorável que é mais frequentemente referido na literatura (entre outros, Van den Akker, Hoogeveen e Van de Velde, 1999, Chen e Powell, 1999a, 1999b, e Chen e Lee, 2002).

As instâncias para o conjunto de testes com horizonte temporal de programação mais longo foram geradas da seguinte forma:

- Número de trabalhos $n \in \{20,30,40,50,60,70,80,90,100,120, 150,200,220\}$.
- Número de máquinas $m \in \{2,4,6,8,10,20,30,50\}$.

- Pesos dos trabalhos w_j=U[1,100].
- Tempos de processamento p_{jk}=U[10,80].
- Datas de disponibilidade das máquinas a_k=U[0,50].
- Configuração inicial das máquinas l_k=U[1,n].
- Datas de disponibilidade dos trabalhos r_j=U[0,100].
- Tempos de preparação dependentes da sequência s_{ij}=U[20,40].
- Níveis de congestionamento do sistema $q \in \{1,2,3,4,5\}$.
- Datas de entrega d_j (adaptado de Ho e Chang, 1995):

  $d_j = \max\{\alpha_j, \beta_q\}$ tal que

  $\alpha_j = \min_{k \in M}\{p_{jk} + \max(a_k, r_j) + \max_{i \in N}(s_{ji})\}$

  β_q=U[1,160r/q], $r=n/m$

De notar ainda que a distribuição dos tempos de preparação dependentes da sequência foi escolhida de forma a satisfazer o princípio das desigualdades triangulares, $s_{ij}+s_{jv} \geq s_{iv}$, i.e., $\min\{s_{ij}\}+\min\{s_{jv}\} \geq \max\{s_{iv}\}$ (por exemplo, para o primeiro conjunto de testes, 20+20≥40). O valor de q indica o nível de congestionamento do sistema a programar. Quanto maior o seu valor, mais congestionado estará o sistema e mais trabalhos sofrerão atrasos relativamente às suas datas de entrega. Foram testados cinco níveis de congestionamento (q=1,2,3,4,5).

A Figura 7.5 ilustra a relação entre o nível de congestionamento e o número de trabalhos com atraso obtida nos nossos testes computacionais.

Para além da já referida preocupação de alinhar os valores escolhidos para cada um dos parâmetros com o que é frequente encontrar na literatura, existiram outras preocupações de carácter mais estratégico que a seguir se descrevem.

Os valores relativos dos tempos de processamento e dos tempos de preparação dependentes da sequência, foram escolhidos de forma a englobarem três situações: tempos de processamento menores, iguais e maiores que os tempos de preparação.

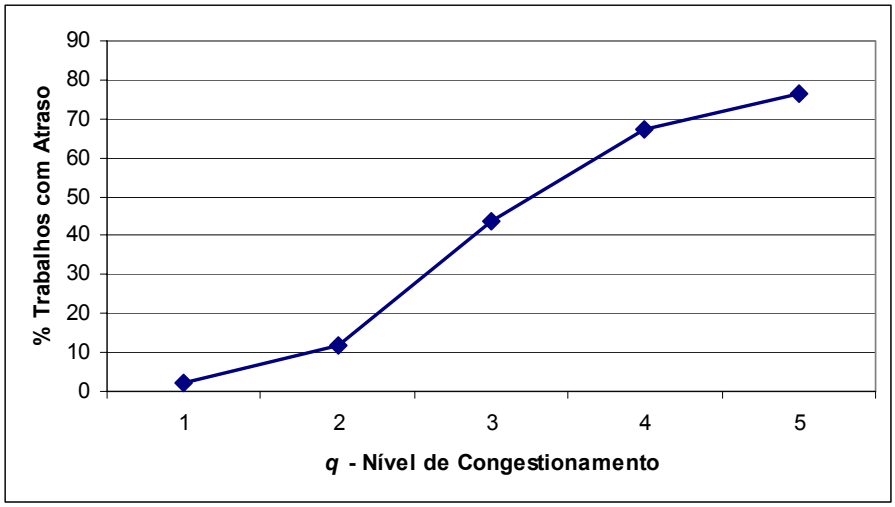


Figura 7.5 – Relação entre o nível de congestionamento e a % de trabalhos com atraso

Os valores das datas de disponibilidade dos trabalhos foram definidos de forma a reflectirem a situação de estes estarem disponíveis para processamento em t=0 ou ficarem disponíveis a curto prazo ($t \leq 100$).

Por último, os valores das datas de disponibilidade das máquinas foram definidos de forma a reflectirem a situação de as máquinas já estarem disponíveis para processamento em t=0 ou estarem a processar o último trabalho antes de iniciarem um novo programa ($t \leq 50$).

7.3. Resultados dos Testes Computacionais

Para cada nível de congestionamento, foi seleccionado um par de valores de número de máquinas (m) e número de trabalhos (n), e foram geradas aleatoriamente 10 instâncias de teste a partir das distribuições associadas aos seus parâmetros, num total de 28 pares (m,n) diferentes e 1250 instâncias de teste.

Os resultados dos testes computacionais efectuados são apresentados em tabelas, sendo que a cada tabela corresponde um único nível de congestionamento. Os dados apresentados nas colunas das tabelas são os seguintes:

- ***q*** - nível de congestionamento;
- ***m*** - número de máquinas;
- ***n*** - número de trabalhos;
- **LP-IP Gap** - a diferença percentual entre o valor da solução óptima da relaxação linear (LP) e o valor da solução óptima inteira (IP);
- **RL** - a percentagem de problemas cuja solução óptima inteira foi encontrada na fase de relaxação linear;
- **Colunas Geradas** - o número de colunas geradas na fase de relaxação linear (RL) e na fase de partição e geração de colunas (B&P) para obter a solução óptima inteira;
- **Tempo de CPU** - o tempo computacional (em segundos) gasto na fase RL e na fase B&P para obter a solução óptima inteira;
- **Nodos Avaliados** - número de nodos avaliados na árvore de pesquisa para obter a solução óptima inteira.

Os resultados computacionais para o primeiro conjunto de testes são apresentados nas Tabelas seguintes.

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	1	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Máx	%	Média	Máx	Média	Max	Média	Máx	Média	Max	Média	Máx	Média	Máx
2	20	0,00%	0,00%	40%	47	94	366	503	1	2	3	6	2	8	26	38
2	30	0,00%	0,00%	10%	87	145	821	1004	6	11	22	30	26	36	50	56
2	40	0,00%	0,00%	50%	99	163	1296	1524	14	18	73	93	51	109	74	76
2	50	0,00%	0,00%	0%	128	186	1862	2479	31	56	408	519	439	575	90	96
4	30	0,00%	0,00%	50%	38	93	466	534	1	1	4	6	2	6	41	48
4	40	0,00%	0,00%	0%	108	154	809	961	2	3	18	21	20	24	67	72
4	50	0,00%	0,00%	10%	129	182	1051	1351	6	10	57	157	58	165	90	98
4	60	0,14%	1,47%	0%	140	190	1611	2011	23	29	222	329	245	358	104	114
6	40	0,00%	0,00%	0%	98	130	595	728	2	3	7	8	10	11	59	68
6	50	0,00%	0,00%	0%	113	162	960	1180	5	7	23	26	28	33	84	92
6	60	0,00%	0,00%	0%	137	178	1211	1300	11	14	54	63	65	75	104	114
6	70	0,00%	0,00%	10%	168	229	1476	1886	21	29	175	228	178	253	117	228
8	40	0,00%	0,00%	0%	107	137	449	543	1	2	4	5	5	6	49	54
8	50	0,00%	0,00%	0%	107	162	702	841	3	5	11	14	16	17	76	88
8	60	0,00%	0,00%	10%	136	225	941	1049	6	8	28	34	32	42	96	108
8	70	0,00%	0,00%	0%	168	216	1179	1354	13	16	87	95	101	111	104	122
8	80	0,70%	7,00%	0%	202	277	1512	2031	25	29	189	221	215	245	130	148
10	50	0,00%	0,00%	0%	92	168	575	664	1	3	7	9	8	10	70	82
10	70	0,00%	0,00%	0%	181	277	1108	1281	10	12	45	49	55	59	118	126
10	90	0,00%	0,00%	0%	186	432	1732	2162	22	31	153	174	175	213	162	170
10	100	0,09%	0.94%	0%	234	385	1768	2352	43	51	350	426	394	477	169	183
20	100	0,00%	0,00%	0%	257	374	1265	1443	15	18	71	82	86	101	160	172
20	120	0,00%	0,00%	0%	274	351	1666	1928	33	41	184	216	218	256	206	220
20	150	0,00%	0,00%	0%	407	605	2254	2696	104	146	707	950	811	1096	248	266
30	150	0,00%	0,00%	0%	686	992	1970	2091	55	63	328	378	384	434	245	266
50	150	0,00%	0,00%	0%	372	594	1330	1501	14	23	95	117	110	129	205	228
50	180	0,00%	0,00%	0%	436	579	2395	2832	45	56	370	460	415	516	235	264
50	220	0,00%	0,00%	0%	550	684	2954	3884	147	213	1106	1305	1254	1496	316	335

Tabela 7.1 – Resultados computacionais para q=1

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	2	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	20	0,20%	2,40%	30%	150	207	323	441	4	6	4	6	7	9	12	30
2	30	1,10%	11,00%	0%	207	407	1307	1681	16	35	42	64	57	81	32	54
2	40	1,00%	1,00%	10%	337	604	2230	3696	54	109	172	337	208	436	45	74
2	50	2,30%	19,30%	0%	481	1045	2408	8363	142	379	1600	4531	1743	4781	59	83
4	30	0,00%	0,00%	0%	173	231	382	491	3	5	4	5	7	9	16	34
4	40	0,60%	3,30%	0%	207	301	1350	2585	8	14	29	62	38	65	43	82
4	50	0,00%	0,00%	0%	205	293	1659	1934	17	25	58	68	76	89	72	80
4	60	0,01%	0,17%	0%	309	481	2706	3830	44	82	295	476	339	558	77	100
6	40	0,40%	4,00%	10%	216	329	587	855	4	9	7	10	11	16	25	42
6	50	0,30%	3,00%	0%	274	445	1195	1383	11	18	25	34	36	49	46	64
6	60	0,20%	1,80%	0%	298	378	1858	2267	18	26	64	82	83	105	73	102
6	70	0,38%	2,00%	0%	343	445	2698	3622	40	66	220	301	261	367	83	112
8	40	1,00%	7,00%	0%	221	286	275	449	2	4	2	3	5	6	14	28
8	50	0,40%	4,00%	0%	258	312	768	889	6	10	11	17	17	26	35	52
8	60	0,40%	3,10%	0%	296	375	1349	1707	11	17	29	37	41	50	61	92
8	70	0,05%	0,53%	0%	333	500	1960	2521	20	26	98	117	118	135	72	88
8	80	0,07%	0,70%	0%	377	534	2473	3062	39	64	186	220	226	268	90	114
10	50	0,00%	0,00%	0%	263	347	489	689	4	5	5	6	9	11	30	50
10	70	0,70%	2,50%	0%	346	481	1531	1731	14	20	40	51	55	69	67	84
10	90	1,38%	13,80%	0%	397	513	2559	3182	39	73	135	164	172	214	127	138
10	100	0,04%	0,26%	0%	476	715	3259	4700	82	128	388	505	470	607	117	158
20	100	0,40%	3,00%	0%	479	576	1248	1500	15	22	33	42	50	60	77	102
20	120	0,10%	1,00%	0%	548	706	2306	2801	30	43	122	141	153	175	135	168
20	150	0,55%	2,60%	0%	705	816	2499	5143	115	193	600	726	716	801	174	223
30	150	0,45%	1,70%	0%	888	942	2207	2617	42	46	141	173	184	215	149	192
50	150	0,10%	1,60%	0%	803	892	481	724	13	19	16	22	30	40	48	72
50	180	0,47%	2,12%	0%	824	1013	1258	1738	24	34	67	130	92	164	97	137
50	220	0,15%	0,82%	0%	1114	1217	2390	2739	71	84	264	322	336	389	128	179

Tabela 7.2 – Resultados computacionais para q=2

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	3	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	20	1,40%	13,60%	80%	219	261	433	638	4	7	10	15	6	18	16	26
2	30	1,50%	6,80%	10%	421	511	114	237	32	43	114	237	135	268	15	36
2	40	1,10%	2,80%	0%	710	1091	1500	3833	144	235	380	1034	524	1177	15	48
4	30	0,40%	1,20%	40%	302	376	156	370	6	8	4	11	8	18	4	8
4	40	1,20%	3,50%	10%	495	546	495	1326	28	37	41	108	66	130	12	40
4	50	2,80%	6,00%	10%	689	757	1492	2019	81	111	278	364	331	447	24	32
6	40	1,20%	8,90%	50%	402	446	247	603	9	12	11	24	15	35	10	30
6	50	1,50%	5,00%	10%	579	624	478	624	32	46	54	105	81	134	12	28
6	60	1,80%	4,40%	0%	761	874	1046	2220	80	99	192	375	272	452	25	60
6	70	2,48%	8,95%	0%	928	1045	1671	3239	321	383	889	2333	1210	2658	28	76
8	40	1,00%	4,00%	30%	353	404	110	256	4	6	3	7	6	11	5	14
8	50	1,90%	3,50%	20%	472	527	458	911	13	17	35	76	41	89	25	52
8	60	1,80%	9,50%	0%	663	760	496	1870	38	49	72	343	110	380	17	102
8	70	1,79%	4,44%	0%	761	959	1102	2961	122	147	390	1381	512	1510	38	192
8	80	2,10%	3,70%	0%	958	1085	1914	4091	274	341	961	2711	1236	2975	47	124
10	50	1,20%	4,00%	20%	446	531	227	605	6	8	9	30	14	37	12	44
10	70	1,70%	3,50%	0%	759	806	595	1051	44	50	96	210	140	253	18	42
10	90	1,18%	2,50%	0%	1023	1116	1492	2242	145	196	418	1001	563	1197	29	52
20	100	1,00%	2,00%	0%	978	1031	598	1140	42	51	116	364	158	407	29	90
20	120	1,80%	2,80%	0%	1181	1279	1827	2987	100	132	537	1465	637	1586	84	168
30	150	1,84%	3,13%	0%	1490	1573	1389	2631	117	142	651	1909	768	2038	108	316
50	150	1,70%	2,70%	0%	1358	1406	1075	1518	37	40	435	1030	473	1070	378	1018
50	180	2,35%	4,17%	0%	1424	1515	1558	2871	79	96	1822	5900	1901	5976	484	1696

Tabela 7.3 – Resultados computacionais para q=3

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	4	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	20	0,03%	0,27%	80%	205	236	97	132	4	6	2	4	4	8	2	2
2	30	0,40%	2,00%	50%	428	488	512	1058	34	42	46	92	57	127	4	10
2	40	0,50%	1,10%	10%	718	894	1930	3618	158	203	499	953	608	1114	16	36
4	30	0,10%	0,60%	60%	312	369	91	106	7	8	2	4	8	12	2	4
4	40	0,20%	0,70%	50%	560	704	381	877	37	53	36	85	55	118	8	22
4	50	0,50%	2,00%	0%	763	821	655	2255	120	134	156	596	277	722	9	38
6	40	0,09%	0,30%	40%	430	476	101	165	11	13	5	9	14	20	2	4
6	50	0,20%	0,50%	20%	606	676	283	447	40	50	32	52	66	95	5	10
6	60	0,90%	2,40%	10%	807	964	952	2203	119	161	253	658	347	788	24	60
6	70	0,69%	1,53%	0%	1062	1420	892	1991	535	738	783	1963	1319	2428	15	50
8	40	0,10%	1,00%	60%	368	394	171	402	4	5	5	14	6	18	8	24
8	50	0,40%	1,00%	10%	522	635	246	811	16	18	19	68	34	84	9	38
8	60	0,30%	1,00%	40%	708	838	463	831	55	66	84	170	106	224	12	26
8	70	0,62%	1,36%	0%	844	943	658	1362	204	230	324	837	528	1059	16	46
8	80	1,41%	2,69%	0%	1020	1096	1865	3314	503	616	2011	3882	2515	4482	50	96
10	50	0,70%	1,40%	10%	456	514	220	477	7	9	11	26	17	33	12	32
10	70	0,60%	1,20%	0%	788	874	483	1045	67	78	114	282	181	360	13	34
10	90	1,90%	3,50%	0%	1113	1256	2623	3827	308	422	2399	4636	2708	4979	98	212
20	100	1,00%	1,50%	0%	1034	1086	1015	1419	58	67	348	637	407	703	72	116
20	120	1,48%	2,08%	0%	1251	1435	1875	3597	285	340	2681	6276	2967	6616	152	382
30	150	1,52%	2,28%	0%	1589	1657	2365	4159	196	227	3686	9060	3883	9254	325	932
50	150	0,60%	1,30%	0%	1403	1431	810	1875	40	45	302	1093	343	1134	263	1040
50	180	1,21%	1,62%	0%	1512	1674	1961	2802	101	129	3674	8379	3776	8481	841	1936

Tabela 7.4 – Resultados computacionais para *q*=4

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	5	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	20	0,10%	0,70%	80%	207	241	145	204	4	7	4	6	5	10	3	4
2	30	0,20%	1,00%	40%	455	566	455	842	37	47	45	84	64	117	3	8
2	40	0,10%	0,40%	40%	715	791	857	2283	158	183	213	507	286	638	6	22
4	30	0,01%	0,10%	70%	318	375	85	114	7	8	3	4	7	10	2	4
4	40	0,10%	0,60%	30%	545	696	282	419	37	55	27	47	56	81	5	12
4	50	0,30%	0,80%	10%	790	1148	623	1450	131	209	149	371	266	489	7	24
6	40	0,10%	0,50%	50%	429	471	161	238	11	15	8	12	15	27	4	8
6	50	0,20%	0,80%	30%	635	713	407	1034	44	52	407	1034	44	52	8	18
6	60	0,50%	1,80%	0%	806	923	889	2378	125	145	264	799	389	919	20	78
6	70	0,40%	0,87%	20%	1140	1467	1043	2263	620	816	938	2367	1371	2891	16	46
8	40	0,20%	0,80%	50%	371	383	160	295	4	5	4	8	6	13	8	16
8	50	0,20%	0,80%	20%	546	617	190	426	17	22	14	35	29	52	5	16
8	60	0,20%	0,95%	30%	718	848	423	832	57	68	78	171	112	232	11	26
8	70	0,39%	1,00%	10%	878	1021	595	1625	227	283	310	948	506	1161	13	48
8	80	0,46%	1,11%	0%	1106	1267	862	1426	606	752	889	1690	1496	2274	14	38
10	50	0,20%	0,70%	40%	478	536	235	428	8	10	10	19	14	27	10	20
10	70	0,50%	0,90%	0%	811	885	453	935	70	78	106	278	176	353	12	40
10	90	0,40%	1,10%	0%	1188	1264	1177	2128	378	450	953	2075	1332	2466	26	66
20	100	0,40%	1,00%	0%	1052	1111	548	1342	57	70	152	416	210	469	30	110
20	120	0,72%	1,14%	0%	1267	1350	1150	2058	315	368	1277	3143	1592	3456	50	154
30	150	0,85%	1,41%	0%	1634	1739	1840	2958	208	225	2883	6909	3092	7123	218	522
50	150	0,40%	1,00%	0%	1428	1477	737	1482	40	45	231	749	271	791	191	676
50	180	0,67%	1,09%	0%	1535	1667	1527	2351	102	130	1844	4448	1947	4562	366	1018

Tabela 7.5 – Resultados computacionais para q=5

A partir dos resultados apresentados (Tabela 7.1 a Tabela 7.5), podemos tecer as seguintes considerações:

- A eficiência do método decresce à medida que o número de trabalhos aumenta (para um número de máquinas fixo) (Figura 7.6) e cresce com o aumento do número de máquinas (para um número de trabalhos fixo) (Figura 7.7).

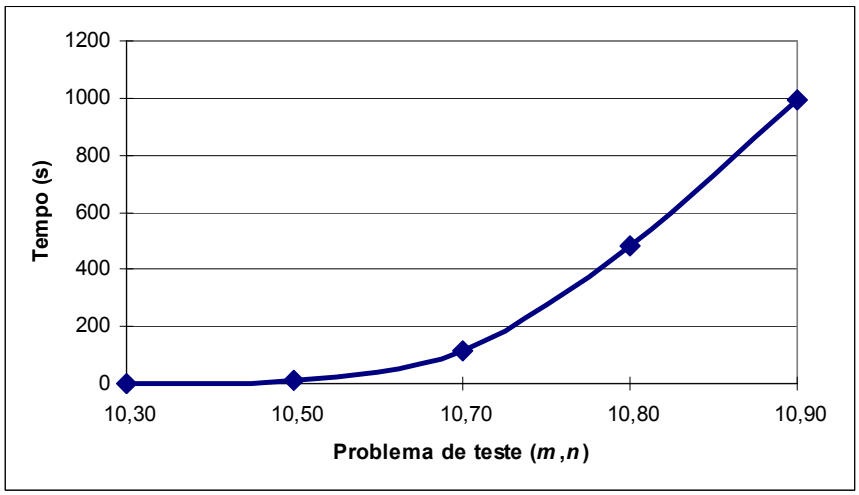


Figura 7.6 – Eficiência do método em função do número de trabalhos (n)

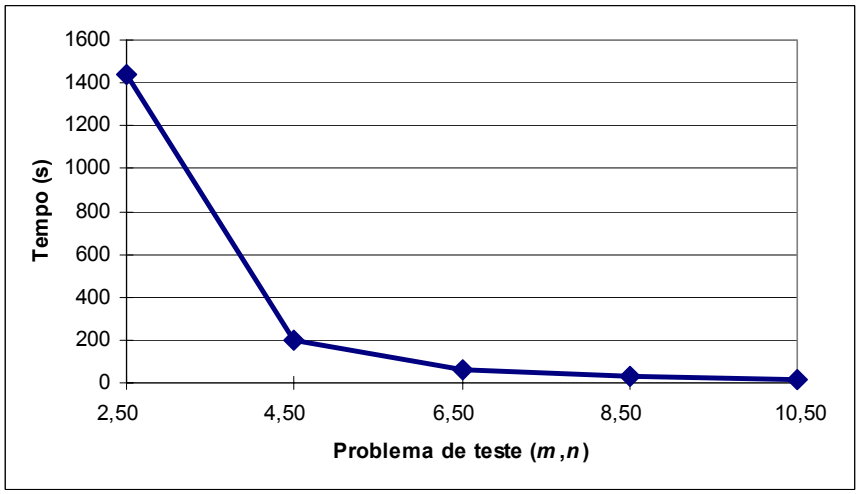


Figura 7.7 – Eficiência do método em função do número de máquinas (m).

- Se bem que o rácio entre o número de trabalhos e o número de máquinas (n/m) seja um factor importante para a eficiência do método, verifica-se que esta depende também da dimensão do problema. A Figura 7.8 mostra que, para um rácio constante (n/m=5), a eficiência do método decresce com a dimensão do problema.

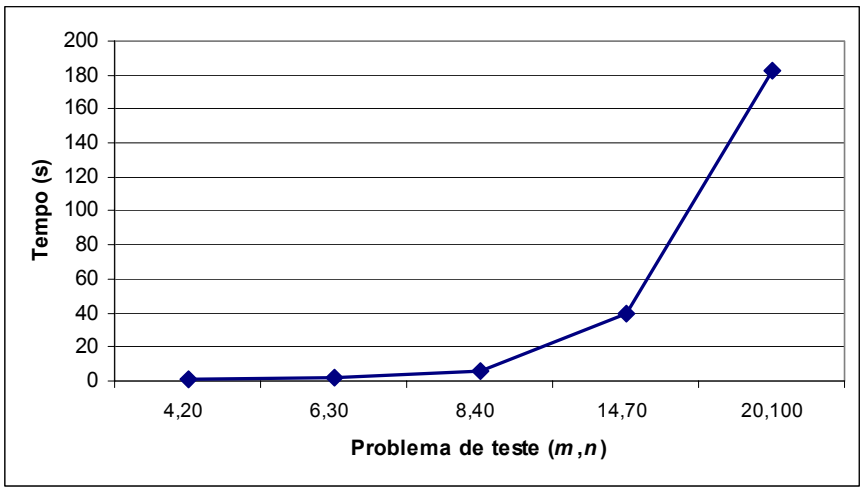


Figura 7.8 – Eficiência do método em função da dimensão do problema (m,n).

- A eficiência do método é maior para os níveis de congestionamento mais baixos (Figura 7.9).

- O valor da solução fornecida pela resolução do problema de relaxação linear encontra-se muito próximo do valor da solução óptima do problema inteiro constituindo um excelente limite inferior (o desvio médio global, "LP-IP gap", é de 0,61%). Acresce o facto da resolução da relaxação linear do problema de programação inteira apresentar tempos computacionais muito interessantes, em que a média global é de 80 segundos e o valor máximo é de 816 segundos. Estes dois factos permitem que esta aproximação possa ser encarada como uma boa plataforma para determinar limites para aferir a qualidade de soluções heurísticas.

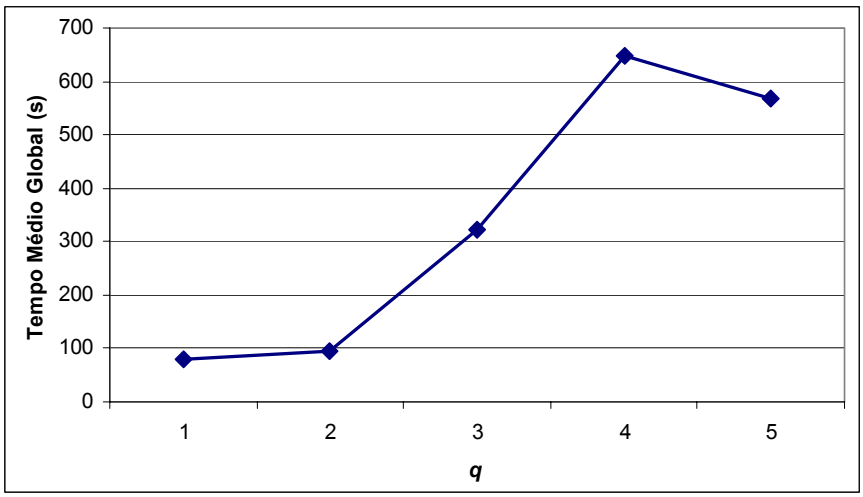


Figura 7.9 – Eficiência do método em função do nível de congestionamento do sistema.

- Uma análise à repartição de tempos entre as duas fases do algoritmo (relaxação linear e partição e geração de colunas) indica que uma parte muito substancial do tempo computacional (> 80%) é gasto na fase de partição e geração de colunas. O mesmo tipo de análise agora feito sobre a repartição de tempos entre o algoritmo simplex e a resolução dos subproblemas indica que uma parte muito substancial do tempo computacional (> 90%) é gasto na resolução dos subproblemas. Estas são indicações claras de direcções de investigação para a melhoria do método.

- O nosso algoritmo mostrou-se capaz de resolver problemas de dimensão significativa em tempo computacional considerado razoável; (m,n)={(10,90), (20,120),(50,180)} e média global de 620 segundos para os níveis médio e alto de congestionamento do sistema (q=3,4,5); (m,n)={(10,100), (20,150),(50,220)} e média global de 195 segundos para os níveis de baixo congestionamento do sistema (q=1,2).

7.4. Análise de Sensibilidade

Nesta Secção é apresentado a análise de sensibilidade do modelo desenvolvido a diferentes valores dos principais parâmetros do problema. Em primeiro lugar, é analisado o comportamento do modelo para a situação de um horizonte temporal de programação mais curto. Seguem-se análises semelhantes para diferentes valores das datas de disponibilidade dos trabalhos. O processo de geração das instâncias de testes segue o esquema descrito na Secção 7.2, sendo que, do conjunto das combinações (m,n) possíveis, vamos testar apenas um seu subconjunto de 6 combinações (m,n) diferentes e 300 instâncias de teste, que consideramos significativo e suficiente para atingir os objectivos pretendidos.

7.4.1. Horizonte Temporal Curto

A adopção de diferentes valores para o horizonte temporal de programação, traduzido pela soma dos tempos de processamento dos trabalhos mais a soma dos tempos de preparação das máquinas, pode ter um efeito significativo no desempenho dos algoritmos de solução. Daí que seja comum proceder-se à análise do comportamento dos algoritmos desenvolvidos para diferentes valores dos referidos tempos. Van den Akker, Hoogeveen e Van de Velde (1999), Chen e Powell (1999b) e Chen e Lee (2002), entre outros, realizaram esta análise e reportaram uma deterioração significativa dos tempos computacionais com o aumento dos tempos de processamento dos trabalhos.

Nesta Subsecção, propomo-nos efectuar o mesmo tipo de análise. Como o principal conjunto de testes foi realizado com o horizonte temporal de programação mais longo, que passaremos a designar por horizonte temporal longo (HTlongo), a adopção agora de um horizonte temporal mais curto permite concretizar o objectivo pretendido.

O intervalo dos tempos de processamento foi reduzido de p_{jk}=U[10,80] para p_{jk}=U[10,50], o intervalo dos tempos de preparação dependentes da sequência foi reduzido de s_{ij}=U[20,40] para s_{ij}=U[10,20], e o intervalo das datas de disponibilidade dos trabalhos foi reduzido de r_j=U[0,100] para r_j=U[0,50].

Para cada par de valores (m,n) foram geradas 10 instâncias de teste que apresentam, relativamente à Secção 7.2, as seguintes diferenças nos valores dos respectivos parâmetros:

- Tempos de processamento p_{jk}=U[10,50]
- Datas de disponibilidade dos trabalhos r_j=U[0,50]
- Tempos de preparação dependentes da sequência s_{ij}=U[10,20]
- Datas de entrega d_j (adaptado de Ho e Chang, 1995):

  $d_j = \max\{\alpha_j, \beta_q\}$ tal que

  $\alpha_j = \min_{k \in M}\{p_{jk} + \max(a_k, r_j) + \max_{i \in N}(s_{ji})\}$

  β_q=U[1,110r/q], $r=n/m$

Os resultados computacionais obtidos são apresentados na Tabelas seguintes.

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	1	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	n	Média	Máx	%	Média	Máx	Média	Max	Média	Máx	Média	Max	Média	Máx	Média	Máx
2	40	0,00%	0,00%	30%	70	143	857	1311	7	13	52	78	43	89	69	78
6	60	0,00%	0,00%	0%	128	173	965	1027	6	7	45	50	51	57	103	110
10	70	0,00%	0,00%	20%	152	214	916	1035	5	7	36	43	34	49	110	122
10	90	0,00%	0,00%	0%	424	592	1732	2162	38	60	153	174	191	219	162	170
20	100	0,00%	0,00%	0%	419	532	1265	1443	17	23	71	82	89	104	160	172
50	150	0,00%	0,00%	0%	389	541	1726	2286	13	17	142	231	155	248	177	219

Tabela 7.6 – Resultados computacionais para o horizonte temporal curto e q=1

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	2	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,00%	0,00%	0%	217	411	1901	3088	18	37	119	259	138	296	54	68
6	60	0,20%	1,90%	0%	262	337	1581	2022	10	15	58	72	68	82	69	92
10	70	0,20%	1,40%	0%	300	373	1281	1787	7	10	36	43	43	53	72	90
10	90	1,38%	13,80%	0%	506	657	2559	3182	27	37	135	164	162	198	127	138
20	100	0,40%	3,00%	0%	588	681	1248	1500	14	18	33	42	48	56	77	102
50	150	0,02%	0,10%	10%	810	967	345	553	15	23	9	16	23	34	34	56

Tabela 7.7 – Resultados computacionais para o horizonte temporal curto e q=2

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	3	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	2,20%	5,80%	10%	693	840	1513	2864	114	140	279	518	393	617	13	36
6	60	0,80%	4,20%	0%	611	800	842	1547	51	79	74	133	125	212	20	54
10	70	0,90%	2,90%	0%	643	694	713	1317	33	38	52	148	86	183	24	62
10	90	0,50%	1,07%	0%	773	983	2507	4762	90	129	339	1029	430	1152	70	202
20	120	0,62%	1,81%	0%	1022	1116	1074	2257	88	105	192	687	280	766	87	580
50	150	0,60%	1,10%	10%	1221	1278	679	1108	37	42	303	730	310	769	144	372

Tabela 7.8 – Resultados computacionais para o horizonte temporal curto e q=3

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	4	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,60%	1,90%	10%	746	1004	1510	2376	151	191	332	527	450	683	10	22
6	60	0,80%	2,70%	0%	796	879	685	1643	110	137	149	339	260	427	13	40
10	70	0,60%	1,50%	10%	779	915	630	1182	70	81	142	315	198	389	22	56
10	90	1,41%	3,18%	0%	1046	1094	1818	2729	274	338	1289	2276	1564	2576	59	106
20	120	0,89%	1,53%	0%	1239	1349	1450	2825	219	251	1393	3345	1612	3574	91	248
50	150	0,20%	0,30%	0%	1293	1356	495	873	43	49	181	499	225	542	74	252

Tabela 7.9 – Resultados computacionais para o horizonte temporal curto e q=4

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	5	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,20%	0,68%	20%	752	1450	1179	2675	151	315	266	554	365	670	7	16
6	60	0,20%	0,70%	10%	844	962	449	1030	123	144	109	311	222	427	7	24
10	70	0,20%	0,40%	10%	813	1049	404	699	75	108	83	127	150	190	10	20
10	90	0,56%	1,75%	0%	1139	1195	1073	2041	348	425	874	1940	1222	2272	27	62
20	120	0,60%	1,02%	0%	1313	1411	1387	1411	257	309	1469	3363	1726	3617	82	216
50	150	0,20%	0,38%	10%	1321	1404	594	1006	44	51	246	585	266	631	104	282

Tabela 7.10 – Resultados computacionais para o horizonte temporal curto e q=5

A análise comparativa dos resultados obtidos para o mesmo conjunto de pares (m,n), apresentada na Tabela 7.11, permite-nos confirmar uma degradação dos tempos computacionais com o aumento do horizonte temporal de programação. Contudo, esta degradação dos tempos computacionais é relativamente modesta (aproximadamente mais 39%) comparativamente à variação do horizonte temporal de programação (aproximadamente mais 67%).

	LP-IP	RL	Colunas Geradas		Tempo de CPU (s)			Nodos
	Gap		RL	B&P	RL	B&P	Total	Avaliados
	Média	%	Média	Média	Média	Média	Média	Média
HTcurto	0,48%	5,0%	710	1179	82	289	364	69
HTlongo	0,71%	4,0%	736	1359	96	417	506	93

Tabela 7.11 – Resultados comparativos - horizonte temporal curto e longo

7.4.2. Datas de Disponibilidade dos Trabalhos

Nesta Subsecção, vamos apresentar um estudo da sensibilidade do modelo desenvolvido a variações nas datas de disponibilidade dos trabalhos. Para isso, vamos estudar duas situações diferentes; a primeira situação envolve tempos de disponibilidade dos

trabalhos iguais a zero para todos os trabalhos (r_j=0); a segunda situação envolve tempos de disponibilidade dos trabalhos gerados de uma forma mista com um subconjuntos dos trabalhos a apresentar datas de disponibilidade iguais a zero e os restantes trabalhos a apresentarem datas de disponibilidade distribuídas uniformemente ao longo do horizonte de programação.

7.4.2.1. r_j=0

Para cada par de valores (m,n) foram geradas 10 instâncias de teste seguindo o mesmo esquema apresentado na Secção 7.2 com a excepção das datas de disponibilidade dos trabalhos que tomam o valor nulo (r_j=0) para todos os trabalhos.

Os resultados computacionais obtidos são apresentados na Tabelas seguintes.

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	1	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Máx	%	Média	Máx	Média	Max	Média	Máx	Média	Max	Média	Máx	Média	Máx
2	40	0,00%	0,00%	0%	93	144	873	1460	14	19	80	143	95	160	70	76
6	60	0,00%	0,00%	0%	124	165	889	1069	11	13	63	70	74	82	100	106
10	70	0,00%	0,00%	20%	175	257	942	1152	12	16	51	64	53	76	107	122
10	90	0,00%	0,00%	10%	179	359	1328	1792	24	34	177	231	184	265	141	150
20	120	0,00%	0,00%	0%	269	323	1648	1992	36	47	217	271	253	318	194	208
50	150	0,00%	0,00%	0%	393	507	1786	2299	18	27	160	275	179	298	175	216

Tabela 7.12 – Resultados computacionais para r_j=0 e q=1

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	2	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,01%	0,12%	0%	317	567	2155	3908	52	109	299	596	351	705	45	66
6	60	1,20%	6,20%	0%	284	450	1517	1888	19	32	85	106	104	130	63	93
10	70	0,28%	2,30%	0%	418	574	1363	1748	20	30	46	60	67	89	63	78
10	90	0,08%	0,87%	0%	392	559	2218	3205	41	73	210	265	252	338	105	127
20	120	0,03%	0,31%	0%	540	671	2036	2295	38	49	129	159	167	196	105	132
50	150	0,08%	0,55%	10%	771	853	492	752	16	21	13	24	28	43	45	69

Tabela 7.13 – Resultados computacionais para r_j=0 e q=2

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	3	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,90%	2,40%	40%	686	899	1617	2207	241	348	648	899	630	1113	15	22
6	60	1,10%	2,20%	10%	690	794	980	3383	122	151	282	1032	376	1096	27	152
10	70	1,60%	3,60%	0%	701	780	719	1303	69	82	164	364	234	342	24	52
10	90	1,50%	5,80%	0%	987	1081	1806	3426	271	344	760	2297	1031	2641	41	108
20	120	1,00%	2,14%	0%	1069	1188	1078	1894	158	202	350	950	508	1152	42	86
50	150	0,54%	1,07%	0%	1166	1230	602	1082	35	43	248	686	284	724	129	402

Tabela 7.14 – Resultados computacionais para r_j=0 e q=3

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	4	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,50%	1,60%	30%	770	1187	1967	4454	292	478	876	2115	906	2312	17	52
6	60	0,70%	2,40%	20%	743	917	695	1641	191	240	289	723	422	910	16	44
10	70	0,70%	1,50%	10%	745	839	525	878	103	124	170	279	257	380	15	32
10	90	1,00%	1,60%	0%	1074	1222	1536	3076	528	772	1846	4265	2374	4821	47	114
20	120	1,43%	2,00%	0%	1243	1295	2195	3544	344	402	4196	10427	4541	10756	244	742
50	150	0,20%	0,38%	0%	1263	1326	292	535	42	53	65	149	108	185	23	66

Tabela 7.15 – Resultados computacionais para r_j=0 e q=4

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	5	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,10%	0,53%	40%	803	1151	1167	2030	303	463	492	852	599	1097	6	10
6	60	0,40%	1,60%	20%	794	942	840	2729	205	247	402	1604	528	1792	21	100
10	70	0,22%	0,61%	20%	797	857	286	572	117	129	86	202	186	330	6	16
10	90	0,50%	0,92%	0%	1185	1561	919	1533	677	1086	1143	2213	1821	2881	19	42
20	120	0,72%	1,11%	0%	1268	1331	1805	3839	372	422	3417	9135	3790	9484	152	454
50	150	0,17%	0,34%	10%	1282	1381	371	659	41	48	90	255	122	297	33	112

Tabela 7.16 – Resultados computacionais para r_j=0 e q=5

Na Tabela abaixo, apresentamos os resultados comparativos entre este conjunto de testes e o conjunto de testes efectuado para r_j=U[0,100] (Secção 7.3), considerando os mesmos problemas de teste e os cinco níveis de congestionamento.

	LP-IP Gap	RL	Colunas Geradas		Tempo de CPU (s)			Nodos Avaliados
			RL	B&P	RL	B&P	Total	
	Média	%	Média	Média	Média	Média	Média	Média
r_j=0	0,50%	8%	707	1222	147	568	684	70
r_j=[0,100]	0,71%	4%	736	1359	96	417	506	93

Tabela 7.17 – Tabela comparativa - r_j=0 e r_j=[0,100]

A partir dos resultados apresentados na Tabela comparativa, podemos verificar que os valores médios dos indicadores escolhidos são ligeiramente melhores para a situação em que r_j=0, com a excepção dos tempos computacionais que são superiores. Este agravamento dos tempos computacionais tem origem nos subproblemas (note-se que o número de colunas e número de nodos gerados na árvore de pesquisa apresentam valores inferiores) e deve-se ao aumento do espaço de estados do algoritmo de programação dinâmica de solução dos subproblemas provocado pelo facto dos trabalhos estarem disponíveis mais cedo (t=0).

7.4.2.2. $r_j = 0$ e $r_j = \mathrm{U}\left[1, (\bar{p} + \bar{s})(n/m - 1)\right]$

A segunda situação que vamos analisar envolve tempos de disponibilidade dos trabalhos gerados através de um processo misto; a um primeiro subconjunto de 2*m* trabalhos (dois trabalhos por máquina) é atribuída uma data de disponibilidade igual a zero; as datas de disponibilidade dos restantes trabalhos são geradas num intervalo de tempo alargado $r_j = \mathrm{U}\left[1, (\bar{p} + \bar{s})(n/m - 1)\right]$, em que os trabalhos vão sendo disponibilizados de uma forma uniforme ao longo de um intervalo de tempo alargado que tem um limite superior, que passaremos a designar por *T'*, igual ao horizonte de programação médio estimado $T = (\bar{p} + \bar{s})n/m$ menos o tempo médio estimado de preparação e processamento de um trabalho $(\bar{s} + \bar{p})$. Os parâmetros $\bar{p}$ e $\bar{s}$ representam as médias dos tempos de processamento e de preparação das máquinas, respectivamente.

A ideia subjacente a esta configuração é a de existirem dois trabalhos por máquina já disponíveis no momento em que a programação é determinada (*t*=0) e os restantes trabalhos estarem a ser processados pelas áreas adjacentes a montante, sendo possível estimar as datas em que estes vão ficar disponíveis para processamento nas máquinas em programação; caso estes últimos trabalhos fiquem disponíveis no horizonte temporal *T'*, são incluídos no programa, com o objectivo de poder obter uma utilização mais eficiente das máquinas.

Os resultados computacionais são apresentados nas Tabelas seguintes.

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	1	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Máx	%	Média	Máx	Média	Max	Média	Máx	Média	Max	Média	Máx	Média	Máx
2	40	0,00%	0,00%	0%	202	269	2088	2276	10	15	65	101	76	116	63	80
6	60	0,00%	0,00%	0%	236	312	1470	1871	8	13	30	44	39	57	80	92
10	70	0,00%	0,00%	0%	287	338	1127	1377	7	11	19	23	26	33	70	94
10	90	0,00%	0,00%	0%	333	428	2080	2433	17	27	86	99	103	117	129	151
20	120	1,40%	13,30%	0%	472	623	1985	2367	15	25	93	107	108	121	147	172
50	150	0,00%	0,00%	0%	309	387	1239	1370	8	11	72	88	80	99	192	218

Tabela 7.18 – Resultados computacionais para *q*=1

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	2	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	2,11%	8,43%	0%	560	790	1912	2699	29	52	85	153	114	189	19	43
6	60	4,50%	37,00%	0%	464	569	1545	2339	13	23	32	75	46	98	35	56
10	70	4,74%	13,92%	0%	507	643	1031	1313	12	15	24	46	36	61	36	53
10	90	3,11%	10,64%	0%	586	799	2687	3559	25	43	83	182	108	225	78	111
20	120	2,76%	5,47%	0%	854	945	1384	2072	29	43	87	179	116	222	57	118
50	150	0,24%	2,43%	0%	677	922	520	689	9	13	14	20	24	28	50	75

Tabela 7.19 – Resultados computacionais para *q*=2

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	3	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,09%	0,94%	40%	800	1298	1531	2144	42	77	63	105	79	146	10	16
6	60	1,19%	6,39%	0%	561	662	1348	2525	16	32	29	57	45	84	30	80
10	70	1,59%	3,58%	10%	583	727	699	1121	14	20	15	31	28	51	22	39
10	90	1,10%	4,24%	0%	767	895	2131	2887	31	54	64	127	96	181	41	53
20	120	0,70%	1,70%	0%	949	1044	1124	1688	38	60	72	98	111	132	38	61
50	150	2,26%	4,01%	0%	1041	1122	527	841	20	27	100	197	121	216	103	244

Tabela 7.20 – Resultados computacionais para *q*=3

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	4	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,01%	0,63%	60%	932	1040	674	1210	49	71	62	144	74	208	4	8
6	60	0,40%	2,01%	0%	642	759	976	1897	19	35	21	53	40	75	20	52
10	70	0,67%	2,17%	10%	650	821	520	940	16	20	12	18	27	36	14	25
10	90	0,82%	3,05%	0%	903	1198	1840	2402	33	55	54	78	87	118	34	49
20	120	0,78%	1,81%	0%	1011	1071	906	1522	44	59	69	172	113	223	35	89
50	150	0,88%	1,39%	0%	1097	1193	546	848	24	33	106	213	130	232	88	198

Tabela 7.21 – Resultados computacionais para q=4

		LP-IP		RL	Colunas Geradas				Tempo de CPU (s)						Nodos	
q=	5	Gap			RL		B&P		RL		B&P		Total		Avaliados	
m	*n*	Média	Max	%	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max	Média	Max
2	40	0,70%	4,63%	50%	935	1094	958	1352	53	80	64	116	85	184	6	10
6	60	0,44%	1,92%	30%	699	952	1033	1919	20	34	20	45	34	60	26	86
10	70	0,65%	1,54%	0%	640	814	545	866	15	21	13	21	28	36	14	26
10	90	0,58%	1,54%	0%	959	1382	2612	14961	34	55	457	4221	489	4221	346	3272
20	120	0,76%	1,51%	0%	1053	1114	1035	1629	46	64	91	176	138	225	47	113
50	150	0,45%	0,84%	0%	1115	1153	390	676	23	26	57	118	80	136	45	110

Tabela 7.22 – Resultados computacionais para q=5

A Tabela 7.23 apresenta os resultados comparativos para as três situações analisadas.

	LP-IP	RL	Colunas Geradas		Tempo de CPU (s)			Nodos
	Gap		RL	B&P	RL	B&P	Total	Avaliados
	Média	%	Média	Média	Média	Média	Média	Média
r_j=0	0,50%	8%	707	1222	147	568	684	70
r_j=[0,100]	0,71%	4%	736	1359	96	417	506	93
r_j=0 e r_j=[1,T']	1,10%	6%	694	1282	24	69	89	63

Tabela 7.23 – Tabela comparativa

A partir destes resultados, podemos verificar que o modelo desenvolvido é robusto a variações das datas de disponibilidade dos trabalhos e que os melhores resultados são obtidos para a terceira situação analisada ($r_j = 0$ e $r_j = \mathrm{U}[1, T']$), com a excepção do valor do Gap LP-IP que é superior. Em particular, os resultados dos tempos computacionais justificam-se pela redução do espaço de estados do algoritmo de programação dinâmica de solução dos subproblemas provocada pelo facto de existirem trabalhos que ficam disponíveis mais tarde do que nos dois outros casos em análise.

Uma última nota para referir que o valor percentual elevado (37.0%) para o Gap LP-IP que aparece na Tabela 7.19 corresponde a um acréscimo em termos de valores absolutos de atraso de aproximadamente 6 unidades de tempo, num horizonte temporal de programação T=761.

7.5. Extensões ao Modelo

Nesta Secção fazemos um estudo do comportamento do modelo desenvolvido quando aplicado a outras funções de custo regulares (não decrescentes com os tempos de conclusão). A nossa escolha recaiu sobre a função de custo soma ponderada dos tempos de fluxo, $F_w = \sum_{j=1}^{n} w_j F_j = \sum_{j=1}^{n} w_j \left(C_j - r_j\right)$, dado que os trabalhos têm datas de disponibilidade arbitrárias e esta função de custo tem como caso particular uma das funções de custo mais estudadas e mais referidas na literatura, a soma ponderada dos tempos de conclusão, $C_w = \sum_{j=1}^{n} w_j C_j$, quando r_j=0.

Como já foi referido no Capítulo 2, o tempo de fluxo mede os tempos de espera e processamento dos trabalhos. O peso atribuído a cada trabalho pode ter várias interpretações, tais como, uma indicação da prioridade de processamento ou uma indicação do valor associado ao trabalho.

Foi realizado um conjunto de testes envolvendo os problemas de maior dimensão para cada m, e usando a janela temporal mais larga. As instâncias de teste para este conjunto de testes foram geradas seguindo o mesmo esquema apresentado na Secção 7.2, com o

devido ajuste para esta função de custo onde não existem agora datas de entrega para os trabalhos e, como tal, não se aplica a utilização de níveis de congestionamento.

Os resultados computacionais são apresentados na Tabela seguinte.

		LP-IP Gap		RL	Colunas Geradas				Tempo de CPU (s)						Nodos Avaliados	
					RL		B&P		RL		B&P		Total			
m	*n*	Média	Máx	%	Média	Máx	Média	Max	Média	Máx	Média	Max	Média	Máx	Média	Máx
2	40	0,07%	0,18%	40%	773	1182	1641	3185	266	443	649	1140	656	1327	8	14
2	50	0,08%	0,29%	20%	1101	1539	2745	5713	871	1240	2850	5872	3151	6802	10	22
4	50	0,05%	0,16%	30%	667	720	379	625	98	115	89	148	160	254	6	12
4	60	0,06%	0,31%	20%	1137	1714	1276	4057	585	847	996	3484	1383	3957	14	66
6	60	0,12%	0,43%	10%	828	986	565	1220	180	219	207	447	367	602	10	32
6	70	0,13%	0,34%	20%	1085	1539	1021	2403	501	674	848	1929	1180	2369	16	56
8	70	0,06%	0,21%	40%	944	1140	684	964	215	292	343	540	421	749	15	24
8	80	0,13%	0,26%	0%	1170	1426	949	1987	549	646	920	2026	1470	2493	16	38
10	70	0,09%	0,34%	20%	884	1148	255	445	99	135	66	117	152	207	5	12
10	90	0,14%	0,22%	0%	1212	1305	939	1629	545	656	1048	1691	1593	2185	17	38
20	100	0,13%	0,26%	0%	1017	1062	494	796	63	74	137	353	200	424	19	54
20	120	0,17%	0,31%	0%	1343	1489	1035	2792	253	293	1293	5873	1546	6120	68	390
50	150	0,13%	0,24%	10%	1381	1417	482	888	36	39	93	256	120	293	51	160
50	180	0,22%	0,30%	0%	1724	1771	1275	2189	100	112	1236	3172	1337	3272	261	784

Tabela 7.24 – Resultados computacionais para o problema $R|a_k, r_j, s_{ij}|\sum w_j F_j$

Na Tabela 7.25, apresentamos os resultados comparativos entre este conjunto de testes e o conjunto de testes efectuado para a função de custo T_w (Secção 7.3), considerando os mesmos problemas de teste.

	LP-IP Gap	RL	Colunas Geradas		Tempo de CPU (s)			Nodos Avaliados
			RL	B&P	RL	B&P	Total	
	Média	%	Média	Média	Média	Média	Média	Média
***q*=1**	0,06%	5,83%	217	1435	19	129	144	142
***q*=2**	0,40%	0,83%	441	1855	27	101	127	82
***q*=3**	1,82%	0,83%	961	1323	122	543	663	100
***q*=4**	0,95%	1,67%	1026	1310	208	1129	1331	131
***q*=5**	0,44%	6,67%	1060	872	236	611	820	63
Fw	0,12%	14,17%	1086	810	242	577	767	41

Tabela 7.25 – Tabela comparativa - T_w e F_w

Da análise comparativa dos resultados realçamos o seguinte:

- O número de soluções óptimas inteiras encontradas na fase de relaxação linear é muito superior para F_w (aproximadamente 14%);
- O número de nodos gerados na fase de partição e avaliação para F_w é claramente inferior ao melhor registo para T_w;
- A média dos tempos computacionais para F_w tem um valor muito inferior ao pior dos casos para para T_w e está próximo da média global dos tempos computacionais (considerando os cinco níveis de congestionamento) para T_w.

Com base no atrás exposto, podemos concluir que o modelo desenvolvido é igualmente eficiente na resolução de problemas $R\left|a_k, r_j, s_{ij}\right|\sum w_j F_j$.

Capítulo 8

Conclusões e Trabalho Futuro

Neste trabalho apresentou-se uma aplicação do método de partição e geração de colunas ao problema de programação de máquinas paralelas não-idênticas com tempos de preparação das máquinas dependentes da sequência de trabalhos a processar e datas de disponibilidade para as máquinas e para os trabalhos, de forma a minimizar a soma ponderada dos desvios positivos (atrasos) às datas de entrega dos trabalhos. Este problema é um caso geral dos problemas de programação de máquinas paralelas e, do conhecimento que temos da literatura, a sua solução exacta nunca foi tentada antes. O modelo desenvolvido pode ser aplicado a outros casos particulares com medidas de desempenho regulares.

Partindo de uma formulação do problema através de um modelo programação matemática com uma função de custo não linear, foi aplicado o método decomposição de Dantzig-Wolfe que conduziu a uma formulação forte do problema e em que todas as suas funções são lineares. Esta formulação foi resolvida utilizando o método de partição e geração de colunas. A alternativa seria utilizar a formulação original com o recurso a técnicas de linearização das funções não lineares, mas sabe-se que tipicamente isso produz formulações mais fracas.

Analisaram-se vários modelos de programação dinâmica para a resolução do subproblema e foi desenvolvida uma estratégia de redução do espaço de estados que permitiu acelerar de forma significativa a resolução dos subproblemas.

Verificou-se que a determinação de uma solução inicial básica admissível para o problema através de colunas artificiais conduzia a uma maior degenerescência no processo de geração de colunas e, consequentemente, maiores tempos computacionais para a resolução da relaxação linear do problema principal. Para resolver este problema foi desenvolvida uma heurística para determinação de uma solução inicial básica admissível.

Referiram-se diferentes estratégias de partição e pesquisa e definiu-se um esquema de partição e avaliação. Foi desenvolvida uma regra específica de partição não trivial e original que diminui significativamente o número de nodos explorados na árvore de pesquisa. A sua eficiência é demonstrada apresentando resultados comparativos com as regras de partição clássicas.

O estudo aprofundado da estrutura do problema permitiu desenvolver estratégias de estabilização e aceleração do algoritmo de geração de colunas que se mostraram decisivas para os bons resultados obtidos. Mostrou-se ainda que estas estratégias são válidas para uma classe alargada de problemas de programação de máquinas paralelas.

Desenvolveu-se um programa de implementação para este problema e são apresentados resultados de testes computacionais para uma gama alargada de configurações do problema.

Analisou-se a sensibilidade do modelo desenvolvido a variações nos seus principais parâmetros.

Estudou-se a extensão do modelo desenvolvido a outras funções de custo.

O modelo desenvolvido mostrou-se capaz de resolver instâncias de dimensão significativa.

São várias as direcções de investigação que podem ser exploradas em trabalhos futuros. O vasto domínio dos problemas de programação proporciona inúmeras possibilidades para a aplicação do método e, assim, construir uma base de conhecimento mais alargada que sirva de suporte ao seu desenvolvimento.

Neste trabalho, a análise dos tempos computacionais parciais revelou duas direcções importantes de investigação para o desenvolvimento do método: a fase de partição e geração de colunas, onde a estratégia de selecção dos arcos para partição se revelou muito importante, e o algoritmo de resolução dos subproblemas.

Uma das abordagens que nos parece mais promissora é a associação de procedimentos heurísticos ao método de partição e geração de colunas. Estes procedimentos heurísticos devem ser capazes de explorar de forma adequada a informação dual fornecida pelo algoritmo simplex para acelerar a resolução dos subproblemas. Devem também ser capazes de não só encontrar soluções inteiras melhores que a solução incumbente actual permitindo assim abandonar um maior número de nodos da árvore de pesquisa, como também, de ajudar a encontrar as partições que mais rapidamente conduzem à solução óptima inteira.

Bibliografia

VAN DEN AKKER, J. M., HOOGEVEEN, J. e S. VAN DE VELDE, (1999), "Parallel Machine Scheduling by Column Generation", *Operations Research*, Vol. 47, 862-872.

VAN DEN AKKER, J. M., HURKENS, C. A. J. e SAVELSBERGH, M. W. P. (2000), "Time-Indexed Formulations for Machine Scheduling: Column Generation". *INFORMS Journal on Computing,* 12:111-124.

BAKER, K. R. e MERTEN, A. G. (1973), "Scheduling with Parallel Processors and Linear Delay Costs", Naval Research Logistic Quarterly, 20:193-204.

BARNES, J. W. (1993). "A Tabu Search Experience in Production Scheduling", Annals of Operations Research, 41:141-156.

BARNES, J. W. e BRENNAN, J. J. (1977), "An Improved Algorithm for Scheduling Jobs on Identical Machines", AIIE Transactions, 9:25-31.

BARNHART, C., JOHNSON, E., NEMHAUSER, G., SAVELSBERGH M. e VANCE P., (1998), "Branch-and-Price: Column Generation for Solving Huge Integer Programs", Operations Research, 46: 316-329.

BELOUADAH, H. e POTTS, C. N. (1995), "Scheduling Identical Parallel Machines to Minimize Total Weighted Completion Time", Discrete Applied Mathematics, 48:201-218.

BEN AMOR, H., DESROSIERS, J., VALÉRIO DE CARVALHO, J.M. (2003), "Dual-optimal Inequalities for Stabilized Column Generation", *Les Cahiers de GERAD G-2003-20*. ISSN: 0711-2440.

BHASKARAN, K. e PINEDO, M. (1992), "Dispatching", *Handbook of Industrial Engineering*, G. Salvendy, 2184-2198, Wiley, New York.

BLAZEWICZ, J., DROR, M. e WEGLARZ, J. (1991), "Mathematical programming formulations for machine scheduling: A survey", *European Journal of Operational Research*, 51:283-300.

BLAZEWICZ, J., LENSTRA, J. K. e RINNOOY KAN, A. (1983), "Scheduling subject to resource constraints: classification and complexity", *Discrete Applied Mathemathics*, 5:11-24.

CHEN, Z.-L. e LEE, C. (2002), "Parallel machine scheduling with a common due window*", European Journal of Operational Research*, 136, 512-527.

CHEN, Z.-L., e POWELL, W. (1999a), "Solving parallel machine scheduling problems by column generation", *INFORMS Journal on Computing*, Vol. 11, No. 1, 78-94.

CHEN, Z.-L., e POWELL, W. (1999b), "A Column Generation Based Decomposition Algorithm for a Parallel Machine Just-In-Time Scheduling Problem*", European Journal of Operational Research*, 116, 220-232.

CHEN, Z.-L. e POWELL, W. (2003), "Exact Algorithms for Scheduling Multiple Families of Jobs on Parallel Machines, *Naval Research Logistics,* 50, 823-840.

CHENG, T. C. E. e SIN, C. C. S. (1990), "A State-of-the-Art Review of Parallel Machine Scheduling Research, European Journal of Operational Research, 47:271-292.

CHENG, R. e GEN, M. (1997), "Parallel Machine Scheduling Problems Using Memetic Algorithms", Computers & Industrial Engineering, 33:761-764.

CHRISTOFIDES, N., MINGOZZI, A. e TOTH, P. (1981), "State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems", Networks 11, 145-164.

DANTZIG, G. e WOLFE, P. (1960), "Decomposition Principle for Linear Programs", *Operations Research* 8:101-111.

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SOLOMON, M. M., SOUMIS, F. e VILLENEUVE, D. (1998), "A unified framework for deterministic time constrained vehicle routing and crew scheduling problems", Fleet management and logistics, Eds. T. G. Crainic and G. Laporte. Kluwer Academic Publishers, 3:57-93.

DESROCHERS, M., DESROSIERS, J. e SOLOMON, M. M. (1992), "A New Optimization Algorithm for the Vehicle Routing Problem With Time Windows", Operations Research, Vol. 40, No. 2, 342-354.

DESROSIERS, J., DUMAS, Y., SOLOMON, M., e SOUMIS, F. (1995), "Time Constrained Routing and Scheduling", Handbook in Operations Research & Management Science, *Elsevier Science*,Vol. 8, 2:35-139.

DESROSIERS, J., SOUMIS, F. e DESROCHERS, M. (1984), "Routing with Time Windows by Column Generation", John Wiley & Sons, Vol 14, 545-565.

DROR, M., (1994), "Note on the complexity of the shortest path models for column generation in VRPTW", *Operations Research*, Vol. 42, No. 5, 977-978.

DU, J. e LEUNG, J.Y.-T. (1990), "Minimizing total tardiness on one machine is NP-hard". *Mathematics of Operations Research*, Vol. 15, No. 5, 483-495.

EASTMAN, W. L., EVEN, S. e ISAACS, I. M. (1964), "Bounds for the Optimal Scheduling of n Jobs on m Processors", Management Science, 11(2):268-279.

ELMAGHRABY, S. E. e PARK, S. H. (1974), "Scheduling Jobs on a Number of Identical Machines", AIIE Transactions, 6:1-13.

EMMONS, H. (1969), "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness", Operations Research, 17:701-715.

FRANÇA, P.M., GENDREAU, M., LAPORTE, G. e MULLER, F.M, (1996), "A Tabu Search Heuristic for the Multiprocessor Scheduling Problem with Sequence Dependent Setup Times", International Journal of Production Economics, 43:79.

FRANÇA, P.M., MENDES, A. S. e MOSCATO, P. (1999), ”Memetic Algorithms to Minimize Tardiness on a Single Machine with Sequence Dependent Setup Times”, DSI99- 5th International Conference of the Decision Sciences Institute.

FRENCH, S., (1982), “Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop”, Ellis Horwood Ltd.

GAREY, M. e JOHNSON, D. (1979), “Computers and Intractability, A Guide to the Theory of NP-Completeness”, W. H. Freeman and Company.

GRAHAM R.L., LAWLER, E.L., LENSTRA, J.K. e RINNOOY KAN, A.H.G. (1979), “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”, Annals of Discrete Mathematics, 5:287-326.

HO, J. C. e CHANG, Y. L. (1991), “Heuristics for Minimizing the Mean Tardiness for *m* Parallel Machines“. *Naval Research Logistics* 38:367-381.

HO, J. e CHANG, Y. (1995), “Minimizing the Number of Tardy Jobs for *m* Parallel Machines“. *European J. Oper. Res.* 84:343-355.

HOOGEVEEN, J., LENSTRA, J. K. e VAN DE VELDE, S. (1997), “Sequencing and Scheduling - Annotated Bibliographies in Combinatorial Optimization”, Eds. M. Dell’Amico, F. Maffioli and S. Martello, John Wiley & Sons, 12:181-197.

HOUCK, J. D. J., PICARD, J. C., QUEYRANNE, M., e VEMUGANTI, R. R. (1980), “The Travelling Salesman Problem as a Constrained Shortest Path Problem: Theory and Computational Experience”. *Operations Research*, 17:93-109.

ILOG, INC., (1998), “Version 6.0 of Using the CPLEX Callable Library“.

KANG, S., MALIK, K. e THOMAS, L. (1999), “Lotsizing and Scheduling on Parallel Machines with Sequence-Dependent Setup Costs”, *Management Science*, Vol 45, 2:273-289.

KOULAMAS, C., ANTONY, S.R. e JAEN, R. (1994), ”A Survey of Simulated Annealing Applications to Operations Research Problems”, Omega, International Journal of Management Science, 22:41-56.

VAN LAARHOVEN, P. J. M., AARTS, E. H. L., e LENSTRA, J. K. (1992),”Job Shop Scheduling by Simulated Annealing”, Operations Research, 40:113-125.

LASDON, L., (1970), “Optimization Theory for Large Systems”, Case Western Reserve University, The Macmillan Company, Collier-Macmillan Limited, London.

LAWLER, E.L. e MOORE, J. M. (1969), “A Functional Equation and its Application to Resource Allocation and Sequencing Problems”. *Management Science,* 16:77-84.

LAWLER, E.L., (1977), “A ‘Pseudopolynomial’ Time Algorithm for Sequencing Jobs to Minimize Total Tardiness”, *Annals of Discrete Mathematics* 1:331-342.

LEE, Y.K., BHASKARAN, K. e PINEDO, M. (1997), “A Heuristic to Minimize the Total Weighted Tardiness with Sequence Dependent Setup Times”, *IIE Transactions*, 29:45-52.

LENSTRA, J.K., RINNOOY KAN, A.H.G. e BRUCKER, P. (1977), “Complexity of Machine Scheduling Problems.”, *Annals of Discrete Mathematics* 1:343-362.

LEUNG, J. Y-T., (1982), “On Scheduling Independent Tasks with Restricted Execution Times”, Operations Research, 30:163-171.

LEUNG, J. Y-T., (2004), “Handbook of Scheduling: Algorithms, Models, and Performance Analysis” – Chapman & Hall/CRC Computer & Information Science Series. ISBN:1584883979. http://web.njit.edu/~leung/handbook.

LÜBBECKE, M. e DESROSIERS, J. (2002), “Selected topics in Column Generation”, *Les Cahiers du GERAD, G-2002-64*. ISSN: 0711-2440.

MEHROTRA, A. e TRICK, M. (1996), “A column generation approach for graph colouring”, *INFORMS J. Comput*., 8(4):344-354.

MENDES, A., MULLER, F., FRANÇA, P.M., e MOSCATO, P. (1999), ”Comparing Meta-Heuristics Approaches for Parallel Machine Scheduling Problems with Sequence-Dependent Setup Times”, CARS & FOF’99 - 15th International Conference on CAD-CAM Robotics and Factories of the Future, Brasil.

MORTON, T. E. e RAMNATH, P. (1992) “Guided Forward Search in Tardiness Scheduling of Large One Machine Problems”, GSIA, Carnegie Mellon University.

MURTY, K. G. (1995), “Operations Research: Deterministic Optimization Models”, Prentice Hall, Englewood Cliffs, New Jersey.

NEMHAUSER, G. e WOLSEY, L. (1988), “Integer and Combinatorial Optimization”, Wiley & Sons, New York.

PANWALKER, S.S. (1990), “Scheduling Rules Revisited”, *Proceedings of the Joint National Meeting of Scheduling*.

PANWALKER, S.S. e ISKANDER, W. (1977), “A Survey of Scheduling Rules”, *Operations Research*, 25:45-61.

PINEDO, M., (1995), “Scheduling: Theory, Algorithms, and Systems”, Prentice Hall.

PINEDO, M., (2001), “Scheduling: Theory, Algorithms, and Systems”, second edition, Prentice Hall–http://www.stern.nyu.edu/om/pinedo/ book1.html.

PINEDO, M. e X. CHAO, (1999), “Operations Scheduling with Applications in Manufacturing and Services”, McGraw-Hill – ISBN:0-07-289779-1. http://www.stern.nyu.edu/om/pinedo/ book2.html).

POTTS, C. N. e VAN WASSENHOVE, L. N. (1991), “Single Machine Tardiness Sequencing Heuristics”, IIE Trans. 23:346-354.

POTTS, C. N. e VAN WASSENHOVE, L. N., (1992). “Single Machine Scheduling to Minimize Total Late Work”, Operations Research, 40:586-595.

PUCHINGER, J. e RAIDL, G. R., (2004). “Models and algorithms for three-stage two-dimensional bin packing”, Technical Report TR 186-1-04-04, Institute of Computer Graphics and Algorithms, Vienna University of Technology, submitted to the *European Journal of Operational Research*.

RAGATZ, G. L., (1993), “A Branch-and-Bound Method for Minimum Tardiness Sequencing on a Single processor with Sequence Dependent Setup Times”, *Proceedings: Twenty-fourth Annual Meeting of the Decision Sciences Institute*, 1375-1377.

ROTHKOPF, M. H. (1966), “Scheduling Independent Tasks on Parallel Processors”, *Management Science*, 12:437-447.

SAHNI, S. K. (1976), “Algorithms for Scheduling Independent Tasks”, *Journal of the Association for Computing Machinery*, 23:116-127.

SARIN, S. C., AHN S. e BISHOP, A. B. (1988), “An Improved Scheme for the Branch-and-Bound Procedure of Scheduling *n* Jobs on *m* Parallel Machines to Minimize the Total Weighted Flowtime”, *International Journal of Production Research*, 26:1183-1191.

SAVELSBERGH, M., (1997), “A branch-and-price algorithm for the generalized assignment problem”, *Operations Research*, 45(6):831-841.

SMITH, W. E. (1956), “Various Optimizers for Single-Stage Production”, Naval Research Logistics Quarterly, 3:325-333.

SOUMIS, F. (1997), “Decomposition and Column Generation”, Annotated Bibliographies in Combinatorial Optimization. Eds. M. Dell’Amico, F. Maffioli e S. Martello, John Wiley & Sons, 8:115-126.

TAN, K. C. e NARASINHAM, R. (1997), “Minimizing Tardiness on a Single processor with Sequence Dependent Setup Times: a Simulated Annealing Approach”, Omega, 6:619-634.

TAN, K. C., NARASINHAM, R., RUBIN P. A. e RAGATZ, G. L. (2000), "A Comparison of Four Methods for Minimizing Total Tardiness on a Single processor with Sequence Dependent Setup Times: a Simulated Annealing Approach", Omega, 28:313-326.

VALÉRIO DE CARVALHO, J. M., (1999), "Exact solution of bin-packing problems using column generation and branch-and-bound". *Ann. Oper. Res*., 86:629-659.

VALÉRIO DE CARVALHO, J. M., (2004), "Using extra dual cuts to accelerate column generation", *INFORMS J. Comput*., (in press).

VANCE, P., (1998), "Branch-and-price algorithms for the one-dimensional cutting stock problem". *Comp. Optim. Appl*., 9(3):211-228.

VAN DE VELDE, S. L. (1991), "Machine Scheduling and Lagrangean Relaxation", PhD Thesis, Technische Universiteit Eindhoven, Amsterdam.

WEBSTER, S. (1992), "New Bounds for the Identical Parallel Processor Weighted Flow Time Problem", Management Science, 38:124-136.

WEBSTER, S. (1993), "A Priority Rule for Minimizing Weighted Flow Time in a Class of Parallel Machine Scheduling Problems", European Journal of Operational Research, 70:327-334.

WEBSTER, S. (1995), "Weighted Flow Time Bounds for Scheduling Identical Processors", European Journal of Operational Research, 80:103-111.

WILKERSON, L. J. e IRWIN, J. D. (1971), "An Improved Algorithm for Scheduling Independent Tasks", AIIE Transactions, 3:239-245.

Anexo A. Programa PMPNI

O fluxograma do programa PMPNI é apresentado nas duas Figuras seguintes.

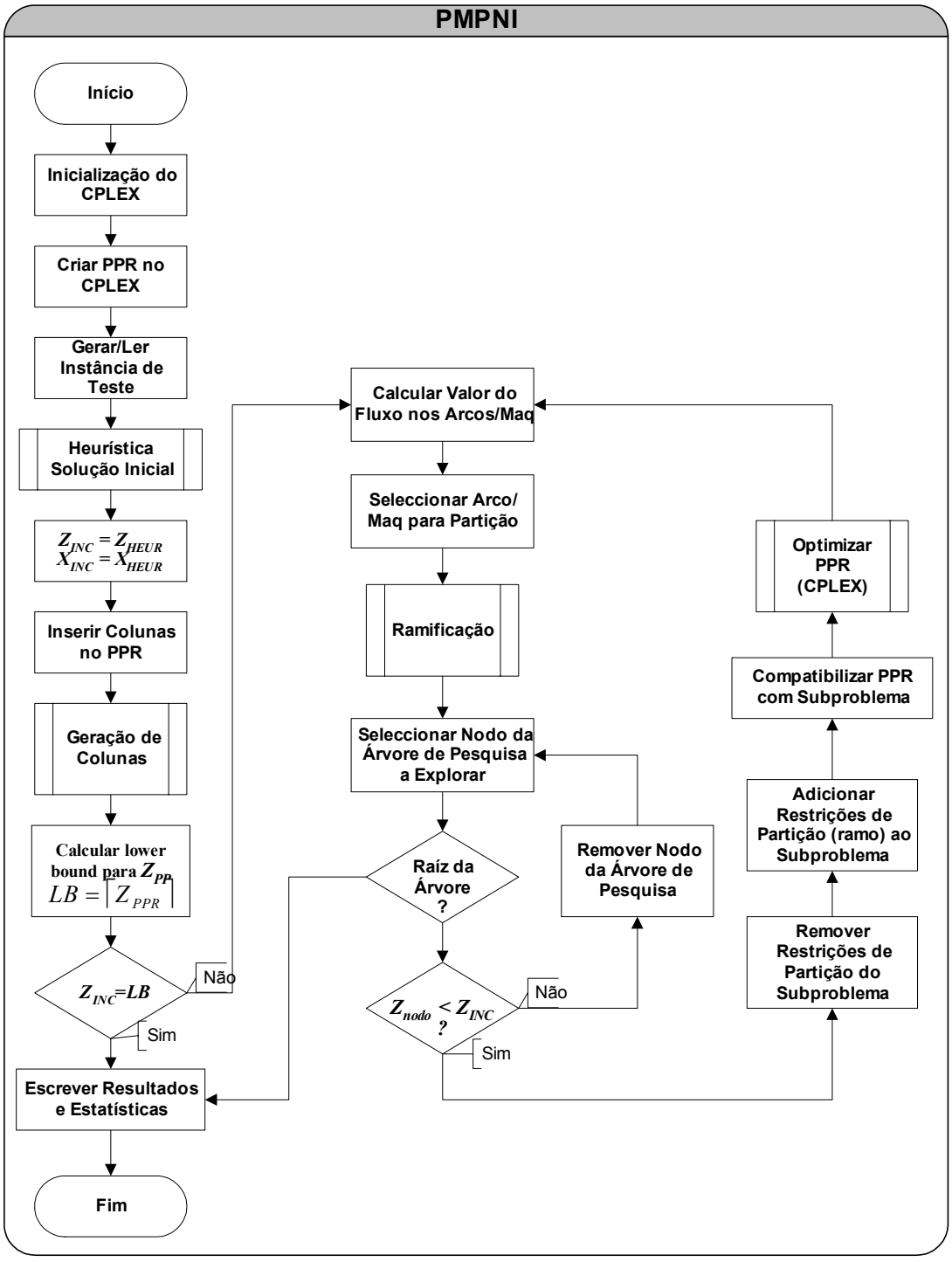


Figura A 1 – Fluxograma do programa PMPNI

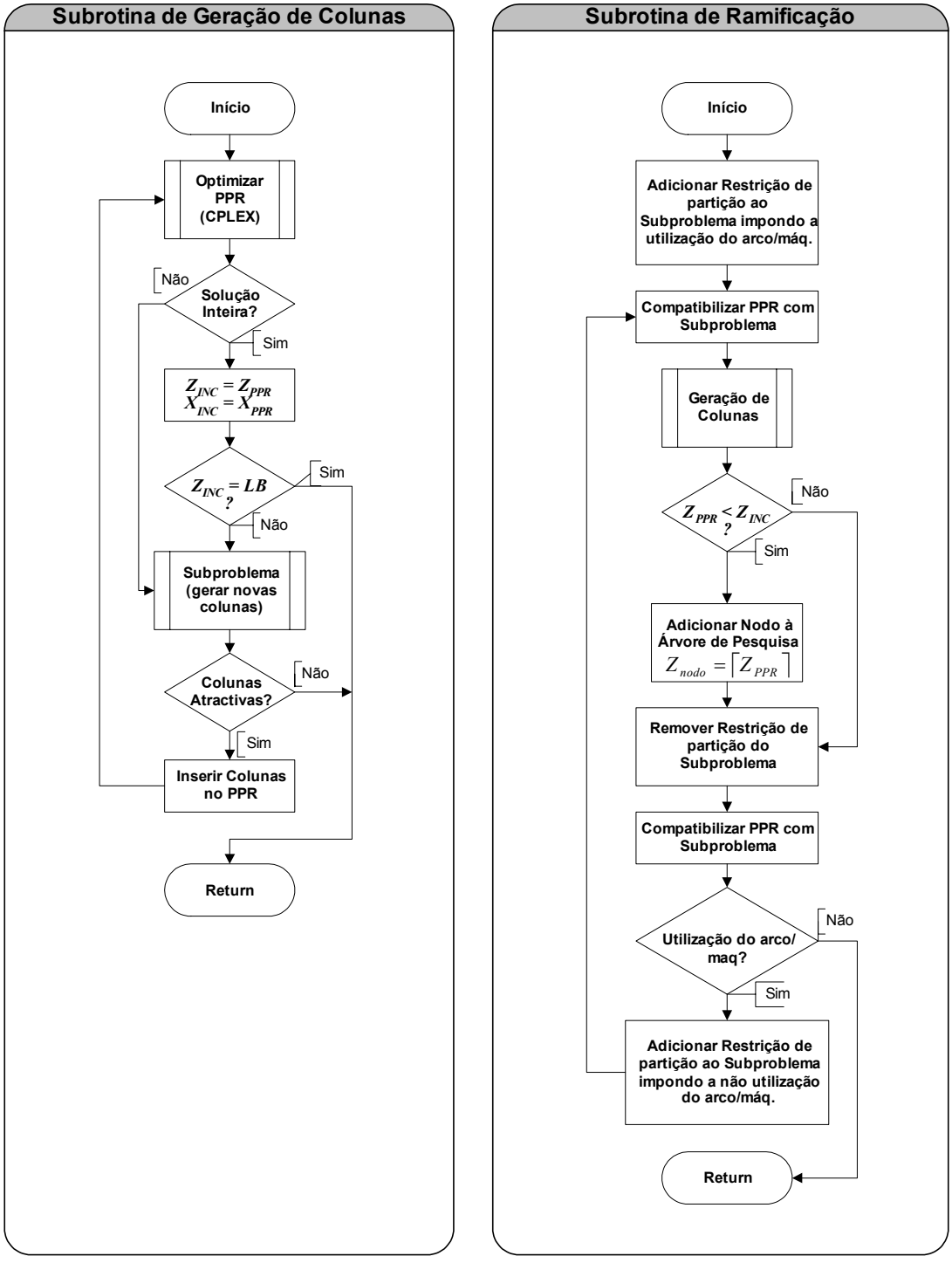


Figura A 2 – Fluxograma do programa PMPNI (Subrotinas)

Anexo B. Biblioteca de Funções CPLEX

O software CPLEX 6.0 consiste num conjunto de funções ("*callable library*") programadas na linguagem C e numa interface que permite ao utilizador introduzir um modelo de programação linear, inteira, mista ou programação convexa quadrática e optimizá-lo, obtendo resultados como a solução óptima, as variáveis duais e resultados relativos a análise de sensibilidade. É ainda possível modificar o problema, inserindo novas variáveis, restrições ou alterando coeficientes.

Como é representado esquematicamente na Figura que se segue, o programa pode ser utilizado de duas formas: através da referida interface ou criando um programa que se sirva da biblioteca de funções.

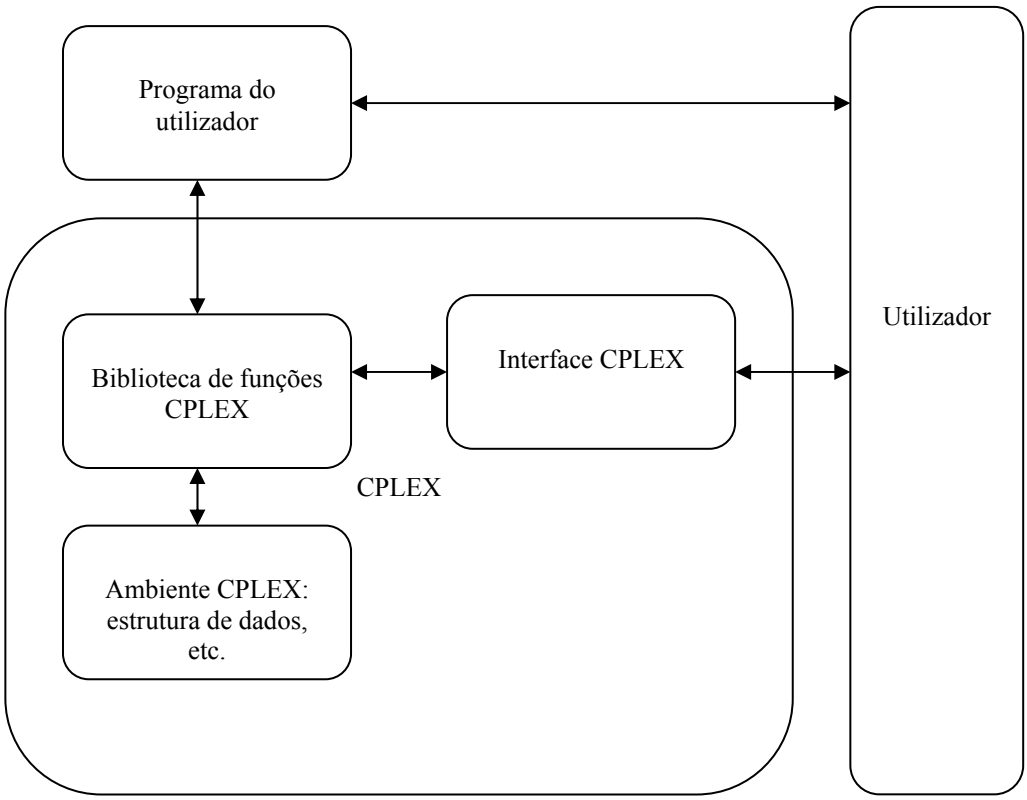


Figura B 1 – Utilização do CPLEX

As funções da biblioteca CPLEX podem ser divididas em seis grupos, cada um com os seguintes propósitos:

- Criação e optimização do problema, obtenção de resultados;
- Entrada e saída de dados;
- Modificação do problema;
- Acesso ao problema e aos resultados;
- Acesso e alteração dos parâmetros de controlo;
- Escrita e leitura de ficheiros.

Um programa que obtenha a solução óptima de um modelo de programação linear, com base na biblioteca de funções CPLEX, deverá incluir os seguintes passos:

1. Inicializar o ambiente CPLEX invocando a função CPXopenCPLEXdevelop;
2. Construir a estrutura de dados que corresponda ao modelo;
3. Criar o problema no ambiente CPLEX invocando a função *CPXcreateprob*;
4. Copiar a estrutura de dados para o ambiente CPLEX invocando a função *CPXcopylp*;
5. Optimizar o problema invocando a função *CPXprimopt*;
6. Ler os resultados invocando a função *CPXsolution*
7. Libertar o problema invocando a função *CPXfreeprob*;
8. Finalizar o ambiente CPLEX invocando a função *CPXcloseCPLEX*.

Algumas funções são referidas a título exemplificativo, podendo invocar-se funções diferentes. Por exemplo, se se pretender que a optimização seja feita com base no algoritmo simplex dual a função invocada no passo 5 deveria ser *CPXdualopt*.

Como se pode observar no esquema inicialmente apresentado, a estrutura de dados CPLEX (que inclui não só a representação do problema como os resultados relativos à sua optimização) apenas pode ser acedida através das funções da biblioteca CPLEX. Há assim a necessidade de o utilizador manter uma estrutura de dados para comunicar com o ambiente CPLEX.

Se ocorrer um erro, a função retorna um valor inteiro diferente de zero, podendo o erro ser identificado com a função *CPXgeterrorstring* ou através da documentação do *software*. A variável *env* corresponde a um apontador para o ambiente CPLEX e a variável *lp* corresponde a um apontador para o problema criado.