Universidade do Minho
Escola de Engenharia

Henrique Daniel Oliveira Lopes

Aplicação de um algoritmo de pesquisa meta-heurística por geração de colunas (SearchCol) ao problema de máquinas paralelas

Outubro de 2012

Aplicação de um algoritmo de pesquisa meta-heurística por geração de colunas (SearchCol) ao problema de máquinas paralelas
Henrique Daniel Oliveira Lopes
UMinho | 2012

Universidade do Minho
Escola de Engenharia

Henrique Daniel Oliveira Lopes

Aplicação de um algoritmo de pesquisa meta-heurística por geração de colunas (SearchCol) ao problema de máquinas paralelas

Dissertação de Mestrado
Mestrado em Engenharia de Sistemas

Trabalho realizado sob a orientação do
Doutor Filipe Pereira Cunha Alvelos

Outubro de 2012

Agradecimentos

Gostaria de expressar o meu agradecimento:

Ao meu orientador, o professor Filipe Alvelos, pelo incentivo que me deu no projeto SearchCol e por partilhar o seu entusiasmo por esta área. Agradeço também sua ajuda e disponibilidade no esclarecimento de dúvidas e pelos conselhos que me deu no aperfeiçoamento da tese.

Quero também agradecer ao professor Manuel Lopes pelo apoio e disponibilidade que demonstrou no projeto SeachCol, assim como por toda a ajuda na compreensão do problema em estudo, e na explicação das heurísticas dos subproblemas.

A todos os meus colegas do departamento pelo apoio e amizade.

Por fim, quero agradecer a toda a minha família, em especial aos meus pais e irmãos, por todo o encorajamento e compreensão que sempre tiveram.

Esta dissertação foi desenvolvida no âmbito do projeto "SearchCol - Metaheuristic search by column generation" (PTDC/EIA-EIA/100645/2008), financiado por fundos nacionais através da Fundação para a Ciência e para a Tecnologia (PIDDAC) e co-financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER) através do COMPETE - Programa Operacional Factores de Competitividade (POFC).

Resumo

Neste trabalho é apresentada a aplicação de um algoritmo de pesquisa meta-heurística por geração de colunas (SearchCol) ao problema de máquinas paralelas não idênticas, com tempos de preparação dependentes da sequência, com o objetivo de minimizar a soma ponderada dos atrasos.

O SearchCol é um método híbrido que combina a geração de colunas com meta-heurísticas com o objetivo de obter boas soluções para problemas de otimização combinatória em tempos aceitáveis. A ideia chave é que a solução de um problema é vista como uma combinação de soluções de problemas de menor dimensão (subproblemas). As soluções dos subproblemas são obtidas pela geração de colunas e sua combinação é feita por uma meta-heurística.

Neste trabalho, são testadas diferentes políticas de inserção de colunas no problema mestre restrito juntamente com os diferentes algoritmos de resolução do subproblema, nomeadamente o algoritmo de programação dinâmica que resolve o subproblema de forma exata e duas heurísticas. São apresentados resultados de testes computacionais para afinação dos parâmetros do SearchCol. Os resultados do SearchCol são comparados com os resultados de um algoritmo de partição e geração de colunas (*Branch-and-Price*) e uma heurística baseada em geração de colunas e programação inteira mista.

Abstract

This work presents the application of a metaheuristic search by column generation (SearchCol) algorithm to an unrelated parallel machines problem, with sequence-dependent setup times and the objective of minimizing the total weighted tardiness.

The SearchCol method consists in a hybridization of column generation with metaheuristics to obtain good solutions for combinatorial optimization problems in acceptable times. The central idea is that the solution of a problem can be seen as a combination of solutions of smaller problems (subproblems). The solutions of subproblems are obtained by column generation and their combination is done by a meta-heuristics.

In this work, different policies insertion of columns in the restricted master problem, are tested, along with different subproblem solving algorithms, namely the dynamic programming algorithm that solves the subproblem accurately and two heuristics. We present the results of computational tests that lead to tuning of SearchCol parameters. The results of SearchCol are compared with the results of a Branch-and-Price algiritm and a heuristic based on column generation and mixed integer programming.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os problemas de escalonamento (*scheduling*) têm, de uma forma geral, como objetivo a definição dos instantes temporais em que um conjunto de tarefas são processadas por um conjunto de recursos de forma a minimizar uma medida tipicamente relacionada com o tempo. Este tipo de problemas tem sido estudado desde há largas décadas (Brucker, 2004). Uma das áreas onde este tipo de problemas tem mais aplicação é no escalonamento da produção. Nesta dissertação aborda-se um desses problemas.

O problema em estudo é designado por problema de máquinas paralelas, no qual se considera um conjunto de máquinas a funcionar em paralelo, um conjunto de tarefas com datas de disponibilidade e de entrega. Considera-se ainda que existem tempos de preparação que dependem da sequência das tarefas e que os tempos de processamento das tarefas em cada máquina são diferentes. O objetivo é minimizar a soma ponderada dos desvios positivos (atrasos) em relação às datas de entrega dos trabalhos.

Os métodos de resolução para problemas de otimização combinatória, grupo onde se pode inserir o problema estudado nesta dissertação, podem ser divididos em dois grupos: os métodos exatos e os métodos aproximados (ou heurísticos). Os métodos exatos garantem que se obtém a solução óptima. Por sua vez, os métodos aproximados não garantem a obtenção de uma solução ótima, perseguindo boas soluções em períodos de tempo aceitáveis.

Um dos métodos exatos mais utilizados, que é implementado em, virtualmente, todos

os *solvers* para programação inteira é o método de partição e avaliação (*branch-and-bound*), que faz uso da enumeração implícita para obter uma solução ótima. Por outro lado os métodos heurísticos são bastante utilizadas em problemas difíceis e de grande dimensão para os quais a utilização de métodos exatos é computacionalmente inviável. Exemplos de métodos aproximados são os métodos construtivos, os métodos de pesquisa local e as meta-heurísticas (como os algoritmos genéticos e pesquisa tabu). Um dos métodos que tem tido mais sucesso na resolução do problema de máquinas paralelas e problemas próximos baseia-se em geração de colunas (Desaulniers et al., 2005). Neste método, a resolução de relaxações lineares de um modelo de programação inteira é feita através da interação entre modelos de programação linear e subproblemas que podem ser resolvidos por métodos específicos para o problema em questão.

No caso do problema a abordar nesta dissertação e problemas próximos, o subproblema da geração de colunas é resolvido por programação dinâmica. Exemplo deste tipo de abordagens podem ser encontrados em Akker et al. (1999, 2005); Chen e Powell (1999); Lopes e Valério de Carvalho (2007). Em todos estes casos o método de geração de colunas é combinado com o método de partição e avaliação (*branch-and-bound*) para a obtenção de soluções ótimas.

Nesta dissertação pretende-se explorar a combinação de um método exato com um método aproximado com base na *framework* SearchCol (Metaheuristic Search by Column Generation) (Alvelos et al., 2010). Este *framework* visa a obtenção de boas soluções para diferentes problemas de otimização combinatória. A ideia central é que a solução de um problema pode ser vista como a combinação de soluções de problemas de menor dimensão (subproblemas). As soluções dos subproblemas são obtidas pela geração de colunas e a sua combinação é feita por uma meta-heuristica. Assim, para que um problema possa ser resolvido por uma abordagem SearchCol tem de poder ser decomposto e a sua relaxação linear ser resolvida por um algoritmo de geração de colunas (que fornece o espaço de soluções a ser usado pela meta-heurística). Em relação ao método de geração de colunas será estudada qual a política de inserção de colunas no problema mestre restrito, que melhor se aplica ao problema das máquinas paralelas não idênticas. Será também estudada a utilização de heurísticas na resolução do subproblema.

A dissertação é composta por 8 capítulos, incluindo este, que estão organizados da seguinte forma.

No Capítulo 2 é feita uma introdução geral aos problemas de escalonamento e em particular aos problemas de máquinas paralelas. Explora-se os diversos tipos de problemas e no final introduz-se o problema abordado nesta dissertação.

No Capítulo 3 é apresentada uma revisão da literatura relativa aos problemas de máquinas paralelas. Esta revisão foca-se nos problemas sem lotes e com tempos de preparação. Para estes, são apresentadas heurísticas, quatro tipos de modelos de programação inteira e para cada tipo é apresentado uma formulação. No final, aborda-se algoritmos de partição e avaliação.

No Capítulo 4 é então definido o problema de máquinas paralelas não idênticas, abordado nesta dissertação. Para três formulações apresentadas no Capítulo 3, apresenta-se as alterações necessárias para adaptar cada uma das formulações ao problema abordado nesta dissertação.

No Capítulo 5 é descrito como se obtém o ótimo linear pelo método de geração de colunas. Inicialmente é apresentado um modelo de decomposição para o problema de máquinas paralelas e para esse modelo, apresenta-se a formulação completa, utilizando um exemplo com apenas 3 tarefas e 2 máquinas. De seguida introduz-se o método de geração de colunas e descreve-se a heurística que gera a solução inicial. Apresenta-se os algoritmos que resolvem, tanto de forma exata como heurística, o subproblema. Descrevem-se alternativas de inserção das colunas no problema mestre restrito. No final apresenta-se, para um pequeno exemplo, as etapas da resolução da relaxação linear pelo método de geração de colunas.

O Capítulo 6 é dedicado ao SearchCol, descrevendo-se como este método é aplicado à resolução do problema das máquinas paralelas.

No Capítulo 7 são apresentados e analisados os testes computacionais. Os testes são apresentados na mesma ordem pelo que os parâmetros foram avaliados, até chegar à afinação dos parâmetros finais que são utilizados nos testes finais do algoritmo SearchCol.

No Capítulo 8 apresentam-se as conclusões deste trabalho.

Capítulo 2

Problemas de Escalonamento

De uma maneira geral, nos problemas de escalonamento (*scheduling*) temos um conjunto de n tarefas $T=\{T_1, ..., T_n\}$, e um conjunto de m máquinas $M=\{M_1, ..., M_n\}$, onde T e M são conjuntos finitos. O objetivo deste tipo de problemas consiste na afetação das tarefas pelas máquinas e na definição dos períodos temporais que cada tarefa é executada na máquina. O planeamento resultante da afetação será ótimo se minimizar ou maximizar um critério de otimalidade ou função objetivo que se pretenda atingir para o caso em estudo. Num problema de escalonamento assume-se que em qualquer instante, uma máquina não processa mais de uma tarefa. São três os elementos principais que caraterizam os problemas de escalonamento: as máquinas, as tarefas e o critério de otimalidade. A caraterização destes elementos será centrada nos problemas de escalonamento determinísticos e máquinas paralelas. Assume-se que os valores de todos os parâmetros tomam sempre valores inteiros, o que não torna os modelos mais restritos, visto que os valores indicam normalmente tempos, e que a variação da escala permite uma grande flexibilidade. Para uma revisão mais detalhada de problemas de escalonamento ver Baker e Trietsch (2009); Brucker (2004); Pinedo (2008).

2.1 Caraterização das tarefas

Uma tarefa consiste num conjunto de trabalhos ou operações. Quando uma tarefa é constituída por mais do que um trabalho, temos famílias ou lotes (*batch*) de trabalhos.

Normalmente um lote contém trabalhos da mesma família. Quando uma tarefa consiste num trabalho estes podem ser vistos como sendo a mesma coisa. Quando um modelo permite que se interrompa a execução de uma tarefa, com o objetivo de a retomar mais tarde, diz-se que o modelo permite interrupção (*preemption*). A continuação da execução da tarefa interrompida pode ser efetuada noutra máquina. Podem ainda ser definidas relações de dependência entre as tarefas, como por exemplo, poder ser necessário terminar a tarefa T_z antes da tarefa T_q. Estas dependências são normalmente definidas num grafo.

Em relação às tarefas podemos ainda ter a seguinte informação:

- Instante de tempo que a tarefa fica disponível para ser processada, r_j, em que j representa a tarefa - (*release date*);
- Instante de tempo que a tarefa deve ficar concluída, d_j, em que j representa a tarefa - Data de entrega (*due date*);
- Peso/prioridade associado a uma tarefa, w_j, em que j representa a tarefa;
- O tempo de processamento da tarefa em cada máquina. Assim p_{jk} representa o tempo de processamento da tarefa j na máquina k. Caso o tempo não dependa da máquina em questão temos apenas p_j, mas mais à frente explicaremos melhor esta caraterística que pode depender do tipo de máquina;
- Tempo de preparação dependente das tarefas, s_{ij}, onde i representa a tarefa anteriormente executada e j a tarefa que se vai executar - (*setup*). Estes tempos são independentes da máquina.

2.2 Caraterização das máquinas

Nos problemas de escalonamento podemos ter apenas uma ou várias máquinas. No caso de várias máquinas estas são paralelas se executarem todas a mesma função, ou dedicadas caso executem diferentes funções. Como exemplo de problemas em que as máquinas são dedicadas temos: *flow shop, job shop* e *open shop*. Dentro das máquinas paralelas podemos identificar três tipos:

- Idênticas (*identical*) - As tarefas têm tempos de execução igual em cada máquina, ou seja, iguais velocidades de processamento;
- Uniformes (*uniform*) - Diferentes velocidades de processamento das tarefas, mas a velocidade das máquinas é constante; não depende da tarefa a executar;
- Não idênticas ou não relacionadas (*unrelated*) - Velocidade das máquinas dependentes das tarefas a executar.

2.3 Critério de otimização

Para cada tarefa j podemos ter as seguintes medidas:

- Tempo de conclusão, C_j,
- Tempo de permanência no sistema (*flow time*), $F_j = C_j - r_j$;
- Desvio em relação à data de entrega (*lateness*), $L_j = C_j - d_j$;
- Avanço (*earliness*), $E_j = \max(0, d_j - C_j)$;
- Atraso (*tardiness*), $T_j = \max(0, C_j - d_j)$;
- Tarefa em atraso (*tardy job*), $U_j = 1$ caso $C_j > d_j$, e $U_j = 0$ caso contrário.

A partir destas medidas podemos detalhar algumas das medidas de desempenho ou critérios de otimalidade:

- Instante de conclusão máximo (*makespan*), $C_{\max} = \max\{C_j\}$;
- Maior desvio à data de entrega (*maximum lateness*), $L_{\max} = \max\{L_j\}$;
- Soma dos tempos de conclusão (*total completion time*), $\sum C_j$;
- Soma ponderada dos tempos de conclusão (*total weighted completion time*), $\sum w_j C_j$;
- Soma dos tempos de fluxo (*total flow time*), $\sum F_j$;

- Soma ponderada dos tempos de fluxo (*total weighted flow time*), $\sum w_j F_j$;
- Soma dos atrasos (*total tardiness*), $\sum T_j$;
- Soma ponderada dos atrasos (*total weighted tardiness*), $\sum w_j T_j$;
- Número de tarefas em atraso (*number of tardy jobs*), $\sum U_j$.

Estas medidas são exemplos de funções objetivo que se consideraram importantes referir para o problema, embora existam outras que envolvem o avanço E_j ou a média de uma determinada medida, como por exemplo o atraso médio $T' = \frac{\sum T_j}{n}$.

Por fim uma função objetivo que é não decrescente em C_j é chamada de regular, como por exemplo, $C_{\max}$. Por outro lado, qualquer função objetivo que contenha E_j é considerada não regular pois, quanto menor for C_j maior E_j, isto considerando que a tarefa é concluída antes da sua data de entrega d_j.

2.4 Classificação

Os problemas de escalonamento podem ser classificados pela notação $\alpha|\beta|\gamma$, proposta por Graham et al. (1979), onde α especifica a caraterística das máquinas, β as características das tarefas e γ os critérios de otimização. Nesta explicação da notação, iremos dar mais ênfase aos problemas das máquinas paralelas.

O campo α, é composto por dois elementos, $\alpha_1\alpha_2$. O primeiro representa o tipo e o segundo o número de máquinas:

$\alpha_1 = P$, máquinas idênticas;

$\alpha_1 = Q$, máquinas uniformes;

$\alpha_1 = R$, máquinas não idênticas;

$\alpha_2 = 1$, uma única máquina. Quando α_1 é omitido apenas temos o número de máquinas. Por sua vez, quando α_2 é omitido o número de máquinas é variável.

Por exemplo, quando $\alpha = P2$ temos um problema de máquinas idênticas, com exatamente duas máquinas. Para além destes casos existem problemas, como o Job shop ou Flow shop, que também são contemplados pela notação mas que não são referidos neste trabalho.

Como vimos anteriormente, β faz referência às caraterísticas das tarefas e pode não conter nenhum elemento, ou vários elementos. Restrições de precedência ($\beta = prec$, $\beta = tree$, $\beta = chains$), se uma tarefa tem datas de disponibilidade ($\beta = r_j$), se a interrupção de uma tarefa é permitida ($\beta = pmtn$) ou até se o modelo contempla tempos de preparação ($\beta = s_{ij}$), são exemplos de caraterísticas detalhadas neste campo. Quando num modelo é permitida interrupção e as tarefas têm data de disponibilidade temos:

$\beta = pmtn; r_j$, onde $pmtn$ indica a possibilidade de interrupção e r_j indica que as tarefas têm datas de disponibilidade.

Por último, no campo γ são representadas as medidas de desempenho e alguns exemplos foram já descritos na secção 2.2. Como exemplo, podemos ter um problema em que o objetivo é minimizar o número de tarefas em atraso e nesse caso $\gamma = \sum U_j$.

2.5 Problema de máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência

Nesta dissertação aborda-se um problema de sequenciamento de tarefas em máquinas paralelas, onde as máquinas são não relacionadas. Como vimos isso implica que o tempo de processamento de cada tarefa seja diferente para cada máquina. Não temos lotes de trabalhos, e as tarefas consistem num único trabalho onde não é possível interromper o seu processamento. Cada tarefa tem um peso/prioridade, uma data de entrega e uma data de disponibilidade para ser processada. Para além disso temos também tempos de preparação que são dependentes da sequência em que as tarefas são executadas, independentemente da máquina onde é feito esse processamento. As máquinas estão inicialmente preparadas

para receber uma tarefa e têm também uma data de disponibilidade. Esta data representa o instante em que as máquinas podem começar a processar ou a serem preparadas para receber tarefas.

De um modo geral, nos problemas de escalonamento são definidos os instantes de tempo em que as tarefas começam a ser processadas nas máquinas. Apesar disso, para este problema, a sequência de afetação é suficiente para conseguirmos definir os instantes em que as tarefas são processadas nas máquinas. Isto acontece porque não compensa haver tempos sem processamento, ou sem preparação. Normalmente estes casos só acontecem por imposição da disponibilidade das tarefas, já que as tarefas não têm nenhuma dependência entre elas. Por fim o objetivo deste problema é descobrir quais e por que ordem cada tarefa é executada em cada máquina, tendo com critério de otimalidade a soma ponderada dos atrasos.

Usando a notação $\alpha|\beta|\gamma$ de Graham et al. (1979), o problema das máquinas paralelas não idênticas com tempos de preparação e datas de disponibilidade dos trabalhos e máquinas, é representado por:

$$R|a_k; r_j; s_{ij}|\sum w_j T_j$$

Na secção 4.1 temos uma explicação mais detalhada do problema e da notação usada.

Capítulo 3

Revisão da literatura

Neste capítulo apresentamos uma revisão da literatura para problemas de escalonamento, focada nos problemas com tempos de preparação. Inicialmente são apresentados modelos heurísticos seguidos de diversos modelos de programação inteira para problemas de máquinas paralelas. Por fim são revistos 3 artigos que usam algoritmos de partição e geração de colunas para o problema de máquinas paralelas.

3.1 Modelos Heurísticos

No artigo de Allahverdi et al. (2008) é feita a revisão da literatura para problemas de escalonamento com tempos de preparação. Para além da normal divisão dos problemas pelo seu tipo (máquina única, máquinas paralelas, *job shop*, *flow shop*, etc.), este artigo classifica os problemas pelos seus tempos de preparação serem dependentes ou independentes da sequência e pelo problema conter ou não lotes (*batch*). Assim, cada tipo de problema é subdividido em quatro grupos, tal como apresentado na figura 3.1. Nessa figura aparece realçado o grupo onde se insere o problema abordado nesta dissertação.

De uma maneira geral, quando os problemas de máquinas paralelas não contêm lotes e os tempos de preparação são independentes da sequência, faz mais sentido incluir esses tempos de preparação nos tempos de processamento. No entanto, existem problemas em que é necessário fazer essa distinção. Como exemplo, temos o caso dos problemas que permitem interrupção de tarefas ou problemas onde a preparação das tarefas nas

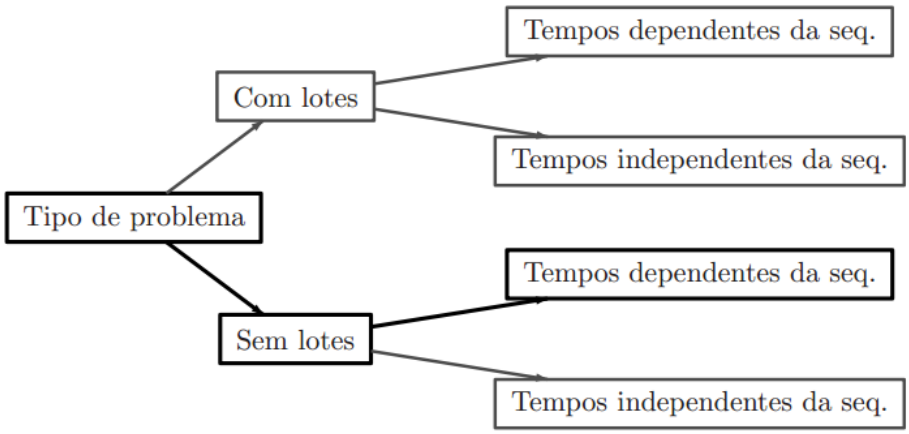


Figura 3.1: Classificação dos problemas de escalonamento por Allahverdi et al. (2008).

máquinas é feita por um ou mais servidores. Neste artigo são também revistos alguns artigos que abordam este tipo de problemas.

Para o problema das máquinas paralelas sem lotes onde os tempos de preparação são dependentes da sequência são apresentadas revisões de vários artigos para máquinas idênticas, uniformes e não relacionadas, com diversas funções objetivo. Nesta dissertação apenas serão referidas abordagens para o tipo de problema em estudo: máquinas paralelas não relacionadas.

Weng et al. (2001) Neste artigo a função objetivo considerada é a média ponderada dos tempos de conclusão das tarefas $\frac{\sum w_j C_j}{n}$. Aqui, são apresentadas sete heurísticas. De uma maneira geral as primeiras seis heurísticas seguem a mesma estrutura na construção das soluções. Basicamente essa estrutura segue dois passos. Primeiro as tarefas são ordenadas por uma determinada medida. Depois de ordenadas são afetadas às máquinas respeitando um determinado critério. Na sétima heurística a tarefa e a máquina são selecionadas de forma simultânea. Assim, para todas as tarefas ainda não escalonadas, são selecionadas, tanto a tarefa j como a máquina k que tenham o menor valor de rácio $(C_{v_k} + s_{v_k j} + p_{jk})/w_j$, onde v_k representa a última tarefa processada na máquina k e C_{v_k} representa o instante de conclusão da última tarefa processada na máquina k. Nos testes computacionais foram utilizadas instâncias com todas as combinações de $n = \{40, 60, 80, 100, 120\}$ tarefas com $m = \{4, 6, 8, 10, 12\}$ máquinas. Na análise dos resultados, não são mencionados tempos de

processamento pois os testes para cada instância demoram menos de um segundo e essa análise concluiu que a sétima heurística obteve melhores resultados que as restantes.

Cheng e Gen (1997) Neste artigo é aplicado um algoritmo genético híbrido ao problema das máquinas paralelas idênticas. Sendo máquinas idênticas, sabemos à partida que o tempo de processamento das tarefas são independentes das máquinas. Neste problema não são permitidas interrupções das tarefas, a data de entrega é comum para todas as tarefas e cada tarefa tem um peso/prioridade. Pretende-se neste artigo, minimizar o máximo desvio absoluto ponderado, ou seja, a função objetivo pretende minimizar: $max\{w_j|C_j - d|\}$. Nesta abordagem o algoritmo genético é complementado com uma heurística que obtém um ótimo local para cada filho gerado, antes de ser introduzido na população. A heurística usa o algoritmo que resolve este mesmo problema de forma ótima quando temos apenas uma máquina (Hall e Posner, 1991). Assim neste método híbrido o algoritmo genético controla a exploração global das soluções (população) enquanto a heurística explora localmente cada solução (cromossomas). Para além da aplicação do método híbrido, neste artigo também é descrito como é calculada a melhor data de entrega comum para todas as tarefas. Esta abordagem foi submetida a testes computacionais para problemas com 30 tarefas e 5 máquinas. Os resultados obtidos, demonstraram que esta abordagem obtêm melhores soluções para o problema, comparativamente com a heurística proposta por Li e Cheng (1994). Por outro lado os resultados do algoritmo genético (simples) foram piores comparativamente com esta abordagem híbrida.

Vallada e Ruiz (2011) Neste artigo é aplicado um algoritmo genético ao problema das máquinas paralelas não relacionadas com tempos de preparação. O critério de otimalidade usado é a minimização do instante de conclusão máximo (*makespan*), ou seja, minimizar C_{max}. O problema contém tempos de preparação que são dependentes tanto da sequência como da máquina, ou seja, o tempo de preparação para receber a tarefa j após ter executado i na máquina k é diferente do tempo de preparação para receber i após ter executado j na mesma máquina. Por sua vez, diferentes máquinas, k e k', têm também diferentes tempos de preparação. Assim $s_{kij} \neq s_{kji} \neq s_{k'ji} \neq s_{k'ij}$.

As instâncias de teste geradas são divididas em dois conjuntos, "*pequenas*" e "*grandes*". Instâncias pequenas contêm todas as combinações de $n = \{6, 8, 10, 12\}$ tarefas com $m = \{2, 3, 4, 5\}$ máquinas. Da mesma forma para as grandes instâncias temos $n = \{50, 100, 150, 200, 250\}$ e $m = \{10, 15, 20, 25, 30\}$. No artigo é apresentado um modelo de programação inteira mista (MIP) para o problema abordado, sendo uma adaptação do modelo proposto por Guinet (1993). Na subsecção 3.2.2 é detalhado este modelo. O algoritmo proposto neste artigo é comparado com outros métodos:

META - Heurística proposta por Rabadi et al. (2006);

MI - Heurística de múltipla inserção proposta por Kurz e Askin (2001);

GAK - Algoritmo genético proposto por Kurz e Askin (2001).

No conjunto de instâncias pequenas, é também testado o modelo de programação inteira mista (MPI). Os diversos métodos são comparados pelo desvio relativo percentual entre o melhor valor obtido entre os modelos e o valor do modelo. Nessa comparação os resultados mostraram que o algoritmo genético proposto obtêm melhores resultados que os restantes métodos.

3.2 Modelos de Programação Inteira

No artigo de Unlu e Mason (2010) é feita a avaliação de diferentes formulações de programação inteira para problemas de máquinas paralelas sem interrupções. Os autores dividem as diferentes formulações em quatro grupos de acordo com os diferentes tipos de variáveis. Na definição dos problemas são considerados os três tipos de máquinas (idênticas, uniformes e não relacionadas), datas de disponibilidade das tarefas e diferentes tipos de funções objetivo. No capítulo seguinte, serão apresentadas modificações a três destes modelos para contemplar as diferenças entre os problemas descritos nesta secção e o problema abordado nesta dissertação, tais como datas de disponibilidade das tarefas e máquinas, tempos de preparação e funções objetivo. Estes modelos são:

M1 Modelo de indexação do tempo - Neste modelo considera-se que o tempo é dividido em períodos $1, \ldots, l$, onde l é limitado pelo tempo de conclusão máximo das tarefas (*makespan*). Neste modelo, de uma forma geral, o que se pretende decidir é qual a máquina e em que período de tempo é que cada tarefa começa a ser processada.

M2 Modelo em rede - Para o problema das máquinas paralelas, num modelo em rede as tarefas são vistas como os nodos da rede e um caminho na rede representa o planeamento para uma máquina. Os nodos visitados nesse caminho traduzem a ordem pela quais as tarefas são processadas, os seja, a sequência para uma máquina. O objetivo neste modelo passa por "descobrir" os melhores caminhos para o conjunto das máquinas.

M3 Modelo de posicionamento - Este modelo assume que uma máquina tem no máximo ℓ posições para processar as tarefas, ou seja, pode receber no máximo ℓ tarefas. Com a posição das tarefas em cada máquina é possível definir a sequência das tarefas em cada máquina.

M4 Modelo de ordenação linear - Este modelo baseia-se essencialmente na utilização de duas variáveis binárias. Uma que define se uma tarefa é processada numa máquina e a segunda se uma tarefa i é processada antes de j, mas não imediatamente antes. Assim é possível definir em que máquina é que cada tarefa é processada e a ordem de processamento para cada máquina.

3.2.1 Modelo de indexação do tempo

Tal como foi referido, em Unlu e Mason (2010) é feita a avaliação de diferentes formulações de programação inteira. O modelo de indexação do tempo é um dos tipos de modelo que podemos encontrar. As variáveis de decisão são as seguintes:

- X_{kj}^{t} - Variável binária, que toma o valor 1 caso a tarefa j comece a ser executada no instante t na máquina k e 0 caso contrário;

- C_j - Instante de conclusão da tarefa j na máquina k;

- T_j - Atraso da tarefa j.

Neste modelo o valor de l limita t pelo valor do instante de conclusão máximo de todas as tarefas (makespan). O modelo apresentado é o seguinte:

$$Min \sum w_j T_j \tag{3.1}$$

$$sujeito\ a:$$

$$T_j \geq 0; \quad \forall j \in N \tag{3.2}$$

$$T_j \geq C_j - d_j; \quad \forall j \in N; \tag{3.3}$$

$$\sum_{k \in M} \sum_{t=0}^{l-1} X_{kj}^t = 1; \quad \forall j \in N; \tag{3.4}$$

$$\sum_{j \in N} \sum_{h=\max(0,t-p_{jk})}^{t-1} X_{kj}^h \leq 1; \quad \forall k \in M; t = 1, \ldots, l; \tag{3.5}$$

$$C_j \geq \sum_{k \in M} \sum_{t=0}^{l-1} (p_{jk} + t) X_{kj}^t; \quad \forall j \in N; \tag{3.6}$$

$$\sum_{k \in M} \sum_{t=0}^{r_j-1} X_{kj}^t = 0; \quad \forall j \in N; \tag{3.7}$$

$$C_j \geq 0, \quad \forall j \in N \tag{3.8}$$

$$X_{kj}^t \in \{0, 1\}; \quad \forall j \in N; \forall k \in M; \forall t = 1, \ldots, l$$

As restrições (3.2) e (3.3) que garantem que os atrasos são bem calculados, para cada uma das tarefas. O conjunto de restrições (3.4) garante que cada tarefa começa a ser executada numa máquina apenas num instante. Já o conjunto (3.5) garante que no máximo uma tarefa execute ao mesmo instante na mesma máquina, ou seja, se uma tarefa j começar a executar no instante t na máquina k então durante o tempo seguinte de execução p_{jk} a máquina k apenas executa j. Com a restrição (3.6) temos o tempo de conclusão de cada tarefa e por fim com a restrição (3.7) não permitimos que uma tarefa comece a sua execução até que esteja disponível.

3.2.2 Modelo em rede

Tal como foi referido, em Vallada e Ruiz (2011) é apresentado um modelo de rede que considera as seguintes variáveis de decisão:

- X_{ij}^k - Variável binária, que toma o valor 1 caso a tarefa i seja executada imediatamente antes de j, na máquina k e 0 caso contrário;
- C_j^k - Instante de conclusão da tarefa j na máquina k;
- C_{max} - Instante máximo de conclusão de todas as tarefas.

$$Min\, C_{max} \tag{3.9}$$

$$sujeito\, a:$$

$$\sum_{k\in M}\sum_{i\in\{0\}\cup\{N\}} X_{ij}^k = 1; \quad \forall j \in N; i \neq j \tag{3.10}$$

$$\sum_{k\in M}\sum_{j\in N} X_{ij}^k \leq 1; \quad \forall i \in N; i \neq j \tag{3.11}$$

$$\sum_{j\in N} X_{0j}^k \leq 1; \quad \forall k \in M \tag{3.12}$$

$$\sum_{h\in\{0\}\cup\{N\}} X_{hi}^k \geq X_{ij}^k; \quad \forall i,j \in N; h \neq i \neq j; \forall k \in M \tag{3.13}$$

$$C_j^k + V(1 - X_{ij}^k) \geq C_i^k + s_{ijk} + p_{jk}; \quad \forall i \in \{0\}\cup\{N\}; \forall j \in N; i \neq j; \forall k \in M \tag{3.14}$$

$$C_0^k = 0; \quad \forall k \in M \tag{3.15}$$

$$C_j^k \geq 0; \quad \forall k \in M; \forall j \in N \tag{3.16}$$

$$C_{max} \geq C_j^k; \quad \forall k \in M; \forall j \in N \tag{3.17}$$

$$C_{max} \geq 0$$

$$X_{ij}^k \in \{0,1\}; \quad \forall i \in \{0\}\cup\{N\}; \forall j \in N; i \neq j; \forall k \in M$$

O primeiro conjunto de restrições (3.10) garante que uma tarefa é afetada apenas a uma máquina e só tem um antecessor. O conjunto de restrições (3.11) garante que cada tarefa tem no máximo um sucessor. Esta restrição não é igual a 1 pois a última tarefa

de cada máquina não tem nenhum sucessor. É de realçar que quando $X_{0j}^k = 1$, j é a primeira tarefa a ser executada na máquina k. Assim o conjunto (3.12) limita que cada máquina tenha no máximo uma tarefa inicial. A garantia de que as sequências de tarefas em cada máquina estão corretas é dada pelo conjunto (3.13). O conjunto (3.14) controla os instantes de conclusão de cada tarefa. Quando numa máquina k, j é executado após i, então $X_{ij}^k = 1$ e o instante de conclusão de j (C_j^k) é superior ao instante de conclusão de i mais o tempo de preparação e de execução de j. Caso $X_{ij}^k = 0$, então o elevado valor da constante V torna a restrição redundante. O conjunto (3.15) define que as máquinas estão disponíveis no instante 0 e o conjunto (3.16) garante que o tempo de conclusão é não negativo. Por fim o conjunto de restrições (3.17) define o instante máximo de conclusão para ser minimizado na função objetivo.

3.2.3 Modelo de posicionamento

O modelo de posicionamento é mais um dos tipos de modelo que podemos encontrar em Unlu e Mason (2010). As variáveis de decisão são as seguintes:

$u_{j\ell}^k$ - Variável binária, que toma o valor 1 caso a tarefa j seja executada na posição ℓ na máquina k e 0 caso contrário;

y_ℓ^k - Instante de conclusão da tarefa, executadas na posição ℓ na máquina k;

C_j - Instante de conclusão da tarefa j na máquina k;

T_j - Atraso da tarefa j.

É de realçar que a variável y_ℓ^k não refere qual a tarefa que é executada em ℓ. A variável $u_{j\ell}^k$ é que nos dá essa informação. Para definir este modelo, é necessário fixar o número de posições que uma máquina tem disponíveis para processar, definido por ℓ. Se pensarmos que uma máquina pode no máximo executar todas as tarefas, então o número de posições máximo é o numero de tarefas. Assume-se assim que $\ell = 1, \ldots, n$, onde n é o número de tarefas, ou seja, todas as máquinas têm posições suficientes para processar todas as tarefas. Posto isto, para o problema das máquinas paralelas não relacionais, com instantes

de disponibilidade das tarefas, onde se pretende minimizar a soma ponderada dos atrasos, temos o seguinte modelo:

$$Min \sum w_j T_j \tag{3.18}$$

$$sujeito\ a:$$

$$T_j \geq 0; \quad \forall j \in N \tag{3.19}$$

$$T_j \geq C_j - d_j; \quad \forall j \in N; \tag{3.20}$$

$$\sum_{k \in M} \sum_{\ell \in N} u_{j\ell}^k = 1; \quad \forall j \in N; \tag{3.21}$$

$$\sum_{j \in N} u_{j\ell}^k \leq 1; \quad \forall \ell \in N; \forall k \in M; \tag{3.22}$$

$$y_1^k \geq \sum_{j \in N} p_{jk} u_{j1}^k; \quad \forall k \in M; \tag{3.23}$$

$$y_\ell^k \geq y_{\ell-1}^k + \sum_{j \in N} p_{jk} u_{j\ell}^k; \quad \forall k \in M; \forall \ell = 2, \ldots, N \tag{3.24}$$

$$C_j \geq y_\ell^k - V(1 - u_{j\ell}^k); \quad \forall j \in N; \forall \ell \in N; \forall k \in M; \tag{3.25}$$

$$y_\ell^k \geq \sum_{j \in N} (p_{jk} + r_j) u_{j\ell}^k; \quad \forall k \in M; \forall \ell \in N \tag{3.26}$$

$$C_j \geq 0; \quad \forall j \in N$$

$$y_\ell^k \geq 0; \quad \forall k \in M; \forall \ell \in N$$

$$u_{j\ell}^k \in \{0, 1\}; \quad \forall j \in N; \forall k \in M; \forall \ell \in N$$

Mais uma vez temos as restrições (3.19) e (3.20) que garantem que os atrasos são bem calculados, para cada uma das tarefas. O conjunto de restrições (3.21) garante que cada tarefa é executada numa posição numa máquina. Por sua vez o conjunto (3.22) garante que cada posição em cada máquina tem no máximo uma tarefa. As restrições (3.23) e (3.24) garantem que as variáveis y_ℓ^k tomam o valor do instante de conclusão da tarefa que executar na posição ℓ. A partir desse valor é possível calcular esse instante para cada tarefa (3.25). Por fim, a restrição (3.26) não permite que nenhuma tarefa seja executada sem estar disponível.

3.2.4 Modelo de ordenação linear

A formulação que apresentamos de seguida provém do artigo de Zhu e Heady (2000). É de realçar que esta formulação apenas é válida para tempos de preparação onde se verifique a desigualdade triangular. Assim, em nenhuma máquina k, em nenhum grupo de três tarefas i, h, j, se pode verificar que s_{ij} seja maior que $s_{ih} + p_{hk} + s_{hj}$. Na figura 3.2 podemos ver um esquema que exemplifica essa situação.

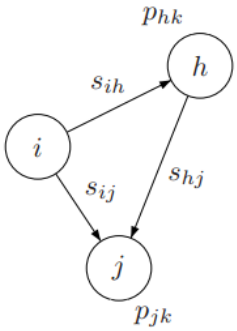


Figura 3.2: Desigualdade triangular.

Sendo i a última tarefa processada numa máquina k e $s_{ih} = 2, s_{hj} = 2, s_{ij} = 10, p_{hk} = 4$ verifica-se que é mais vantajoso processar h seguido de j, do que processar apenas j já que:

(i,h,j): $s_{ih} + p_{hk} + s_{hj} = 8$;

(i,j): $s_{ij} = 10$.

Pelo exemplo anterior podemos ver um caso que não respeita a restrição. Por fim passamos a explicação do modelo que usa a seguinte notação:

- V - Constante de grande valor;
- Y_{ij} - Variável binária, que toma o valor 1 caso a tarefa i seja executada antes de j e 0 caso contrário. De referir que a tarefa i pode não ser executada imediatamente antes de j e não é mencionada a máquina onde as tarefas são executadas. Por outro lado é reconhecido que $Y_{ij} + Y_{ji} = 1$;
- Z_j^k - Variável binária, que toma o valor 1 caso a tarefa j seja executada na máquina k e 0 caso contrário;

- C_j - Instante de conclusão da tarefa j;
- w_j - Peso do atraso da tarefa j;
- w'_j - Peso do avanço da tarefa j.

$$Min \sum_{j \in N} (w_j T_j + w'_j E_j) \tag{3.27}$$

$$sujeito\ a:$$

$$C_j - T_j + E_j = d_j; \quad \forall j \in N \tag{3.28}$$

$$\sum_{k \in M} Z_j^k = 1; \quad \forall j \in N \tag{3.29}$$

$$C_j + V(1 - Y_{ij}) + V(1 - Z_j^k) + V(1 - Z_i^k) \geq C_i + s_{ij} + p_{jk};$$
$$\forall k \in M; \forall i \in N; j \neq i; j = i + 1, \ldots, N \tag{3.30}$$

$$C_i + VY_{ij} + V(1 - Z_j^k) + V(1 - Z_i^k) \geq C_j + s_{ji} + p_{ik};$$
$$\forall k \in M; \forall i \in \{0\} \cup \{N\}; j \neq i; j = i + 1, \ldots, N \tag{3.31}$$

$$\sum_{i \in N} Z_i^k \geq 1; \quad \forall k \in M \tag{3.32}$$

$$C_j \geq 0; \quad \forall j \in N$$

$$T_j \geq 0; \quad \forall j \in N$$

$$E_j \geq 0; \quad \forall j \in N$$

$$Z_j^k \in \{0, 1\}; \quad \forall j \in N; \forall k \in M$$

$$Y_{ij} \in \{0, 1\}; \quad \forall i, j \in N$$

O primeiro conjunto de restrições (3.28) define se uma tarefa é executada com avanço ou atraso. O segundo conjunto de restrições (3.29) assegura que cada tarefa é executada apenas numa máquina. Quando duas tarefas são executadas na mesma máquina, então, ou tarefa j é executada após a tarefa i, ou o contrário, i é executada após j. No primeiro caso C_j tem de ser superior a $C_i + s_{ij} + p_{pk}$, já segundo C_i tem de ser superior a $C_j + s_{ji} + p_{ik}$. Assim, quando acontece o primeiro caso, o conjunto (3.30) garante essa restrição e (3.31)

torna-se redundante pelo valor da constante V. Quando acontece o segundo caso verifica-se o contrário, (3.30) torna-se redundante. Quando as duas tarefas não são executadas na mesma máquina então o ou os valores ve V tornam os dois conjuntos de restrições, (3.30) e (3.31), redundantes. É de realçar que nesta modelação são utilizadas tarefas fictícias que representam a tarefa executadas inicialmente, representadas com índice 0. Assim $C_0 = 0$ e $Z_0^k = 1$. Por último, o conjunto (3.32) "obriga" a que todas as máquinas sejam usadas, o que reduz o número de soluções ótimas que têm de ser visitadas. Esta restrição só faz sentido quando o número de tarefas é superior ao número de máquinas e quando se assume que a solução ótima passa pela utilização de todas as máquinas, o que normalmente acontece em problemas reais. Neste artigo foram testadas 6 replicações de cada instância com todas as combinações de $n = \{5, 6, 7, 8, 9\}$ tarefas com $m = \{1, 2, 3\}$ máquinas. Os testes foram executados no *Hyper Lindo software* e num 66 MHz, 486 PC. O tempo médio máximo foi de 5397.33 segundos para o problema com 3 máquinas e 9 tarefas.

3.2.5 Comparação entre modelos

Tal como referido na secção 3.2, no artigo de Unlu e Mason (2010) é feita a avaliação das quatro formulações referidas nessa secção. Iremos, tal como na secção 3.2, utilizar M1, M2, M3 e M4 para nos referir ao Modelo de indexação do tempo, Modelo em rede, Modelo de posicionamento e Modelo de ordenação linear, respetivamente. Nos testes computacionais foram testados problemas de máquinas paralelas idênticas com $m = \{2, 3, 5, 10, 15\}$ máquinas e $n = \{20, 50, 100\}$ tarefas, com e sem instantes de disponibilidade das tarefas, r_j .Em relação aos critérios de otimização foram testados dois: $\sum w_j C_j$ e C_{max}.

De forma geral, o modelo M1 obtém as soluções de melhor qualidade em menor tempo. Tal pode explicar-se pelos melhores limites inferiores fornecidos pela relaxação linear deste modelo. No entanto, este modelo é muito sensível ao aumento dos tempos de processamento das tarefas. Quando esses tempos são elevados, o modelo M2 tem o melhor comportamento, o que se pode justificar pelos limites inferiores dados pela sua relaxação linear serem os segundos melhores. Os modelos M3 e M4 têm um comportamento claramente

inferior.

3.3 Partição e avaliação

Nesta secção iremos rever três artigos onde são abordados algoritmos exatos na resolução de problemas de máquinas paralelas.

Lopes e Valério de Carvalho (2007) Neste artigo os autores aplicam um algoritmo de partição e avaliação ao problema de máquinas paralelas não idênticas com tempos de preparação dependentes da sequência, datas de disponibilidade das tarefas e máquinas. O critério de otimização a minimizar foi a soma ponderada dos atrasos. Inicialmente os autores apresentam uma primeira formulação onde a função custo é não- linear. Para obter uma formulação de programação inteira equivalente, aplicam o método de decomposição de Dantzig-Wolfe, onde cada variável de decisão (coluna) representa uma sequência para uma máquina. O modelo de decomposição apresentado na secção 5.1 é baseado no modelo apresentado neste trabalho. A relaxação linear desta formulação é resolvida pelo método de geração de colunas. As colunas são obtidas pela resolução dos subproblemas. Com a resolução dos subproblemas obtém-se novas sequências para as diferentes máquinas. O algoritmo utilizado na resolução dos subproblemas é baseado em programação dinâmica. Este algoritmo é também descrito na secção 5.5. A solução da relaxação linear obtida é usada num algoritmo de partição e avaliação para resolver a formulação de programação inteira. Em termos de testes computacionais são apresentados resultados para instâncias com até 50 máquinas e 150 tarefas.

Akker et al. (1999) Neste artigo é aplicado um algoritmo de geração de colunas ao problema das máquinas paralelas idênticas com o objetivo de minimizar a soma ponderada dos tempos de conclusão das tarefas. Os autores referem que a principal dificuldade do problema é a afetação das tarefas pelas máquinas uma vez que para uma determinada afetação é possível definir-se a sequência ótima com uma regra de prioridade. O algoritmo de geração de colunas usado passa por formular o problema como um problema de pro-

gramação inteira. Nesse algoritmo a relaxação linear pode ser resolvida de forma eficiente pela geração de colunas uma vez que o problema de partição de conjuntos pode ser resolvido em tempo pseudo-polinomial. Quando a solução resultante da resolução da relaxação linear não for inteira ou não puder ser convertida para inteira, uma vez que em certas condições identificadas no trabalho isso era possível, tem que se aplicar um esquema de partição para obter a solução ótima. Inicialmente o algoritmo é testado para $m = \{3, 4, 5\}$ máquinas e $n = \{20, 30, 40, 50\}$ tarefas. Depois testam o algoritmo com instâncias maiores com até 100 tarefas e 10 máquinas. Os resultados mostram que quanto menor é o rácio entre tarefas e máquinas mais fácil é a instância. Isso acontece pois, quantas mais tarefas são, em média, afetadas por máquina mais difícil se torna o problema.

Chen e Powell (1999) Neste artigo os autores propõem um procedimento de partição e avaliação para a resolução de problemas de máquinas paralelas. 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. Quando a solução obtida na relaxação linear não é inteira é aplicado um esquema de partição para obter a solução ótima. A metodologia proposta por Chen e Powell (1999) tem como grande vantagem o facto de ser genérica. Os autores afirmam que é possível aplicar a metodologia a quaisquer problemas do mesmo género, tanto para máquinas idênticas, como para máquinas uniformes e máquinas não idênticas, desde que as funções objetivo sejam aditivas. Neste artigo são apresentados resultados da aplicação desta metodologia para problemas de máquinas paralelas idênticas, uniformes e não idênticas. Para cada um dos três tipos de máquinas são testados dois tipos de funções objetivo: soma ponderada dos tempos de conclusão das tarefas e soma ponderada do número de tarefas em atraso. Os resultados mostram que conseguem resolver em tempos aceitáveis instâncias com 100 tarefas e 10 processadores.

Os autores referem que uma linha de investigação a seguir é utilizar heurísticas na resolução do subproblema, abordagem que é usada neste trabalho.

Capítulo 4

Definição do problema e modelos de programação inteira

Neste capítulo iremos inicialmente, ver quais as restrições que caraterizam o problema de máquinas paralelas não idênticas, abordado nesta dissertação. Para este mesmo problema iremos também apresentar diferentes modelos de programação inteira baseados em modelos revistos na literatura.

4.1 Definição do problema

Na definição do problema de máquinas paralelas não idênticas temos que:

- Cada tarefa é processada uma vez numa máquina e cada máquina só processa uma tarefa de cada vez;
- Cada tarefa contém apenas um trabalho ou operação;
- A execução de uma tarefa numa máquina não pode ser interrompida;
- As máquinas podem executar o mesmo tipo de funções (máquinas paralelas), ou seja, todas as máquinas podem executar todas as tarefas;
- O tempo de execução para cada tarefa depende da máquina que a executa (não

idênticas). Assim p_{jk} representa o tempo de processamento da tarefa j na máquina k;

- O Tempo de preparação (setup) das máquinas é dependente da sequência (s_{ij}), onde i representa a tarefa anteriormente executada e j a tarefa que se vai executar. Neste problema não faz sentido executar a mesma tarefa duas vezes seguidas, assim não exite tempo de preparação quando $i = j$, ou seja, s_{ii} não existe.

Em relação às tarefas temos ainda a seguinte informação:

- r_j - Instante de tempo que a tarefa fica disponível a ser processada (release date);
- d_j - Instante de tempo que a tarefa deve ficar concluída (due date);
- w_j - Peso/prioridade associado a uma tarefa.

Em relação às máquinas temos ainda a seguinte informação:

- a_k - Instante de tempo que a máquina fica disponível para processar tarefas ou ser preparada para receber uma tarefa, em que k representa a máquina;
- l_k - Tarefa que a máquina k está preparada para receber logo que fica disponível. Quando $l_k = j$ a máquina já está preparada para processar a tarefa j, assim não tem tempo de preparação ou considera-se igual a zero. Quando $l_k \neq j$, a máquina tem de ser preparada para receber j, ou seja, tem um tempo de preparação de $s_{(l_k)j}$.

Sempre que uma tarefa é concluída após a sua data de entrega, sofre um atraso. Para a medida de desempenho ou função objetivo, é usada a soma das penalizações, $\sum w_j T_j$.

Normalmente, nos problemas das máquinas paralelas, quando passamos de problemas do tipo $P|\beta|\gamma$ (máquinas idênticas) para problemas do tipo $Q|\beta|\gamma$ (máquinas uniformes), a complexidade aumenta. Da mesma forma quando passamos para problemas do tipo $R|\beta|\gamma$ (máquinas não idênticas). Isto acontece porque os problemas do tipo $P|\beta|\gamma$ são um caso particular de $Q|\beta|\gamma$. E, da mesma forma, que os problemas do tipo $Q|\beta|\gamma$ são um caso particular de $R|\beta|\gamma$. Visto isto, podemos dizer que $R|a_k; r_j; s_{ij}|\sum w_j T_j$ é uma generalização do problema $P||\sum w_j T_j$. Sendo este último um problema NP-difícil, até quando temos

apenas uma máquina (Lenstra et al., 1977), o primeiro problema (máquinas não idênticas) também é NP-difícil, e foi resolvido de forma exata pela primeira vez por Lopes e Valério de Carvalho (2007).

4.2 Modelo em rede

O modelo de programação inteira apresentado por Vallada e Ruiz (2011), foi revisto na subsecção 3.2.2. Este modelo foi adaptado para criar uma formulação para o problema abordado nesta dissertação. Os dois problemas têm algumas diferenças. No problema abordado nesta dissertação os tempos de preparação são apenas dependentes da sequência, a função objetivo é soma ponderada dos atrasos, temos data de entrega das tarefas e temos também datas de disponibilidade tanto para as máquinas como para as tarefas.

Os conjuntos de restrições (4.1) e (4.2) garantem que o atraso é definido de forma correta.

$$T_j \geq 0; \quad \forall j \in N \tag{4.1}$$

$$T_j \geq C_j^k - d_j; \quad \forall j \in N; \forall k \in M \tag{4.2}$$

Com o valor dos atrasos para cada tarefa, em (4.3) temos a função objetivo.

$$Min \sum w_j T_j \tag{4.3}$$

Para representar o instante em que a máquina fica disponível alteramos o instante de conclusão da tarefa fictícia 0 para cada máquina (4.4). Por fim como temos datas de disponibilidade das tarefas, estas só podem iniciar a sua execução depois de estarem disponíveis (r_j) e para garantir essa restrição foi adicionado o conjunto (4.5).

$$C_0^k = a_k; \quad \forall k \in M \tag{4.4}$$

$$C_j^k + V(1 - X_{ij}^k) \geq r_j + p_{jk}; \quad \forall i \in \{0\} \cup \{N\}; \forall j \in N; i \neq j; \forall k \in M \tag{4.5}$$

Assim com a função objetivo (4.3) e com os conjuntos de restrições (4.1), (4.2), (3.10),

(3.11), (3.12), (3.13), (3.14), (4.5), (4.4) e (3.16) temos uma formulação para o problema abordado nesta dissertação.

4.3 Modelo de posicionamento

O modelo de Unlu e Mason (2010) apresentado na subsecção 3.2.3, foi formulado para um problema onde não eram contemplados tempos de preparação nem tempos de disponibilidade das máquinas, duas diferenças para o problema abordado nesta dissertação. Assim é possível adicionar restrições ao modelo original de forma a contemplar essas diferenças.

$$y_1^k \geq a_k + \sum_{j \in N} (s_{l_k j} + p_{jk}) u_{j1}^k; \quad \forall k \in M; \tag{4.6}$$

$$y_\ell^k \geq y_{\ell-1}^k + \sum_{i \in N} s_{ij} u_{i(\ell-1)}^k + \sum_{j \in N} p_{jk} u_{j\ell}^k; \quad \forall k \in M; \forall \ell = 2, \ldots, N \tag{4.7}$$

$$\sum_{i \in N} u_{i\ell}^k \geq \sum_{j \in N} u_{j(\ell-1)}^k; \quad \forall k \in M; \forall \ell = 1, \ldots, N-1; \tag{4.8}$$

Em relação ao instante de disponibilidade das máquinas temos a restrição (4.6) que garante que se na primeira posição duma máquina é executada uma tarefa, então essa máquina está disponível. Por sua vez, a restrição (3.24) é modificada, e em (4.7) passa-se a considerar os tempo de preparação. Sabe-se à partida que as posições não vão ser todas ocupadas, e uma vez que com os tempos de preparação a ordem é importante, a restrição (4.8) garante que se uma posição ℓ é ocupada então a posição anterior também o é.

4.4 Modelo de ordenação linear

O modelo apresentado em Zhu e Heady (2000) foi também revisto na subsecção 3.2.4. Como foi referido esta formulação apenas faz sentido para valores de preparação razoáveis e é assumido que na solução ótima são usadas todas as máquinas. Partindo do princípio que estes pressupostos são válidos para o problema em estudo nesta dissertação, o modelo pode ser adaptado para ser construída uma formulação válida. O problema tratado em

Zhu e Heady (2000) não considerava instantes de disponibilidade para as máquinas e para as tarefas. Assim o conjunto (4.9) garante que uma tarefa só começa a ser executada depois de estar disponível.

$$C_j \geq r_j + p_{jk} Z_j^k; \quad \forall k \in M; \forall j \in N \tag{4.9}$$

Por fim com o conjunto de restrições (4.10) garantimos que nenhuma máquina começa a processar antes de estar disponível.

$$C_j + V(1 - Z_j^k) \geq a_k + s_{l_k j} + p_{jk}; \quad \forall k \in M; \forall j \in N \tag{4.10}$$

Posto isto a formulação tem a função objetivo (4.3) e os conjuntos de restrições (4.1), (4.2), (3.29), (3.30), (3.31), (4.9), (4.10) e (3.32).

Capítulo 5

Modelo de decomposição e resolução da relaxação linear

Neste capítulo iremos discutir de que forma é obtido o valor ótimo da relaxação linear usando o método de geração de colunas. Inicialmente apresentamos o modelo de decomposição. De seguida introduzimos o método de geração de colunas, apresentamos a heurística que cria as colunas iniciais e os algoritmos que resolvem o subproblema. Detalhamos ainda as políticas de inserção de colunas. No final do capítulo apresentamos um exemplo que tenta demonstrar as etapas descritas.

5.1 Modelo de decomposição

O modelo que se apresenta nesta secção, baseia-se na enumeração de todas as sequências de afetação possíveis para todas as máquinas. Para a definição do modelo de decomposição, considera-se os seguintes conjuntos, parâmetros e variável de decisão:

Conjuntos:

- T - Conjunto de tarefas;
- M - Conjunto de máquinas;
- P^k - Conjunto de sequências para a máquina k.

Parâmetros:

- v_{jp}^k - Número de vezes que a tarefa j é executada na sequência p da máquina k;
- c_p^k - Custo (soma ponderada dos atrasos) da sequência p da máquina k.

Variável de decisão:

- y_p^k - Variável de decisão para a sequência p na máquina k. Será 1 caso a sequência faça parte da solução, 0 caso contrário.

$$Min \sum_{k \in M} \sum_{p \in P^k} c_p^k y_p^k \tag{5.1}$$

$$sujeito\ a:$$

$$\sum_{p \in P^k} y_p^k \leq 1; \forall k \in M \tag{5.2}$$

$$\sum_{k \in M} \sum_{p \in P^k} v_{jp}^k y_p^k \geq 1; \forall j \in T \tag{5.3}$$

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

No modelo de decomposição ou problema mestre cada coluna representa uma sequência para uma máquina. As restrições (5.2) garantem que uma máquina não é utilizada mais do que uma vez numa solução. As restrições (5.3) garantem que cada tarefa é executada pelo menos uma vez. O mais natural para esta restrição seria admitir a restrição do tipo "= 1", ou seja, executar cada tarefa apenas uma vez. Mas, como iremos ver mais à frente, o algoritmo que resolve o subproblema pode admitir que uma tarefa seja executada mais de uma vez. Por fim c_p^k representa o custo associado à sequência p, que é calculado pela soma ponderada dos atrasos na máquina k. O objetivo (5.1) passa pela decisão de quais as sequências devem ser selecionadas, que satisfazendo as restrições, minimizam o somatório dos custos. É de realçar que a ordem em que uma tarefa é executada é importante e uma vez que as colunas não têm nenhuma representação da ordem em que são feitas as tarefas (sequência), podemos ter colunas aparentemente iguais, mas não o são pois o custo tem

de ser diferente.

5.2 Exemplo

		a_k	l_k
k	*1*	0	1
	2	0	2

(a) Informação das máquinas

		r_j	d_j	w_j
	1	0	35	15
j	*2*	0	30	10
	3	0	35	15

(b) Informação das tarefas

		j		
s_{ij}		*1*	*2*	*3*
	1	-	4	8
i	*2*	10	-	5
	3	5	5	-

(c) Tempos de preparação (setup)

		k	
p_{jk}		*1*	*2*
	1	20	30
j	*2*	25	15
	3	20	25

(d) Tempos de processamento das tarefas

Tabela 5.1: Dados do exemplo com 3 tarefas e 2 máquinas.

Nesta secção apresentaremos um exemplo com 3 tarefas e 2 máquinas onde temos detalhada toda informação dividida pelo conjunto de tabelas 5.1. Na tabela 5.1a temos a informação que apenas diz respeito às máquinas. Nessa tabela, sendo k o índice da máquina, temos as datas de disponibilidade (*release dates*) - a_k e sua configuração inicial - l_k, ou seja, a tarefa que a máquina está previamente preparada para receber. Neste caso a máquina 1 está configurada para processar a tarefa 1.

Por sua vez, na tabela 5.1b temos a informação referente às tarefas. Mais uma vez a primeira coluna é índice mas desta vez das tarefas, as restantes colunas contêm respetivamente, a seguinte informação:

- Data de disponibilidade das tarefas - r_j
- Data de entrega (*due dates*) das tarefas - d_j
- Prioridades das tarefas - w_j

As tabelas 5.1c e 5.1d têm uma estrutura muito semelhante, pois ambas contêm informação de tempos e nestas, a primeira coluna e linha são os índices. Na tabela 5.1c temos o instante que demora a preparar para receber a tarefa j depois de executada a tarefa i. Neste exemplo, o tempo de preparação para receber a tarefa 2 depois de executada a tarefa 1 é 4. Do mesmo modo na tabela 5.1d temos os tempos de processar a tarefa j na máquina k.

Máquina (k)	Índice da sequência (p)	Variável de decisão associada (y_p^k)	Sequência	Custo (c_p^k)
1	0	y_0^1	()	0
1	1	y_1^1	(1)	0
1	2	y_2^1	(2)	0
1	3	y_3^1	(3)	0
1	4	y_4^1	(1,2)	190
1	5	y_5^1	(1,3)	195
1	6	y_6^1	(2,1)	360
1	7	y_7^1	(2,3)	285
1	8	y_8^1	(3,1)	270
1	9	y_9^1	(3,2)	280
1	10	y_{10}^1	(1,2,3)	625
1	11	y_{11}^1	(1,3,2)	675
1	12	y_{12}^1	(2,1,3)	1140
1	13	y_{13}^1	(2,3,1)	945
1	14	y_{14}^1	(3,1,2)	790
1	15	y_{15}^1	(3,2,1)	1075
2	1	y_1^2	(1)	0
		...		
2	15	y_{15}^2	(3,2,1)	1025

Tabela 5.2: Algumas sequências da formulação completa.

Na tabela 5.2 são apresentados algumas sequências que permite a definição do modelo de decomposição para o exemplo. Nas primeiras 16 linhas da tabela 5.2 temos para a máquina 1 todas as sequências válidas, incluindo a sequência vazia. Apesar da tabela 5.2 não conter todos as sequências, para a máquina 2 teríamos exatamente as mesmas que temos para a máquina 1. Para um problema com 3 tarefas e 2 máquinas, todas as

combinações possíveis fazem um total de 32 colunas que representam soluções válidas, ou seja, onde uma tarefa não é executada mais de uma vez. Na última coluna temos detalhado o custo da sequência que corresponde à soma ponderada dos atrasos, $\sum w_j T_j$. Podemos ver que o custo da sequência com índice 4 na máquina 1 é de 190. Este custo é calculado da seguinte forma:

$T_1 = max(0, (max(a_1 + s_{l_1 1}, r_1) + p_{11}) - d_1) = max(0, (max(0 + 0, 0) + 20) - 35) = 0$
$T_2 = max(0, (max(C_1 + s_{12}, r_2) + p_{21}) - d_2) = max(0, (max(20 + 4, 0) + 25) - 30) = 19$
$c_4^1 = w_1 * T_1 + w_2 * T_2 = 15 * 0 + 10 * 19 = 190$

Posto isto, com a informação da tabela anterior podemos agora formular o modelo para o exemplo dado:

$$Min \quad 0y_0^1 + 0y_1^1 + 0y_2^1 + 0y_3^1 + 190y_4^1 + 195y_5^1 + 360y_6^1 + 285y_7^1 + 270y_8^1 + 280y_9^1 + 625y_{10}^1 + 675y_{11}^1 + 1140y_{12}^1 + 945y_{13}^1 + 790y_{14}^1 + 1075y_{15}^1 + 0y_1^2 + \cdots + 1025y_{15}^2;$$

$$sujeito\ a:$$
$$y_0^1 + y_1^1 + y_2^1 + y_3^1 + y_4^1 + y_5^1 + y_6^1 + y_7^1 + y_8^1 + y_9^1 + y_{10}^1 + y_{11}^1 + y_{12}^1 + y_{13}^1 + y_{14}^1 + y_{15}^1 \leq 1;$$
$$y_1^2 + \cdots + y_{15}^2 \leq 1;$$

$$y_1^1 + y_4^1 + y_5^1 + y_6^1 + \cdots + y_{15}^1 + y_1^2 + \cdots + y_{15}^2 \geq 1;$$
$$y_2^1 + y_4^1 + y_6^1 + y_7^1 + y_9^1 + \cdots + y_{14}^1 + \cdots + y_{15}^2 \geq 1;$$
$$y_3^1 + y_5^1 + y_7^1 + y_8^1 + y_9^1 + \cdots + y_{15}^1 + \cdots + y_{15}^2 \geq 1;$$

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

Neste exemplo, a primeira linha corresponde à função objetivo que se pretende minimizar. É de realçar que algumas variáveis, como por exemplo y_1^1 têm coeficiente 0, o que significa que o seu custo é nulo. A segunda e terceira linha correspondem ao conjunto de restrições (5.2), para a primeira e segunda máquina, respetivamente. Para o segundo conjunto de restrições (5.3) temos as últimas três linhas, uma para cada uma das tarefas. Para este problema os métodos de enumeração completa não são viáveis, pois à medida

que o tamanho das instancias do problema aumenta o número de colunas aumenta exponencialmente pelo fator $((n!\,(n-1))+n)m$, onde n o número de tarefas e m o número de máquinas.

5.3 Geração de colunas

Na secção 5.1 vimos um modelo de decomposição para o problema das máquinas paralelas. Vimos também que é impraticável a enumeração de todas as vaiáveis para grandes instâncias. O método de geração de colunas permite assim que se obtenha a solução ótima da relaxação linear fazendo uso da enumeração implícita das variáveis. Quando é usado o método de geração de colunas é usual designar-se o modelo de decomposição por problema mestre. Na geração de colunas em cada iteração consideram-se apenas algumas colunas do problema mestre, logo, considera-se um problema mestre restrito, que passa a conter apenas algumas variáveis.

Este método pode ser descrito pelos passos:

- **Passo 1**- Iniciar o problema mestre restrito com colunas que representam uma solução válida;
- **Passo 2**- Obter a solução ótima para o problema mestre restrito;
- **Passo 3**- Com a informação obtida com o problema mestre restrito (duais) resolver subproblemas;
- **Passo 4**- Caso as variáveis, associadas às soluções dos subproblemas, sejam atrativas inserir uma ou mais colunas no problema mestre restrito, dependendo da política de inserção de colunas e retornar ao passo 2;
  Caso contrário Terminar.

No método utilizado, o problema mestre restrito é iniciado com colunas geradas por uma heurística, que representa uma solução válida para o problema. Cada coluna representa uma sequência, para uma máquina. A estrutura é a mesma do modelo de decomposição (problema mestre). A resolução do problema mestre restrito permite obter

a informação do valor das duais de cada restrição. Os valores das duais das restrições relativas às tarefas serão usados na resolução dos subproblemas para avaliar quais as soluções mais vantajosas para serem inseridas no problema mestre restrito. Quando os subproblemas não conseguirem gerar soluções atrativas para nenhuma máquina, o algoritmo termina, e obtemos o ótimo da relaxação linear.

5.4 Heurística inicial

Esta heurística, utilizada em Lopes (2004), tem como objetivo gerar colunas válidas para iniciar o problema mestre restrito com uma solução válida. Inicialmente é criada uma lista ordenada de forma crescente pelo rácio ($\frac{d_j}{w_j}$), onde j representa a tarefa. Duma maneira geral as primeiras tarefas da lista têm datas de entrega curtas e/ou prioridades altas.

Pela ordem que aparecem na lista ordenada, cada tarefa é afetada na máquina com a menor soma ponderada dos atrasos. Quando duas ou mais máquinas tiverem o mesmo valor da soma ponderada dos atrasos, é selecionada aquela com o menor tempo de conclusão, isto é, o menor tempo de preparação mais o tempo de processamento ($s_{ij} + p_{jk}$), onde i é a última tarefa na máquina k. A tarefa fica então afetada na máquina selecionada e a seguinte tarefa da lista é considerada. A heurística termina quando todas as tarefas estiverem afetadas a uma máquina.

A complexidade desta heurística é no pior caso $O(n^2 + nm)$, uma vez que são ordenadas n tarefas ($n * n$) e afetadas n tarefas em m máquinas. Isto para uma implementação trivial pois existem algoritmos de ordenação com complexidade $\Theta(n \log n)$. Na subsecção 5.7.1 temos a demostração do modo de funcionamento para um exemplo.

5.5 Algoritmos do subproblema

O algoritmo do subproblema é resolvido para uma máquina de cada vez. Este recebe em cada iteração a informação do valor das duais para cada tarefa α_j, onde j representa a

tarefa. Na resolução do problema mestre, cada restrição tem associado um valor dual. Na secção 5.1 vimos que no modelo de decomposição, temos uma restrição para cada uma das tarefas j, representadas pelo conjunto de restrições 5.3. Na resolução do problema mestre restrito, cada uma destas restrições tem associada o valor da dual α_j, que é utilizado no algoritmo. O algoritmo decide para uma máquina, que tarefas são executadas e por que ordem, com o objetivo de minimizar o custo reduzido. O custo reduzido é calculado pela soma das duais das tarefas selecionadas com os respetivos atrasos ponderados. A variável x_{jk}, representa o número de vezes que a tarefa j é executada na solução do algoritmo, onde, como vimos k é fixo. Assim sendo, o objetivo do algoritmo passa por:

$$\min \sum_j ((w_j T_j) x_{jk} - \alpha_j x_{jk}) \tag{5.4}$$

Nesta secção iremos apresentar três algoritmos que resolvem o subproblema. Na construção do primeiro modelo usamos o algoritmo exato de programação dinâmica descrito em Lopes (2004, p.80). Este resolve o subproblema gerando soluções cíclicas sem ciclos de ordem 1 e 2 e obtêm a solução ótima. São também apresentadas duas heurísticas, baseadas nesse algoritmo de programação dinâmica. Nessas heurísticas o espaço de soluções válidas é reduzido, não garantindo a solução ótima do subproblema. Por fim é explicado como o subproblema poderia ser resolvido por um modelo de programação inteira.

5.5.1 Modelo exato

Este algoritmo de programação dinâmica determina a sequência para uma máquina, minimizando o custo reduzido. Para explicação mais detalhada, formulação e equações de recorrência do modelo, ver Lopes (2004). Inicialmente o subproblema recebe, em cada chamada, o valor da dual de cada tarefa α_j. Cada dual é encarada como um *"prémio"* que se recebe sempre que a tarefa é selecionada para entrar na rede de programação dinâmica. Este algoritmo para além das sequências admissíveis, representa também sequências não admissíveis, ou seja, sequências em que uma tarefa é executada mais de uma vez. Com o intuito de reduzir o número de sequências admitidas são introduzidas algumas restrições

ao modelo de forma a melhorar significativamente o desempenho. Assim, são eliminadas as sequências com ciclos de ordem 1 e 2, que são do tipo (i,i) e (i,j,i), respetivamente. Para melhor entender quais são as sequências excluídas, aqui ficam dois exemplos:

- Ordem 1: Executar as tarefas $(1,2,2)$;
- Ordem 2: Executar as tarefas $(2,1,2)$.

No algoritmo de programação dinâmica cada estado representa uma tarefa. Para se introduzir um estado na rede de programação dinâmica, são avaliadas todas as sequências possíveis para lá chegar. Para esse mesmo estado, as duas melhores sequências são registadas. Para esse efeito a estrutura usada guarda o estado anterior de cada uma das sequências. Assim, cada estado tem referência para dois estados anteriores, garantindo que existem duas alternativas para lá chegar, o que garante também, que conseguimos construir uma sequência onde não temos ciclos de ordem 2. Assim para cada estado, são registadas duas sequências de afetação: $H^k(j,t)$ a melhor sequência e $H_2^k(j,t)$ a segunda melhor, onde k representa a máquina e t o instante de tempo que a tarefa j é concluída. Para não incluir ciclos de ordem 1, quando estamos a avaliar a introdução de um estado numa sequência, temos de garantir que este, só é introduzido caso seja diferente do estado anterior nessa mesma sequência. Para além disso, de forma a melhorar o desempenho do algoritmo, aproveitando uma propriedade da estrutura especial do subproblema, cada tarefa só é introduzida numa sequência sempre que o custo da sua introdução seja compensado pela sua dual, ou seja, sempre que o valor absoluto da dual da tarefa seja superior ao atraso ponderado da mesma tarefa. Desta forma, todas as sequências são de custo reduzido decrescente, logo, uma tarefa só é adicionada a uma sequência se diminuir o seu custo reduzido. O algoritmo termina quando não é possível introduzir mais tarefas em nenhuma sequência (todas as tarefas foram avaliadas), ou quando não existem tarefas que reduzam o custo reduzido. Com o registo de todas estas sequências é selecionada aquela que o custo reduzido é menor, sendo essa a solução do subproblema, em cada chamada. O custo reduzido é a soma dos atrasos ponderados de todas as tarefas executadas e das respetivas duais. Por exemplo se tivermos a sequência $(1,2,3,4,2)$ com os respetivos atrasos ponderados e duais:

tarefa	atrasos ponderados	duais
1	0	-100
2	10	-200
3	20	-30
4	0	-10
2	40	-200

O custo reduzido tomará o valor de -470.

5.5.2 Heurísticas

Como vimos o algoritmo programação dinâmica gera soluções cíclicas sem ciclos de ordem 1 e 2. Com o objetivo de apenas gerar colunas válidas, ou seja, soluções onde cada tarefa só é afetada no máximo uma vez, podemos usar heurísticas. O uso da heurística é explicado pois, embora exista um modelo exato que não gera soluções cíclicas, não são conhecidas implementações eficientes deste modelo. Isto acontece porque a sua resolução passa pela avaliação de todas as combinações de sequências de afetação válidas. Em Lopes (2004, p.69) podemos encontrar as equações de recorrência para este modelo exato.

Como vimos o algoritmo exato de programação dinâmica regista para cada estado duas sequências de afetação: $H^k(j,t)$ e $H_2^k(j,t)$. Na construção das heurísticas foi usada a estrutura do algoritmo exato de programação dinâmica mas de forma a não gerar colunas com ciclos, passamos a registar apenas uma sequência de afetação: $H^k(S)$, onde S é o conjunto de tarefas já executadas. Um estado só é avaliado para ser introduzido nessa sequência se não pertencer já ao conjunto S dessa mesma sequência. É de realçar que as sequências (1,2,3,4) e (1,2,4,3) têm o mesmo conjunto S, já que a ordem pela qual as tarefas são executadas não é representada. O que acontece na realidade com esta alteração, é que o número de espaço de estados é relaxado e por isso não é obtida a solução ótima. As duas heurísticas apenas diferem na maneira como os estados são relaxados, ou seja, na maneira como os estados são selecionados para serem incluídos nas sequências de afetação e consequentemente em S.

Heurística 1 Nesta primeira heurística o critério de seleção usado é o tempo t. Quando para a mesma tarefa, a avaliação de inclusão desta tem duas ou mais sequências de afetação com o mesmo t é escolhida aquela com o menor custo reduzido. Por exemplo, para a tarefa 2 temos as sequências (1,4,2) e (3,2) e ambas têm o mesmo t, ou seja, terminam a execução da tarefa 2 no mesmo instante. A sequência selecionada é aquela com o menor custo reduzido até aquele momento de execução. Se neste caso (1,4,2) fosse a sequência com o menor custo reduzido (3,2) deixaria de ser uma sequência admissível, ou seja, em nenhum momento a sequência final seria, por exemplo, (3,2,1,4) uma vez que já tinha sido excluída de avaliação para a inclusão de novos estados.

Heurística 2 Para esta segunda heurística o critério de seleção é o número de tarefas já executadas *ntar*, ou seja, o número de tarefas de S. Assim, quando na avaliação de inclusão de uma tarefa, temos duas ou mais sequências de afetação com o mesmo número de tarefas já executas é escolhida aquela com o menor custo reduzido. Por exemplo na avaliação da tarefa 4 temos as sequências (1,2,3,4) e (7,6,5,4), aquela com o menor custo reduzido é selecionada para continuar como sequência admissível.

5.6 Políticas de inserção das colunas

O modo como as colunas são geradas e inseridas no problema mestre restrito influência o número de colunas necessárias para chegar à solução ótima e consequentemente o tempo gasto durante este processo. Na realidade, a instância dita qual a melhor política a usar, pois para uma instância pode ser mais vantajoso gerar muitas colunas de cada vez, em vez de uma de cada vez. Assim nesta secção iremos descrever quatro políticas (*A, B, C* e *D*), esquematizada nas figuras (5.1, 5.2, 5.3 e 5.4), respetivamente. As quatro figuras partilham a seguinte notação:

- k representa o índice do subproblema a ser tratado no momento. Para o problema das máquinas paralelas cada subproblema é resolvido para uma máquina, assim k representa o índice da máquina;
- m representa o número total de máquinas;

- SP^k representa o subproblema da máquina k;
- S^k representa a solução obtida com a resolução de SP^k;
- CR representa o custo reduzido da solução S^k;
- *ini* e *atrativa* variáveis auxiliares.

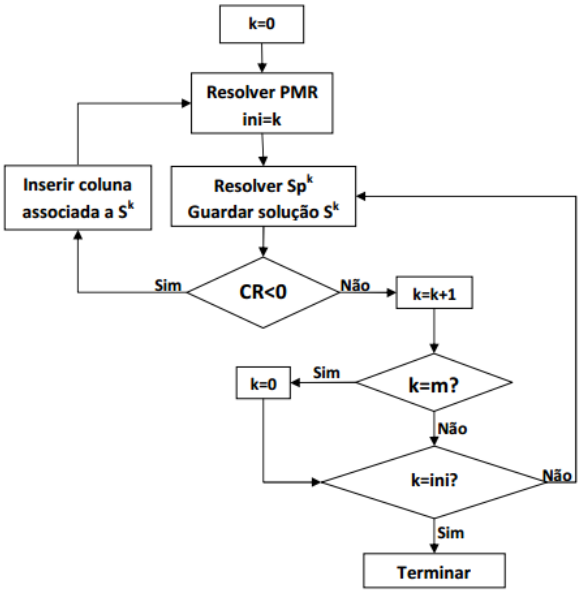


Figura 5.1: Gerar coluna e inserir coluna; variar máquina (Política A).

Na política A esquematizada na figura 5.1 o primeiro passo é resolver o problema mestre restrito RMP. Com a informação (duais) do RMP é gerada no subproblema uma solução que caso seja atrativa (custo reduzido é negativo) é inserido no RMP a coluna associada a essa solução. O processo volta novamente ao primeiro passo. Apenas geramos soluções para a máquina seguinte quando uma solução gerada não for atrativa. Para um exemplo com 2 máquinas, primeiro são geradas soluções para a máquina 0 depois para a máquina 1, voltando novamente à máquina 0 e assim consecutivamente. O processo termina quando não forem geradas soluções atrativas para nenhuma das máquinas. Na política B esquematizada na figura 5.2 o processo é semelhante ao da figura 5.1. A grande diferença é que o subproblema gera soluções sempre para o próximo k, ou seja, a máquina está sempre a variar.

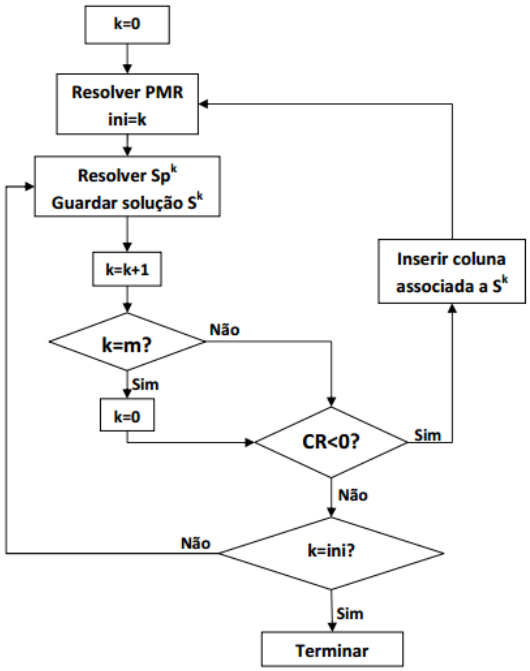


Figura 5.2: Gerar coluna, variar máquina e inserir coluna (Política B).

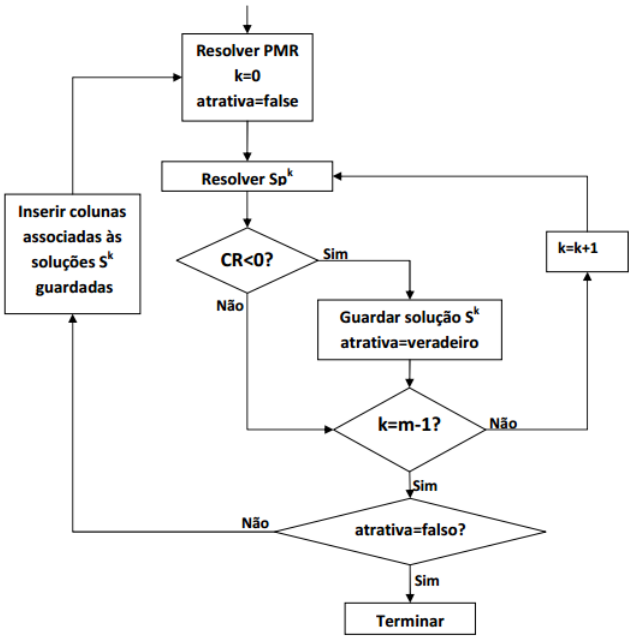


Figura 5.3: Gerar colunas; inserir colunas geradas (Política C).

Na política C esquematizada na figura 5.3 é gerada uma solução para cada máquina. Depois deste processo, as colunas associadas às soluções atrativas, são inseridas no *RMP*. De seguida resolve-se o *RMP* e este processo repete-se até que, para nenhuma máquina

seja gerada uma solução atrativa. Esta política privilegia problemas em que seja mais vantajoso gerar várias colunas (resolver vários subproblemas) do que resolver várias vezes o *RMP*. Por fim a política *D* esquematizada na figura 5.4 é semelhante à anterior. O subproblema é resolvido para todas as máquinas, obtendo uma solução para cada máquina. A principal diferença é que no *RMP* só inserimos uma coluna atrativa que representa a melhor solução das obtidas para cada máquina.

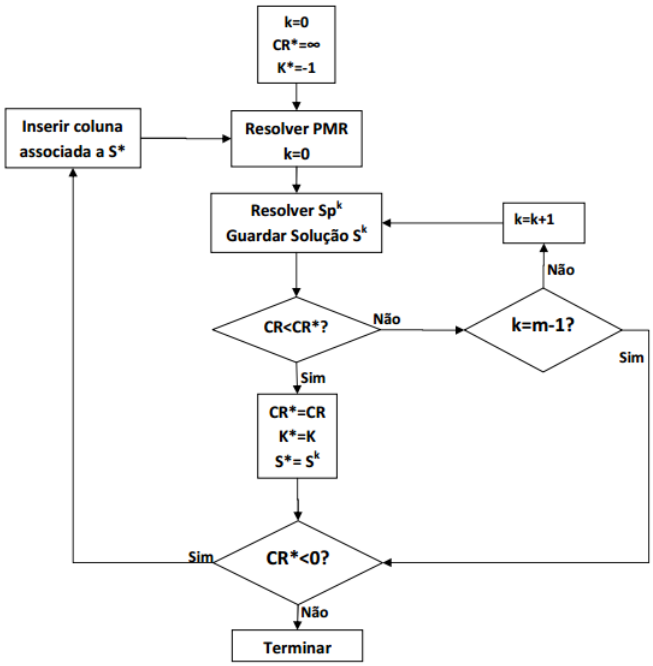


Figura 5.4: Gerar colunas e variar máquina; inserir melhor coluna (Política *D*).

5.7 Exemplo

Nesta secção iremos demonstrar, para pequeno exemplo de duas máquinas e quatro tarefas, como são geradas colunas e como estas interagem com o problema mestre restrito com o objetivo de obter a solução ótima linear. No conjunto de tabelas 5.3 temos os dados para o exemplo. Esse conjunto de tabelas segue a mesma estrutura do conjunto de tabelas 5.1, detalhada na secção 5.2.

		a_k	l_k
k	*1*	0	1
	2	12	2

(a) Informação das máquinas

		r_j	d_j	w_j
j	*1*	7	95	95
	2	23	100	62
	3	4	80	55
	4	12	97	68

(b) Informação das tarefas

s_{ij}		j *1*	*2*	*3*	*4*
i	*1*	-	21	38	31
	2	26	-	32	24
	3	27	30	-	12
	4	30	30	26	-

(c) Tempos de preparação (setup)

p_{jk}		k *1*	*2*
j	*1*	31	51
	2	30	31
	3	72	49
	4	26	60

(d) Tempos de processamento das tarefas

Tabela 5.3: Dados do exemplo com 4 tarefas e 2 máquinas.

5.7.1 Heurística inicial

Depois de descrito o exemplo iremos exemplificar as etapas da heurística inicial, para este mesmo exemplo. A primeira etapa é ordenar as tarefas. Assim ordenados pelo rácio temos 1,4,3,2, uma vez que:

tarefa	rácio
1	95/95=1
4	97/62=1.43
3	80/55=1.45
2	100/62=1.6

Depois de ordenadas as tarefas, para cada uma é selecionada qual a máquina onde esta vai ser executada. Começando pela tarefa 1, com menor rácio, temos:

Tarefa 1:

- Máquina 1: $max(0, (max(a_1 + s_{l_1 1}, r_1) + p_{11}) - d_1) * w_1 = max(0, (max(0 + 0, 7) + 31) - 95) * 95 = 0$
- Máquina 2: $max(0, (max(a_2 + s_{l_2 1}, r_1) + p_{12}) - d_1) * w_1 = max(0, (max(12 + 26, 7) + 51) - 95) * 95 = 0$

Como temos o mesmo valor para as duas máquinas usa-se o fator de desempate:

- Máquina 1: $s_{l_1 1} + p_{11} = 0 + 31 = 31$
- Máquina 2: $s_{l_2 1} + p_{12} = 26 + 51 = 77$

Assim a tarefa 1 é executada na máquina 1 (figura 5.5a).

Tarefa 4:

- Máquina 1: $max(0, (max(C_1 + s_{14}, r_4) + p_{41}) - d_4) * w_4 = max(0, (max(38+31, 12) + 26) - 97) * 68 = 0$
- Máquina 2: $max(0, (max(a_2 + s_{l_2 4}, r_4) + p_{42}) - d_4) * w_4 = max(0, (max(12+24, 12) + 60) - 97) * 68 = 0$

Mais uma vez como temos o mesmo valor para as duas máquinas usa-se o fator de desigualdade:

- Máquina 1: $s_{14} + p_{41} = 31 + 26 = 57$
- Máquina 2: $s_{l_2 4} + p_{42} = 24 + 60 = 84$

A tarefa 4 é executada na máquina 1 após a tarefa 1 (figura 5.5b).

Tarefa 3:

- Máquina 1: $max(0, (max(C_4 + s_{43}, r_3) + p_{31}) - d_3) * w_3 = max(0, (max(95 + 26, 4) + 72) - 80) * 55 = 6215$
- Máquina 2: $max(0, (max(a_2 + s_{l_2 3}, r_3) + p_{32}) - d_3) * w_3 = max(0, (max(12 + 32, 4) + 49) - 80) * 55 = 715$

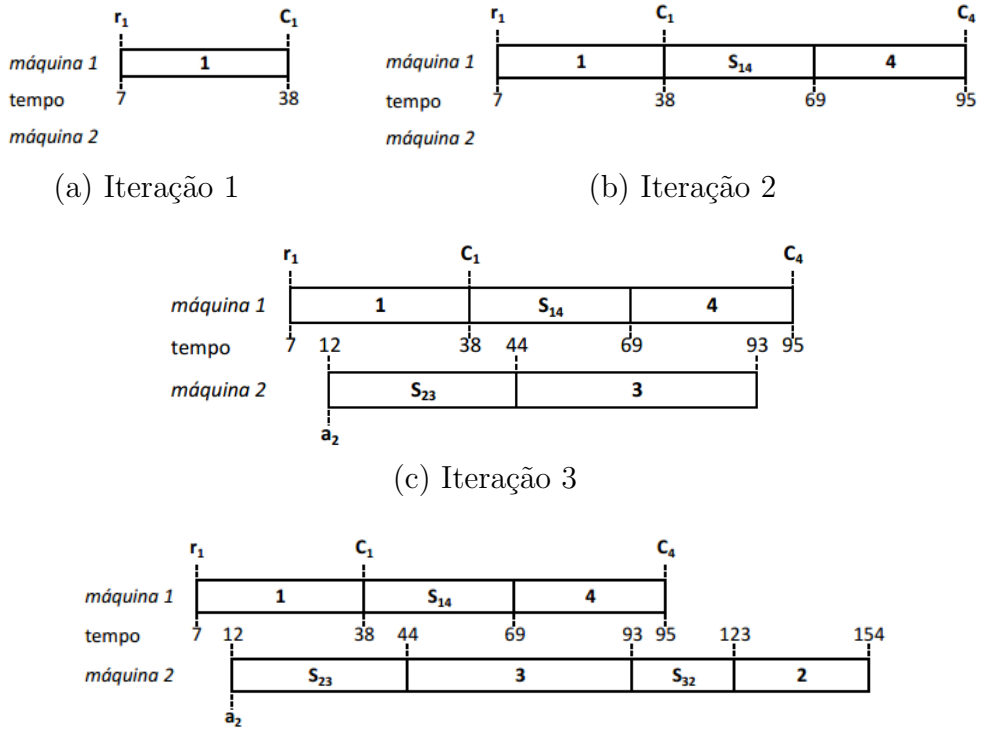
Assim sendo a tarefa 3 é executada na máquina 2 (figura 5.5c).

Tarefa 2:

- Máquina 1: $max(0, (max(C_4 + s_{42}, r_2) + p_{21}) - d_2) * w_2 = max(0, (max(95+30, 23) + 30) - 100) * 62 = 3410$

- Máquina 2: $max(0,(max(C_3+s_{32},r_2)+p_{22})-d_2)*w_2 = max(0,(max(93+30,23)+31)-100)*62 = 3348$

Por fim a tarefa 2 é executada na máquina 2 após a tarefa 3 (figura 5.5d). Nessa mesma figura 5.5d temos a solução final da heurística.



(a) Iteração 1

(b) Iteração 2

(c) Iteração 3

(d) Iteração 4

Figura 5.5: Solução heurística.

5.7.2 Resolução da relaxação linear

Já vimos a estrutura do problema mestre restrito e como são geradas as colunas no subproblema. Podemos agora explicar como estes interagem, usando o exemplo descrito. Neste exemplo usaremos a política de inserção de colunas B esquematizada na figura 5.2 e o algoritmo que resolve o subproblema de forma exata (algoritmo programação dinâmica). O problema mestre restrito é inicializado com as colunas geradas pela heurística inicial, uma coluna para cada máquina. Assim, as primeiras duas colunas correspondem

à afetação das tarefas para as máquinas 1 e 2, que têm as sequências (1,4) e (3,2), respetivamente. Assim, é garantindo que o problema mestre é iniciado com colunas que contêm uma solução válida. Posteriormente resolvemos o problema mestre, obtendo as duais que serão enviadas ao subproblema. É então executado o algoritmo do subproblema (algoritmo de programação dinâmica) que devolve uma nova coluna. Estes dois passos, resolução do problema mestre e algoritmo programação dinâmica, são repetidos até que não se obtenha mais nenhuma coluna atrativa. As colunas não se repetem e para uma coluna ser atrativa tem de ter custo reduzido negativo. Quando para todas as máquinas, nenhuma das colunas geradas for atrativa o processo termina e é obtida a solução ótima da relaxação linear.

iteração	máquina	custo reduzido	custo sequência	sequência	dual 1	dual 2	dual 3	dual 4
1	1	-4063	0	(2)	0	-4063	0	0
2	2	-5584	2542	(1,2)	-4063	-4063	0	0
3	1	-2542	0	(1,2)	-1271	-1271	-2792	0
4	2	-3348	715	(3)	0	0	-4063	0
5	1	-6288	408	(2,4)	0	-3348	-1521	-3348
6	2	-4784	3025	(2,3)	-2940	-3348	-4461	-3348

Tabela 5.4: Execuções do algoritmo do subproblema.

Na tabela 5.4 temos a listagem para cada iteração, das sequências geradas pelo algoritmo que resolve o subproblema. Para além disso temos informação do custo reduzido, o custo da sequência (soma ponderada dos atrasos) e quais as duais que "alimentaram"

	y_1^1	y_1^2	y_2^1	y_2^2	y_3^1	y_3^2	y_4^1	y_4^2
m=1	1	0	1	0	1	0	1	0
m=2	0	1	0	1	0	1	0	1
t=1	1	0	0	1	1	0	0	0
t=2	0	1	1	1	1	0	1	1
t=3	0	1	0	0	0	1	0	1
t=4	1	0	0	0	0	0	1	0
Z	0	4063	0	2542	0	715	408	3025

Tabela 5.5: Estrutura do problema mestre restrito com todas as colunas.

o subproblema, em cada iteração. Podemos ver, por exemplo, que na primeira iteração a tarefa 2 é a única que tem uma dual atrativa, portanto como seria de esperar só essa tarefa foi selecionada. Por outro lado, na última linha todas as tarefas têm duais atrativas, mas a sequência que minimiza o custo reduzido apenas inclui duas tarefas. Na tabela 5.5 temos o problema mestre restrito final com todas as colunas geradas assim como as colunas iniciais da heurística.

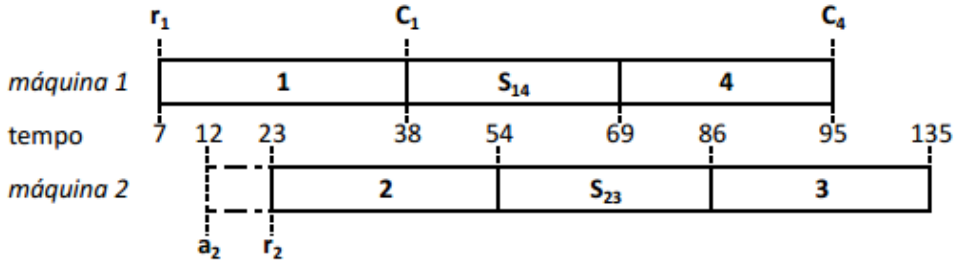


Figura 5.6: Esquema da solução.

Na figura 5.6 temos um esquema com a representação gráfica da solução da instância do exemplo. Nesta instância em particular, o ótimo linear corresponde ao ótimo inteiro que passa pela seleção das colunas y_1^1 e y_4^2, ou seja, selecionar as sequências 1 e 4 para as máquinas 1 e 2 respetivamente. Por outro lado as colunas y_1^1 e y_4^2, representam as sequências (1,4) e (2,3), para as máquinas 1 e 2, respetivamente. A solução ótima tem assim um custo de 3025, que é a soma dos custos das colunas selecionadas.

Capítulo 6

SearchCol

Neste capítulo é apresentado um algoritmo SearchCol para o problema das máquinas paralelas não idênticas, abordado nesta dissertação. Para este método são detalhadas as diversas etapas que compõe o algoritmo assim como diferentes estratégias/opções que podem ser tomadas para o problema abordado nesta dissertação.

6.1 Framework SearchCol

O *framework* SearchCol, proposto por Alvelos et al. (2010), é um *framework* genérico para problemas de otimização combinatória. Este tem uma abordagem diferente dos métodos enumerativos, como o *branch-and-price*, ao combinar a geração de colunas com a procura por meta-heurísticas na obtenção de boas soluções aproximadas num curto espaço de tempo.

O primeiro passo de um algoritmo SearchCol é resolver a relaxação linear de um modelo de decomposição por geração de colunas. Com esta primeira fase são obtidas diversas soluções (colunas) para cada subproblema. No SearchCol, a solução geral do problema é vista como a combinação de soluções para cada um dos subproblemas, obtidas na geração de colunas. Dependendo do problema, a solução do subproblema pode ser vazia. No problema das máquinas paralelas, essa solução existe, que passa por uma máquina não executar nenhuma tarefa.

O segundo passo de um algoritmo SearchCol é a pesquisa por meta-heurística da

solução, ou seja, por composição de uma solução para cada subproblema. Em cada iteração esta procura é influenciada pela informação recebida pelo ótimo do problema mestre restrito e pelo valor da solução incumbente atual. Desta forma o algoritmo SearchCol é independente do problema, pois para diferentes problemas, a solução é construída da mesma forma, o que varia é o algoritmo que resolve o subproblema, que é dependente do problema, mas visto pelo SearchCol como uma "caixa-preta". O passo seguinte é definir pertubações na geração de colunas, com o objetivo de gerar novas colunas, esperando que estas melhorem a solução incumbente. Depois de geradas as novas colunas (com as colunas perturbadas) inicia-se uma nova pesquisa meta-heurística. O algoritmo termina quando um dos critérios de paragem se verificar. No algoritmo 1 temos os principais passos de um algoritmo SearchCol.

```
1 Geração de colunas;
2 Pesquisa meta-heurística;
3 repita
4     Definição da perturbação na geração de colunas;
5     Resolução da geração de colunas (perturbada);
6     Pesquisa meta-heurística;
7 até Validação critério de paragem;
```

Algoritmo 1: Algoritmo SearchCol

6.2 Definição do espaço de procura restrito

Como vimos uma solução geral do problema consiste em selecionar uma solução de cada subproblema. Se considerarmos todas as combinações possíveis, o número de soluções do espaço de procura é $n_1 \times n_2 \times \ldots \times n_k$ onde n_k é o número de soluções do subproblema k. O número de combinação pode ser bastante gastante grande, o que dificulta a procura e piora o desempenho. Inicialmente o espaço de procura vai conter as soluções dos subproblemas geradas durante a resolução da relaxação linear (D). Durante as sucessivas iterações o espaço de procura será diferente, pois são acrescentadas as soluções dos subproblemas obtidas na perturbação de geração de colunas. O que se pretende é que o espaço de procura não seja muito pequeno, correndo o risco de "retirar" possíveis boas soluções, nem

muito grande para não dificultar a procura. Assim, se tamanho do espaço de procura inicial vezes um parâmetro λ ultrapassar um determinado valor ($\lambda \times |D|$), colunas obtidas na relaxação linear, com um alto valor do custo reduzido são removidas da geração de colunas e do espaço de procura, com o objetivo de aumentar o desempenho do algoritmo. O parâmetro λ deve ser ≥ 1.

6.3 Perturbação da geração de colunas

Embora existam outras maneiras, no SearchCol, a perturbação da geração de colunas é feita adicionando restrições ao problema mestre restrito e modificando, de acordo com as restrições, os coeficientes da função objetivo do algoritmo do subproblema. A perturbação da geração de colunas fixa variáveis dos subproblemas a 0 ou 1. No caso das máquinas paralelas, as variáveis dos subproblemas correspondem ao número de vezes que uma tarefa é executada numa máquina, assim, a perturbação da geração de colunas força que tarefas sejam executadas/não executadas nas máquinas, forçando que um ou mais planos, que contenham determinada tarefa, sejam selecionados ou não selecionados. Este tipo de perturbação de geração de colunas tem a vantagem de não ser necessário alterar o subproblema, pois os valores das duais das restrições de perturbação obtidas do problema mestre restrito são usadas nos valores dos coeficientes da função objetivo do subproblema. Para este tipo de implementação existem diversas alternativas de perturbar a geração de colunas, seja quando a solução corrente (incumbente) é admissível ou não admissível.

De uma maneira geral, quando é não admissível, o que se faz é adicionar perturbações para garantir que as restrições não são violadas. Para este caso temos duas alternativas, que são, apenas considerar as variáveis do subproblema da solução incumbente que violam restrições, ou considerar todas as variáveis de todos os subproblemas. Durante as sucessivas iterações diferentes perturbações podem ser utilizadas de modo a diversificar o modo como as colunas são geradas. Na subsecção seguinte iremos apresentar um exemplo de perturbação para quando a solução é admissível. Para mais detalhe sobre a perturbação de geração colunas e diferentes tipos de perturbações ver Alvelos et al. (2010, 2013)

6.3.1 Exemplo de perturbação

Nesta subsecção iremos exemplificar o modo de funcionamento da pertubação de geração de colunas utilizando a perturbação baseado na geração de colunas. Este tipo de perturbação usa o valor ótimo obtido na última solução da geração de colunas juntamente com um parâmetro p de valor real entre 0 e 1. Este parâmetro é utilizado para definir a partir de que valor as variáveis dos subproblemas são fixadas no valor b, que pode ser 1 ou 0. Assim, quando o valor (fracionário) da variável do subproblema é maior que $(1-p)$ então esta é fixada a 1 ou 0, dependendo do tipo de iteração

Por questões de simplificação vamos assumir que existe uma instancia com 4 tarefas e 2 máquinas, onde no fim duma iteração de geração de colunas são obtidos os valores detalhados na tabela 6.1.

Máquina (Subproblema)	Valor das variáveis	Tarefas executadas	Valor do custo (soma ponderada dos atrasos)
1	$y_1^1 = 0,25$	$(1,3)$	33
	$y_2^1 = 0,25$	$(1,2)$	34
	$y_3^1 = 0,5$	$(2,3,4)$	105
	$y_4^1 = 0$	$(1,4)$	90
2	$y_1^2 = 0,25$	$(2,3)$	63
	$y_2^2 = 0,5$	$(1,4)$	48
	$y_3^2 = 0$	$(1,3)$	90

Tabela 6.1: Dados para o exemplo da perturbação.

Na tabela 6.1 temos os planos gerados até essa iteração. Para a máquina 1 e 2 temos, respetivamente, 4 e 3 planos. Para cada plano temos também o valor do custo, as tarefas que são executadas, e o valor das variáveis obtidos na resolução do problema mestre restrito (solução da geração de colunas). Por fim vamos assumir como valor de parâmetros $p = 0,4$ e $b = 1$.

Posto isto, neste momento temos de verificar para as variáveis do subproblema x_{jk}, quais satisfazem a restrição $x_{jk} \geq 1 - p$, partindo do valor das variáveis y_j^k. Para isso são somados os valores das varáveis y_j^k onde para variável do subproblema x_{jk}, a tarefa j é

executada:

$$x_{11} = y_1^1 + y_2^1 + y_4^1 = 0,25 + 0,25 + 0 = 0,5$$
$$x_{21} = y_2^1 + y_3^1 = 0,75$$
$$x_{31} = y_1^1 + y_3^1 = 0,75$$
$$x_{41} = y_3^1 + y_4^1 = 0,5$$
$$x_{12} = y_2^2 + y_3^2 = 0,5$$
$$x_{22} = y_1^2 = 0,25$$
$$x_{32} = y_1^2 + y_3^2 = 0,25$$
$$x_{42} = y_2^2 = 0,5$$

Assim sendo, temos duas variáveis que satisfazem a condição, x_{21} e x_{31} já que $0,75 \geq 0,6$. Passamos assim a ter mais duas restrições no problema mestre restrito uma para cada variável, que forçam estas variáveis a ser executadas uma vez que o parâmetro b é igual a 1. A dual de cada uma destas restrições será representada por σ_j^k, onde j o índice da tarefa e k o índice da máquina. As duais das tarefas são representadas por ω_j, onde j o índice da tarefa. No problema mestre restrito temos então:

$$Min\ 33y_1^1 + 34y_2^1 + 105y_3^1 + 90y_4^1 + 63y_1^2 + 48y_2^2 + 90y_3^2$$
$$sujeito\ a:$$
$$y_1^1 + y_2^1 + y_3^1 + y_4^1 \leq 1;$$
$$y_1^2 + y_2^2 + y_3^2 \leq 1;$$
$$y_1^1 + y_2^1 + y_4^1 + y_2^2 + y_3^2 \geq 1; \qquad (\omega_1)$$
$$y_2^1 + y_3^1 + y_1^2 \geq 1; \qquad (\omega_2)$$
$$y_1^1 + y_3^1 + y_1^2 + y_3^2 \geq 1; \qquad (\omega_3)$$
$$y_3^1 + y_4^1 + y_2^2 \geq 1; \qquad (\omega_4)$$
$$y_2^1 + y_3^1 = 1; \qquad (\sigma_2^1) \qquad (6.1)$$
$$y_1^1 + y_3^1 = 1; \qquad (\sigma_3^1) \qquad (6.2)$$

Em 6.1 e 6.2 temos então as duas restrições, correspondendo respetivamente às variáveis x_{21} e x_{31}. Da resolução do problema mestre restrito obtemos os seguintes valores das duais:

$$\omega_1 = 48$$
$$\omega_2 = 21$$
$$\omega_3 = 42$$
$$\omega_4 = 0$$
$$\sigma_2^1 = 99$$
$$\sigma_3^1 = 77$$

Com estes valor das duais (ω_j, σ_j^k), caso o próximo subproblema a resolver correspondesse à máquina 1, então o valor das duais das tarefas passadas ao algoritmo do subproblema (α_j) seriam:

$$\alpha_1 = \omega_1 = 48$$
$$\alpha_2 = \omega_2 + \sigma_2^1 = 120$$
$$\alpha_3 = \omega_3 + \sigma_3^1 = 119$$
$$\alpha_4 = 0$$

6.3.2 Perturbações usadas

Foram usadas dois tipos de perturbações neste trabalho. O primeiro tipo de perturbações consiste em fixar a 0 todas as variáveis que tenham um valor fraccionário e que não pertençam à solução incumbente. O segundo tipo de perturbações consiste em fixar a 0 as variáveis que fizeram parte de soluções visitadas na pesquisa mais vezes. O número de variáveis selecionadas é baseado num parâmetro que corresponde à proporção de variáveis que já foram visitadas pelo menos uma vez. Este parâmetro foi fixado a 0,1.

O primeiro tipo de perturbações é utilizado sempre que a solução incumbente foi melhorada na iteração atual. Se a incumbente não foi melhorada, é utilizada o segundo tipo de perturbações.

Foram utilizados dois critérios de paragem: duas iterações seguidas sem melhoria da solução incumbente e o tempo limite de 7200 segundos.

6.4 Representação e avaliação de soluções

No problema das máquinas paralelas cada solução dos subproblemas corresponde a um plano para uma máquina. Uma solução para o problema geral consiste num conjunto de soluções dos subproblemas para cada uma das máquinas. Por exemplo, para um problema com quatro máquinas em que para cada máquina temos quatro planos (soluções dos subproblemas), na solução $s = \{3, 1, 4, 1\}$ para a primeira máquina temos o terceiro plano, para a segunda máquina o primeiro plano e assim consecutivamente.
Cada solução tem a si associada dois valores: valor de admissibilidade (*feasibility*) e não admissibilidade (*infeasibility*).

Para o valor de admissibilidade podemos considerar ou os custos das solução dos subproblemas que compõem a solução geral ou o custo reduzido. Em relação ao valor de não admissibilidade podemos ter o número de restrições violadas na solução s, a quantidade total violada ou ainda uma função que compõe estes dois valores. A quantidade total violada é calculada pelo valor de folga, ou desvio, de cada restrição. Se a restrição for de igualdade a folga é o valor absoluto da diferença entre os valores. Se for de desigualdade $(\geq, >, \leq, <)$ o valor da folga é a quantidade que viola a restrição. Na tabela seguinte temos exemplos para cada caso:

restrição	folga
$11 = 6$	5
$14 \leq 7$	7
$5 \geq 4$	0
$5 \geq 8$	3

Depois de detalhar como os valores de admissibilidade e não admissibilidade das soluções são calculados podemos agora explicar como comparamos duas soluções. Uma solução com menor valor de não admissibilidade é sempre melhor que uma com maior valor de não admissibilidade, qualquer que sejam os valores de admissibilidade. Quando o valor

de não admissibilidade é igual então o valor de admissibilidade é usado para comparar as soluções.

6.5 Meta-heurísticas

6.5.1 Considerações gerais

Um aspecto importante da procura no SearchCol é como obtemos novas soluções pela alteração de soluções existentes. Para isso temos dois tipos de movimentos básicos, que são:

- retirar uma solução de um subproblema a uma solução geral (parcial ou completa);
- adicionar uma solução de um subproblema a uma solução parcial.

Com a combinação destes dois tipos de movimentos podemos fazer a alteração de uma solução dum subproblema por outra (retirar e adicionar). Quando fazemos uma alteração deste género, temos uma solução que é vizinha doutra. Por definição a k-vizinhança de uma solução é o conjunto de soluções obtidas alterando k ou menos soluções de subproblemas. A definição da k-vizinhança define o número de vizinhos do espaço de procura.

Pesquisa Local A procura de um ótimo local começa com uma solução base seguida da procura por melhores soluções numa vizinhança. Durante a procura na vizinhança podemos ainda optar por duas estratégias.

Na primeira, sempre que se encontra um vizinho melhor do que a solução atual, este passa a ser a solução atual, e este processo repete-se até a solução atual não ter melhores vizinhos. No algoritmo 2 temos os passos desta estratégia.

Na segunda, a procura da melhor solução é feita para todas as soluções na sua vizinhança, e se o melhor vizinho (solução), for melhor que a solução atual, passa então a ser a solução atual. Se não existir, temos um ótimo local. Esta estratégia segue os passo descritos no algoritmo 3.

```
1  s = solBase;
2  repita
3      s'=vizinhoSeguinte(s);
4      se s' melhor que s então
5          s=s';
6      fim
7  até que não haja vizinhos analisados;
```

Algoritmo 2: Pesquisa Local - Estratégia 1.

```
1  s = solBase;
2  fim=falso;
3  repita
4      s'=melhorVizinho(s);
5      se s' melhor que s então
6          s=s';
7      senão
8          fim=verdadeiro;
9      fim
10 até fim=verdadeiro;
```

Algoritmo 3: Pesquisa Local - Estratégia 2.

6.5.2 Soluções iniciais

A pesquisa de soluções na heurística parte duma solução inicial. Existem diversas alternativas para definir as soluções iniciais. Aqui apenas apresentamos quatro - Aleatória uniforme, Aleatória baseada na geração de colunas, Baseada na geração de colunas e Incumbente.

Aleatória uniforme Neste tipo de geração da solução inicial cada solução de cada subproblema é selecionada de forma aleatória, em que cada solução de cada subproblema tem a mesma probabilidade de ser selecionada (distribuição uniforme).

Aleatória baseada na geração de colunas Nesta alternativa a seleção da solução para cada subproblema também é aleatório, embora neste caso a probabilidade de uma solução de cada subproblema ser selecionada é dada pelo valor das variáveis (soluções) de cada subproblema obtido na última resolução do problema mestre restrito.

Baseada na geração de colunas Nesta alternativa, a solução de cada subproblema será selecionada pelo maior valor das variáveis de cada subproblema, obtido na última resolução do problema mestre restrito.

Incumbente Nesta alternativa, a solução inicial usada será a solução incumbente, ou seja, a melhor solução até ao momento.

6.5.3 Pesquisa em Vizinhanças Variáveis

A meta-heurística Pesquisa em Vizinhanças Variáveis (*Variable Neighborhood Search - VNS*) tem diversas variantes e pode ser aplicada a vários problemas (Mladenovic e Hansen, 1997). No algoritmo 4 temos o pseudocódigo da variante implementada no *framework* SearchCol++ (Alvelos et al., 2010).

```
1  s =solucaoInicial();
2  k=1;
3  numiter=1;
4  repita
5      s'=perturbarAleatoriamente(s,k);
6      s"=pesquisaLocal(s');
7      se s" melhor que s então
8          k=1;
9          numiter=1;
10         s=s";
11     senão
12         se k ≠ kmax_então
13             k=k+1;
14         fim
15         numiter=numiter+1;
16     fim
17 até numiter=nitersemmelhoria;
```

Algoritmo 4: Pesquisa em Vizinhanças Variáveis

O algoritmo é iniciado com uma solução inicial (ver subsecção 6.5.2). Se a solução obtida da pesquisa for melhor que solução base então passa a solução base e os parâmetros k e ***numiter*** são atualizados. A função de perturbação, linha 5, pega na solução s e altera aleatoriamente k soluções dos subproblemas. Por exemplo aplicar esta função à

solução $s = \{1, 1, 1, 1\}$ sendo $k = 2$, uma solução resultante possível seria $s' = \{1, 3, 1, 4\}$, assumindo que todas estas soluções dos subproblemas existem, os subproblemas 2 e 4 foram alterados aleatoriamente. Assim, tanto a escolha dos subproblemas como as soluções dos subproblemas é aleatória. O algoritmo termina quando o número de iterações seguidas sem melhoria for máximo (*nitersemmelhoria*).

6.5.4 Exemplo

Nesta subsecção iremos demonstrar o modo de funcionamento do algoritmo Pesquisa em Vizinhanças Variáveis. Uma vez que no algoritmo são feitas pesquisas locais, inicialmente iremos apresentar o modo de funcionamento do algoritmo de Pesquisa Local. O exemplo apresentado de seguida servirá para demostrar ambos os algoritmos.

Consideremos que na resolução da relaxação linear por geração de colunas para um problema com 5 tarefas e 3 máquinas obtínhamos, para cada uma das máquinas (subproblemas), as soluções dos subproblemas (colunas) dadas na tabela 6.2.

Máquina (Subproblema)	Solução do subproblema	Tarefas executadas	Valor do custo
1	0	(1, 2)	56
	1	(1, 3)	76
	2	(3)	22
	3	(2)	33
2	0	(2, 3, 4)	65
	1	(4, 5)	55
	2	(1, 3)	51
	3	(5)	23
	4	(2)	13
3	0	(3, 4)	54
	1	(1)	33

Tabela 6.2: Soluções dos subproblemas.

Na tabela 6.2 para cada uma das soluções dos subproblemas para cada máquina temos o valor do custo e as tarefas que nela são executadas. Podemos ver que para a primeira, segunda e terceira máquina temos, respetivamente 4, 5 e 2 colunas. Na avaliação das soluções o valor de não admissibilidade usado será o número de restrições violadas, que

neste caso é o número de tarefas que não são executadas. Como valor de admissibilidade a soma dos custos.

Pesquisa Local Para demostrar o algoritmo Pesquisa Local iremos utilizar 1-vizinhança e como estratégia de procura o melhor vizinho do espaço de procura. A solução inicial é $s = \{1, 2, 1\}$, com valor de admissibilidade 160 e valor de não admissibilidade 3.

Vizinho	Solução			Valor de admissibilidade	Valor de não admissibilidade
V1	0	2	1	140	2
V2	2	2	1	106	3
V3	3	2	1	117	2
V4	1	0	1	174	1
V5	1	1	1	164	1
V6	1	3	1	132	2
V7	1	4	1	122	2
V8	1	2	0	181	2

Tabela 6.3: Vizinhança da solução $\{1, 2, 1\}$.

Na tabela 6.3 temos a primeira iteração do algoritmo. Cada linha corresponde a um vizinho da solução atual. Podemos ver que temos 8 vizinhos, que são todas as soluções que conseguimos obter a partir da solução base, alterando apenas uma solução do subproblema. Nesta primeira iteração temos o melhor vizinho (V5) com melhor valor que solução atual. Assim a algoritmo continua, passando V5 a ser a solução base da nova iteração. Nesta procura local temos duas iterações, chegando ao ótimo local correspondendo à solução $s = \{2, 1, 1\}$ com valor de admissibilidade 110 e valor de não admissibilidade 1.

Pesquisa em Vizinhanças Variáveis Iremos agora apresentar o modo de funcionamento da meta-heurística Pesquisa em Vizinhança Variáveis (*Variable Neighborhood Search - VNS*). A pesquisa local deste exemplo usa a mesma estratégia e a mesma k-vizinhança usada no exemplo anterior. Como critério de paragem são usadas 5 iterações sem melhoria de solução. Na tabela 6.4 temos os resultados da execução do algoritmo, em cada iteração. A solução base inicial usada é $s = \{1, 2, 1\}$ (iteração 1). Com a perturbação aleatória dessa solução obtemos $s = \{2, 2, 1\}$. A solução da perturbação é então usada na

pesquisa local e chegando ao ótimo local, $s = \{2, 1, 1\}$. Este ótimo como é melhor que a solução atual será a solução atual da iteração seguinte. Na terceira iteração obtemos a solução $s = \{0, 3, 0\}$. Como esta solução é a melhor solução possível, o algoritmo itera mais 5 vezes, prefazendo 8 iterações no total.

Iteração	k	Resultado				Valor de admissibilidade	Valor de não admissibilidade
1	1	Solução atual	1	2	1	160	3
		Sol. perturbada	2	2	1		
		Pesquisa Local	2	1	1	110	1
2	1	Solução atual	2	1	1	110	1
		Sol. perturbada	3	1	1		
		Pesquisa Local	2	1	1	110	1
3	2	Solução atual	2	1	1	110	1
		Sol. perturbada	0	3	1		
		Pesquisa Local	0	3	0	133	0
4	1	Solução atual	0	3	0	133	0
...							
8	2	Pesquisa Local	0	3	0	133	0

Tabela 6.4: Iterações do VNS.

6.5.5 SearchCol com *MIP*

Na ideia base do SearchCol é que na fase de procura se utilize heurísticas, embora seja também possível nessa fase, utilizar um *MIP solver*, que retorna o valor ótimo para as soluções dos subproblemas disponíveis até ao momento de execução.

6.6 Critérios de paragem

Como critério de paragem podemos temos quatro formas de terminar o SearchCol. Logo que uma condição se verifique o SearchCol termina e a melhor solução obtida até ao momento é retornada. Os quatro critérios de paragem são:

- Limite temporal;
- Número máximo de iterações sem melhoria da solução incumbente;

- Número máximo de iterações totais;
- O desvio relativo (gap) ser menor que um determinado valor.

A cada critério de paragem temos associado um parâmetro com o valor definido para cada critério. Os primeiros três parâmetros podem tomar valores superiores a 0. Já o parâmetro do último critério de paragem (desvio relativo) pode tomar valores entre 0 e 1. Ainda em relação ao último critério, o desvio é calculado pela fórmula $\frac{|Z_{inc}-Z_{RL}|}{|Z_{inc}|}$, onde Z_{inc} representa o valor da solução incumbente e Z_{RL} o valor da relaxação linear inicial, obtido sem perturbações da geração de colunas.

Capítulo 7

Testes Computacionais

Neste capítulo são apresentados os diversos testes realizados no *framework Search-Col++*. Todos os testes computacionais apresentados foram realizados num computador pessoal com um processador Intel i5-2430M a 2.40GHz e com 4 GB de memória.

7.1 Política de inserção de colunas e algoritmos do subproblema

Nesta primeira fase, foram testadas as políticas de geração de colunas. No *framework SearchCol++* o parâmetro *PARDEC_onecolperiter* controla a política utilizada. Na tabela seguinte temos, para cada política detalhada na secção 5.6, o respetivo valor do parâmetro:

Politica	Valor parmetro (*PARDEC_onecolperiter*)	Figura
A	*PARDEC_onecolperiter*==1	5.1
B	*PARDEC_onecolperiter*==3	5.2
C	*PARDEC_onecolperiter*==0	5.3
D	*PARDEC_onecolperiter*==2	5.4

Para além da política de geração de colunas, neste primeiro conjunto de testes, foram testados os algoritmos de resolução dos subproblemas detalhados na secção 5.5. De seguida temos, para os três algoritmos, o nome utilizado:

- *Modelo exato:* original;

- *Heurística 1:* t-j;

- *Heurística 2:* ntar-j.

As instâncias de teste foram retiradas do conjunto de instâncias referidas em Lopes (2004). Nesse conjunto de instâncias, para cada instância existem cinco níveis de congestionamento. Na mesma instância ao subir de nível de congestionamento, os dados dessa instância são exatamente iguais, exceto as datas de entrega dos trabalhos que são reduzidas, ou seja, os maiores níveis de congestionamento têm datas de entrega de menor valor. Nestes testes foram utilizados apenas os congestionamentos 3 e 4. Por outro lado, para cada um destes conjuntos temos 10 instâncias. Por exemplo, *50-180-4* representa o conjunto de 10 instâncias, com 50 máquinas, 180 tarefas e congestionamento 4.

Por fim nesta primeira fase foi utilizado o *MIP solver* na fase de pesquisa e o parâmetro *spheur=3*, explicado na secção seguinte.

Instâncias	original				j-t				ntar-j			
	trl	tmip	valmip	numcol	trl	tmip	valmip	numcol	trl	tmip	valmip	numcol
10-50-3	2	0	13793	579	2	0	13517	566	1	0	13548	702
10-50-4	2	0	40489	602	2	0	40783	598	1	0	40468	754
10-70-3	12	66	44033	954	13	56	32814	943	6	125	34723	1376
10-70-4	16	123	122286	991	18	15	79165	1014	10	24	78279	1512
10-90-3	38	360	79020	1311	37	528	78941	1283	17	1061	85623	2165
10-90-4	72	234	183325	1423	75	565	150037	1473	45	1215	185794	2546
10-100-3	69	597	121301	1482	68	1162	121301	1503	29	1883	121301	2711
10-100-4	150	304	241588	1670	165	1452	244007	1795	96	2110	252978	3324
20-100-3	9	205	28366	1290	10	230	27075	1275	4	198	25527	1523
20-100-4	11	108	75760	1355	13	75	75530	1361	5	39	74641	1643
20-120-3	25	646	61147	1637	25	662	55229	1603	7	609	56988	2078
20-120-4	41	903	134515	1796	44	751	119718	1780	18	736	137755	2358
30-150-3	26	1003	71133	2052	27	1179	73606	2004	7	1039	72567	2432
30-150-4	38	1068	167662	2195	43	1085	159894	2226	15	954	166859	2726
50-150-3	4	28	27637	1927	7	6	27471	1990	2	8	27518	2066
50-150-4	5	6	71421	2034	7	4	71342	2093	3	3	71371	2225
50-180-3	15	177	23854	2512	19	235	23989	2584	6	338	24645	2724
50-180-4	15	754	92235	2574	20	763	86032	2659	7	734	88539	2943

Tabela 7.1: Tempos médios - onecolperiter==0

Nas tabelas 7.1, 7.2, 7.3, 7.4, temos os resultado obtidos para cada política/algoritmos do subproblema, em valores médios, já que temos 10 instância para cada conjunto. De seguida temos o significado de cada uma das colunas:

- *trl* - Tempo da relaxação linear;
- *tmip* - Tempo do MIP;
- *valmip* - Valor da solução;
- *numcol* - Número de colunas totais.

Para melhor comparar os resultados, temos na tabela 7.5 o somatório dos valores médios para cada um dos campos. Nesta tabela não foi incluída a política *onecolperiter==2*, pois para essa política apenas foi testado o algoritmo do subproblema original, uma vez que os tempos de computação são bastante maus. Como podemos ver, na última linha da tabela 7.3, temos o somatório dos valores médios, que é muito superior ao das outras políticas, logo esta não é uma política viável para este problema.

Analisando os resultados da tabela 7.5 podemos ver que o melhor tempo da relaxação linear é obtido com *onecolperiter==0* e o algoritmo *ntar-j*. Por outro lado o melhor valor da solução é obtido com *onecolperiter==1* e o algoritmo *t-j*. Como não temos uma política que domine as outras utilizaremos nos próximos testes, as políticas/algoritmos do subproblema seguintes: *onecolperiter==0/ntar-j*, *onecolperiter==1/t-j*.

Instâncias	original				j-t				ntar-j			
	trl	tmip	valmip	numcol	trl	tmip	valmip	numcol	trl	tmip	valmip	numcol
10-50-3	4	0	13559	720	5	0	13484	670	3	0	13376	750
10-50-4	4	0	40370	779	5	0	40276	752	4	0	40428	791
10-70-3	23	60	28204	1479	24	23	26595	1418	12	86	31475	1596
10-70-4	30	138	81699	1560	32	10	75736	1572	16	46	79399	1711
10-90-3	77	857	50476	2285	85	540	28072	2565	33	2400	78438	2650
10-90-4	133	1154	144365	2632	140	1215	106233	3123	61	1103	160677	2995
10-100-3	148	1733	89484	2838	168	1806	74312	3644	51	2106	114798	3652
10-100-4	316	1792	207436	3842	292	2164	199604	4446	127	2046	252704	4166
20-100-3	17	191	26988	1564	17	143	25166	1484	7	209	29473	1567
20-100-4	20	99	75367	1655	22	67	74820	1594	9	85	74889	1677
20-120-3	45	632	43675	2251	44	670	45202	2175	14	638	57265	2273
20-120-4	71	606	110941	2467	69	698	114153	2341	25	792	133677	2433
30-150-3	44	1094	56686	2434	46	1125	57451	2341	14	972	71855	2407
30-150-4	64	1063	150342	2654	64	1035	136644	2485	23	973	151953	2600
50-150-3	8	10	27566	1572	9	6	27592	1522	6	10	27481	1576
50-150-4	9	7	71521	1630	10	5	71379	1595	6	6	71435	1627
50-180-3	22	432	23852	2233	26	219	23876	2213	11	386	24177	2237
50-180-4	25	663	89082	2348	28	1237	88329	2251	13	810	86618	2332

Tabela 7.2: Tempos médios - onecolperiter==1

Instâncias	original			
	trl	tmip	valmip	numcol
10-50-3	11	0	14047	383
10-50-4	13	0	41675	411
10-70-3	79	20	63588	634
10-70-4	122	19	113860	713
10-90-3	270	100	85623	894
10-90-4	540	70	191224	1007
10-100-3	481	100	114765	1017
10-100-4	1178	81	252978	1205
20-100-3	94	1443	34118	772
20-100-4	133	952	83335	828
20-120-3	261	5899	68227	978
20-120-4	529	5698	145047	1135
30-150-3	336	5940	80704	1116
30-150-4	623	6057	175097	1268
50-150-3	78	38	28147	996
50-150-4	84	27	72273	1025
50-180-3	260	2525	33931	1307
50-180-4	323	4283	125288	1338
$\sum$	5416	33252	1723926	17025

Tabela 7.3: Tempos médios - onecolperiter==2

Instâncias	original				j-t				ntar-j			
	trl	tmip	valmip	numcol	trl	tmip	valmip	numcol	trl	tmip	valmip	numcol
10-50-3	3	0	13781	400	3	0	13851	401	2	0	13698	493
10-50-4	2	0	40602	422	3	0	40495	432	2	0	40644	542
10-70-3	11	28	63859	662	12	28	50134	668	7	42	45949	991
10-70-4	15	22	113514	713	18	12	81391	734	10	28	91386	1039
10-90-3	33	89	85623	906	33	74	71477	905	19	223	85623	1457
10-90-4	62	72	191224	1016	70	140	181805	1076	45	330	191224	1737
10-100-3	57	176	121301	1050	60	199	121301	1073	31	363	121301	1799
10-100-4	132	87	252978	1225	147	274	240722	1309	99	550	252978	2220
20-100-3	8	669	28940	813	9	663	27998	817	5	377	26873	963
20-100-4	11	166	77603	870	13	66	76743	879	7	159	76826	1051
20-120-3	20	5555	64296	1038	20	5147	59443	1021	9	5365	62006	1297
20-120-4	36	4846	119729	1156	39	4934	121305	1179	20	1196	139244	1515
30-150-3	20	5544	80704	1225	21	5922	80704	1208	10	4347	80704	1432
30-150-4	34	6239	159959	1308	37	5816	166681	1354	18	4259	175097	1622
50-150-3	6	11	27705	1089	7	6	27730	1107	5	17	27690	1142
50-150-4	6	8	71662	1114	7	5	71653	1120	5	6	71475	1179
50-180-3	14	831	24362	1421	16	1078	24365	1428	9	1032	24103	1518
50-180-4	16	3148	112991	1433	18	3157	105178	1440	13	1588	89842	1571

Tabela 7.4: Tempos médios - onecolperiter==3

		trl	tmip	valmip	numcol
onecolperiter==0	original	550	6579	1599564	28384
	t-j	595	8767	1480449	28749
	ntar-j	279	11076	1559124	37807
onecolperiter==1	original	1060	10531	1331612	36941
	t-j	1086	10964	1228922	38191
	ntar-j	437	12669	1500115	39042
onecolperiter==3	original	485	27491	1650831	17861
	t-j	531	27521	1562976	18151
	ntar-j	314	19883	1616663	23569

Tabela 7.5: Somatório dos tempos médios

7.2 Parâmetro *spheur*

Os testes anteriores mostraram que as heurísticas obtinham melhores resultados que o algoritmo original. Como foi referido, a simples utilização das heurísticas não garante a obtenção da solução ótima da relaxação linear sendo necessário a utilização do algoritmo exato. Existem dois modos de alternar entre os algoritmo heurístico e exato, que é controlado com o parâmetro *spheur*. Aqui ficam os dois valores do parâmetro e a respetiva descrição:

- **spheur=1** - Heurística é usada até que não sejam geradas colunas atrativas, depois uma coluna do algoritmo exato é inserida e o processo repetido;
- **spheur=3** - Heurística é usada até que não sejam geradas colunas atrativas, depois o algoritmo exato é utilizado até ao final.

Nos testes da secção anterior, foi utilizado *spheur=3*. Neste segundo conjunto de testes pretende-se perceber se com *spheur=1* obtemos melhores resultados. Para isso foram selecionadas seis instâncias com diferentes número de tarefas e máquinas. Assim, neste segundo conjunto de testes será utilizado mais uma vez o *MIP* na procura e *spheur=1*.

	spheur=1		spheur=3	
	tempo total	valmip	tempo total	valmip
onecol_0/ntar-j	4179,372	365343	2336,609	376368
onecol_1/t-j	4182,101	239622	3879,57	213857
$\sum$	8361,473	604965	6216,179	590225

Tabela 7.6: Comparação parâmetro *spheur*

Na tabela 7.6 temos os resultados obtidos deste segundo conjunto de testes. Como podemos ver, com *spheur=1* obtemos melhores valores das soluções para a política/heurística *onecol_0/ntar-j*, tendo *spheur=3* obtido melhores resultados para a outra política/heurística. Se considerarmos os somatório dos valores (última linha), vemos que *spheur=3* tem resultados melhores e tempos melhores. Assim sendo, nos próximos testes iremos utilizar *spheur=3*.

7.3 Proibição de soluções

Neste conjunto de testes iremos introduzir proibições na geração de colunas. Sabemos que, a variável x_{jk} representa o número de vezes que a tarefa j é executada na máquina k. Sabemos como obtemos o valor de x_{jk} a partir das variáveis y_p^k (plano p em k), obtidas na geração de colunas. Assim, quando o valor da variável x_{jk} se encontra entre $0 < x_{jk} \leq T$ ou $0 \leq x_{jk} \leq T$, é adicionado uma restrição na geração de colunas que traduz a restrição: $x_{jk} \leq 0$.

Neste segundo conjunto de testes iremos utilizar novamente o *MIP* e as seis instâncias das subsecções anteriores: *10-50-3-2*, *10-70-3-7*, *10-90-3-4*, *10-100-3-5*, *20-100-3-10* e *20-120-3-3*. De seguinte temos os valores dos parâmetros que iremos variar, para além de *spheur*, e o seu significado:

- *solsFromForbidFirst=0* - Não utilizar proibição (testes anteriores);
- *solsFromForbidFirst=1* - Proibir quando $0 < x_{jk} \leq T$;
- *solsFromForbidFirst=2* - Proibir quando $0 \leq x_{jk} \leq T$;
- *solsFromForbidThreshold* - Valor do limite T.

	solsFromForbidFirst	tempo total	valmip	numSolsFromForbidFirst
onecol_0/ntar-j	0	2336,609	376368	-
	1	2423,213	141876	16799
	2	4615,113	206068	18343
onecol_1/t-j	0	3879,57	213857	-
	1	4813,545	151902	19570
	2	6410,535	274622	23209

Tabela 7.7: Variação de solsFromForbidFirst.

Na tabela 7.7 temos o resultado dos testes da variação do parâmetro *solsFromForbidFirst* pelos três valores possíveis. Nessa mesma tabela temos, os tempos totais, o valor das soluções finais e o *numSolsFromForbidFirst* que representa o número de soluções introduzidas pela proibição. Todos estes valores são o somatório dos valores obtidos para

	solsFromForbidThreshold	tempo total	valmip	numSolsFromForbidFirst
onecol_0/ntar-j	0,1	4473,87	365453	666
	0,4	1312,367	138278	12069
	0,5	2423,213	141876	16799
	0,6	6230,667	218468	19805
	0,8	9132,975	316903	26417
onecol_1/t-j	0,1	3994,558	249611	449
	0,4	1764,317	137418	13368
	0,5	4813,545	151902	19570
	0,6	7054,411	157473	25069
	0,8	10046,558	240810	30617

Tabela 7.8: Variação de solsFromForbidThreshold.

cada uma das seis instâncias e como podemos ver, os resultados estão divididos por política/heurística. Neste primeiro conjunto de testes o limite utilizado foi de $T = 0.5$, assim temos o parâmetro *solsFromForbidThreshold=0.5*. Pela comparação dos valores obtidos, quando *solsFromForbidFirst=1* obtemos os melhores valores da solução final.

Depois de sabemos qual o melhor valor para o parâmetro *solsFromForbidFirst*, fizemos outro conjunto de testes, onde inicialmente, variámos o limite $T = \{0.1, 0.5, 0.8\}$, utilizando novamente o *MIP* e as seis instâncias anteriores. Na tabela 7.8 o resultado desses testes, tabela esta que segue a mesma estrutura da tabela 7.7.

Com este conjunto de testes, vimos que temos os melhores valores das soluções (*valmip*) quando *solsFromForbidThreshold=0.5*. Por último, tentamos perceber se variando o melhor limite atual $T = 0.5$ obtínhamos melhores resultados. Desta forma testamos também os limites $T = \{0.4, 0.6\}$. Os resultados desses testes estão também representados na tabela 7.8. Analisando estes últimos testes, podemos ver que com *solsFromForbidThreshold=0.4* obtemos sempre melhores resultados face aos outros limites.

Posto isto, podemos dizer que o melhor valor das soluções é obtido utilizando os seguintes valores:

- Heurística *t_j*;
- *onecolperiter=1*;
- *spheur=3*;

- *solsFromForbidFirst=1*;
- *solsFromForbidThreshold=0.4*.

7.4 Testes Finais

Os testes finais do SearchCol, foram realizados com os valores dos parâmetros referidos no final da secção anterior, mas em vez de utilizar o *MIPsolve*, utilizamos na fase de pesquisa a meta-heurística Pesquisa em Vizinhanças Variáveis. Por outro nesta fase são também usadas perturbações. As perturbações usadas são descritas nas subsecção 6.3.2. Como solução inicial foi usada a solução inicial incumbente descrita na subsecção 6.5.2. No anexo A temos nas tabelas A.1 a A.9 os resultados desses testes. Para além dos resultados desses testes, temos também nessas tabelas, os resultados do algoritmo Branch-and-Price (BP) de Lopes e Valério de Carvalho (2007) e do SearchCol com *MIPsolver* na fase de pesquisa a que chamamos de MIPHeur. Ainda nessa tabela temos nas últimas colunas, o desvio relativo (GAP) das soluções do MIPHeur e SearchCol relativamente à solução ótima do BP.

Para melhor analisar os resultados, criamos a tabela 7.9, com o somatório dos tempos de execução e dos valores das soluções, para o conjunto de cada 10 instâncias. Temos também representado nas duas últimas colunas dessa tabela, o GAP, relativo ao somatório das soluções ótimas do BP com o somatório das soluções do MIPHeur e SearchCol.

Atendendo às qualidades das soluções vemos que:

- O MipHeur obtém melhores soluções que o SearchCol;
- O SearchCol melhora nas instâncias mais congestionadas assim como o Mipheur, ou seja, o GAP é menor para instâncias com congestionamento 4;
- No SearchCol mantendo o mesmo número de máquinas e aumentando o número de tarefas, a qualidade de soluções piora. Por outro lado, para o mesmo número de tarefas a qualidade melhora com o aumento do número de máquinas.

Em relação aos tempos de computação:

Instância	Tempos totais segundos			Valor das Soluções			GAP	
	BP	MipHeur	SearchCol	BP	MipHeur	SearchCol	MipHeur	SearchCol
10-50-3	27	49	118	131943	134842	145874	2%	11%
10-50-4	31	55	82	401895	402764	421354	0,2%	5%
10-70-3	796	475	1539	236983	265946	331214	12%	40%
10-70-4	561	420	1207	718291	757363	977444	5%	36%
10-90-3	1359	6250	3541	209104	280715	565361	34%	170%
10-90-4	4502	13558	4244	832307	1062328	1574427	28%	89%
10-100-3	4659	19736	6718	310881	743125	913409	139%	194%
10-100-4	17871	24559	7695	1124156	1996038	2049143	78%	82%
20-100-3	634	1607	2561	232937	251660	391425	8%	68%
20-100-4	635	887	1934	721673	748198	1062182	4%	47%
20-120-3	1979	7141	4762	205680	452018	564258	120%	174%
20-120-4	5159	7665	4077	752721	1141525	1380404	52%	83%
30-150-3	2204	11702	8628	256297	574505	647386	124%	153%
30-150-4	15948	10989	6722	898839	1366442	1622081	52%	80%
50-150-3	721	157	1651	271359	275922	514809	2%	90%
50-150-4	552	151	726	708616	713785	1024859	1%	45%
50-180-3	4131	2453	6284	233458	238757	562956	2%	141%
50-180-4	10894	12653	3002	828356	883285	1450646	7%	75%
$\sum$	72663	120506	65489	9075496	12289218	16199232		

Tabela 7.9: Testes finais-resumo.

	BP	MipHeur	VNS
Congestionamento 3	16510	49569	35801
Congestionamento 4	56153	70936	29688

Tabela 7.10: Soma dos tempos totais por congestionamento.

- Olhando para a soma total dos tempos, o SearchCol é o que tem melhores tempos seguindo-se o Branch-and-Price e só depois o MipHeur. Se essa análise for dividida por congestionamento, tabela 7.10, o Branch-and-Price evidência-se nas instâncias menos congestionadas sendo o SearchCol o mais rápido nas instâncias de congestionamento 4.

- Com o SearchCol temos quatro conjuntos de instâncias, todas elas de congestionamento 4, com menor tempo total *10-100-4, 20-120-4, 30-150-4, 50-180-4*. Para esses conjuntos o GAP é cerca de 80%.

Por fim realizamos um conjunto de testes com instâncias grandes. Para estas instân-

cias o BP atinge o limite de quatro horas. O tempo e os valores das soluções obtidas com o SearchCol são apresentados na tabela 7.11. Analisando tanto os tempos de computação como as soluções obtidas, vemos que o SearchCol se torna vantajoso para grandes instâncias com alto valor de rácio de tarefas sobre máquinas, sendo estas de grande congestionamento.

Instância	SearchCol		
	Tempo total (segundos)	Valor	GAP da relaxação linear
50-220-4-1	7201	247374	42%
50-220-4-2	897	201272	55%
50-220-4-3	9959	215806	45%
50-220-4-4	6654	198825	43%
50-220-4-5	11364	176922	46%
50-220-4-6	6032	188302	46%
50-220-4-7	7537	241631	43%
50-220-4-8	11823	167624	55%
50-220-4-9	7689	258685	39%
50-220-4-10	4858	203458	49%

Tabela 7.11: Testes para instâncias 20-220-4.

Capítulo 8

Conclusões

Neste trabalho apresentou-se a aplicação do *framework* "pesquisa meta-heurística por geração de colunas" "SearchCol" ao problema de máquinas paralelas não idênticas com tempos de preparação das máquinas dependentes da sequência de afetação das tarefas, tendo como objetivo a minimização da soma ponderada dos atrasos.

Foram apresentadas diversas formulações encontradas na literatura, relativas a problemas de máquinas paralelas. Para três dessas formulações foram descritas quais as modificações necessárias fazer de forma a serem adaptadas ao problema abordado nesta dissertação.

Foram detalhados, tanto o algoritmo de programação dinâmica que resolve o subproblema de forma exata, como heurísticas que resolvem o subproblema com soluções sem ciclos. O teste das heurísticas, juntamente com o algoritmo exato permitiu verificar que com uma das heurísticas tínhamos uma redução significativa dos tempos de computação na relaxação linear (ntar-j), por sua vez com a heurística (j-t) obtínhamos normalmente boas soluções finais. Nesse conjunto de testes foram também testadas as diferentes formas de introdução das colunas no problema mestre. A partir da análise de resultados optamos por duas políticas de geração de colunas/heurísticas. Os testes seguintes tiveram como objetivo selecionar os diferentes valores dos parâmetros que melhor se adequassem à aplicação do problema de máquinas paralelas não idênticas. Os resultados dos testes finais do SearchCol, foram apresentados juntamente com os resultados do algoritmo Branch-

and-Price e com a utilização de um *solver* genérico de programação inteira (MIPHeur) na resolução do problema inteiro resultante da geração de colunas.

Sabendo que as soluções do BP são ótimas, o objetivo final da dissertação é a comparação do SearchCol tanto na qualidade das soluções como no tempo necessário para as obter. A utilização do SearchCol ao problema de máquinas paralelas não idênticas, faz sentido para grandes instâncias com grandes valores de congestionamentos, pois como vimos, os tempos de computação do MIPHeur e Branch-and-Price (BP) eram muito superiores. No caso extremo do conjunto das 10 instâncias que demoraram mais tempo, 10 máquinas 100 tarefas e congestionamento elevado, a diferença entre o tempo do SearchCol e o BP é um pouco mais de 10000 segundos. No lado oposto, vimos também que para as instâncias pequenas sem grande congestionamento, 10 máquinas 50 tarefas e congestionamento reduzido, o SearchCol demora mais de 4 vezes que o BP.

Para instâncias de maior dimensão (50 máquinas e 220 tarefas), dos métodos testados apenas o SearchCol forneceu soluções.

Bibliografia

Akker, J., Hoogeveen, J., e Velde, S. (1999). Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872.

Akker, J., Hoogeveen, J., e Velde, S. (2005). *Applying Column Generation to Machine Scheduling*, pages 303–330. Springer.

Allahverdi, A., Ng, C., Cheng, T., e Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032.

Alvelos, F., de Sousa, A., e Santos, D. (2010). Searchcol: Metaheuristic search by column generation. In *Hybrid Metaheuristics*, volume 6373 of *Lecture Notes in Computer Science*, pages 190–205. Springer Berlin / Heidelberg.

Alvelos, F., Sousa, A., e Santos, D. (2013). Combining column generation and metaheuristics. In Talbi, E.-G., editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 285–334. Springer Berlin Heidelberg.

Baker, K. R. e Trietsch, D. (2009). *Principles of Sequencing and Scheduling*. Wiley.

Brucker, P. (2004). *Scheduling Algorithms*. Springer, 4th edition.

Chen, Z. e Powell, W. (1999). Solving parallel machine scheduling problems by column generation. *Informs Journal on Computing*, 11(1):78–94.

Cheng, R. e Gen, M. (1997). Parallel machine scheduling problems using memetic algorithms. *Computers & Industrial Engineering*, 33(3-4):761–764.

Desaulniers, G., Desrosiers, J., e Solomon, M. M. (2005). *Column generation*. Springer, New York.

Graham, R., Lawler, E., Lenstra, J., e Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(2):287–326.

Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *International Journal of Production Research*, 31(7):1579–1594.

Hall, N. G. e Posner, M. E. (1991). Earliness-tardiness scheduling problems, i: Weighted deviation of completion times about a common due date. *Operations Research*, 39(5):836–846.

Kurz, M. E. e Askin, R. G. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39(16):3747–3769.

Lenstra, J., Rinnooy Kan, A., e Brucker, P. (1977). *Complexity of Machine Scheduling Problems*, volume 1, pages 343–362. Elsevier.

Li, C.-L. e Cheng, T. C. E. (1994). The parallel machine min-max weighted absolute lateness scheduling problem. *Naval Research Logistics*, 41(1):33–46.

Lopes, M. P. (2004). *Resolução de Problemas de Programação de Máquinas Paralelas pelo Método de Partição e Geração de Colunas*. PhD thesis, Universidade do Minho.

Lopes, M. P. e Valério de Carvalho, J. M. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, 176(3):1508–1527.

Mladenovic, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer, 3rd edition.

Rabadi, G., Moraga, R., e Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17:85–97.

Unlu, Y. e Mason, S. J. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785 – 800.

Vallada, E. e Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622.

Weng, M. X., Lu, J., e Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, 70(3):215–226.

Zhu, Z. e Heady, R. B. (2000). Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. *Computers & Industrial Engineering*, 38(2):297–305.

Apêndice A

Resultados testes finais

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
10-50-3-1	3	8573	5	8575	6	8956	0%	4%
10-50-3-2	2	17532	5	18097	9	18704	3%	7%
10-50-3-3	6	6951	5	7788	15	10053	12%	45%
10-50-3-4	2	12242	5	12242	26	15878	0%	30%
10-50-3-5	2	9275	5	9275	15	11862	0%	28%
10-50-3-6	3	11254	5	12380	7	12085	10%	7%
10-50-3-7	4	15701	5	16011	11	16233	2%	3%
10-50-3-8	2	18624	5	18624	13	18929	0%	2%
10-50-3-9	2	4243	4	4302	14	5626	1%	33%
10-50-3-10	1	27548	5	27548	2	27548	0%	0%
10-50-4-1	1	26910	5	26910	2	26910	0%	0%
10-50-4-2	3	48587	5	48587	10	51418	0%	6%
10-50-4-3	6	39997	6	40065	6	41171	0%	3%
10-50-4-4	2	40560	6	40560	3	40560	0%	0%
10-50-4-5	2	35846	5	35846	8	36048	0%	1%
10-50-4-6	2	36747	6	36747	3	36747	0%	0%
10-50-4-7	4	41512	6	41512	19	43881	0%	6%
10-50-4-8	7	41879	5	42680	13	52270	2%	25%
10-50-4-9	2	32140	6	32140	3	32140	0%	0%
10-50-4-10	2	57717	6	57717	15	60209	0%	4%

Tabela A.1: Testes Finais - (10-50).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
10-70-3-1	32	31521	33	35020	77	33930	11%	8%
10-70-3-2	59	35772	52	38298	152	47369	7%	32%
10-70-3-3	45	30577	35	33859	141	39920	11%	31%
10-70-3-4	172	18300	162	24676	184	30224	35%	65%
10-70-3-5	90	17392	27	18813	192	22737	8%	31%
10-70-3-6	49	12722	26	14053	193	19605	10%	54%
10-70-3-7	30	15220	35	17927	137	23004	18%	51%
10-70-3-8	161	12685	43	13598	94	33062	7%	161%
10-70-3-9	55	37415	31	41349	131	49826	11%	33%
10-70-3-10	103	25379	32	28353	237	31537	12%	24%
10-70-4-1	38	75538	27	76704	121	84447	2%	12%
10-70-4-2	62	76984	34	83041	139	88762	8%	15%
10-70-4-3	48	76860	56	81637	79	99354	6%	29%
10-70-4-4	95	73443	35	77854	123	113303	6%	54%
10-70-4-5	37	67343	37	67753	105	72048	1%	7%
10-70-4-6	60	53881	61	61078	137	65397	13%	21%
10-70-4-7	80	66199	36	69401	131	130645	5%	97%
10-70-4-8	53	48732	32	49429	87	79242	1%	63%
10-70-4-9	39	106431	35	109267	182	114976	3%	8%
10-70-4-10	49	72880	67	81199	105	129270	11%	77%

Tabela A.2: Testes Finais - (10-70).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
10-90-3-1	31	15786	308	20378	145	31038	29%	97%
10-90-3-2	64	27145	337	35547	371	81387	31%	200%
10-90-3-3	128	48445	1446	62810	336	102083	30%	111%
10-90-3-4	97	18981	888	26293	398	62880	39%	231%
10-90-3-5	197	31463	100	31808	323	86014	1%	173%
10-90-3-6	65	8903	148	10158	378	28800	14%	223%
10-90-3-7	285	12127	334	14162	479	39919	17%	229%
10-90-3-8	140	8140	84	9995	380	30345	23%	273%
10-90-3-9	142	11795	1589	22782	419	16864	93%	43%
10-90-3-10	210	26319	1017	46782	313	86031	78%	227%
10-90-4-1	247	59117	1530	76706	328	121360	30%	105%
10-90-4-2	274	77336	321	86555	414	125306	12%	62%
10-90-4-3	753	112140	5015	174996	418	206571	56%	84%
10-90-4-4	367	86847	1157	106762	545	153617	23%	77%
10-90-4-5	685	127892	2514	184377	510	252450	44%	97%
10-90-4-6	162	60834	130	64788	402	130117	6%	114%
10-90-4-7	115	53149	112	53392	415	58205	0%	10%
10-90-4-8	380	63357	1145	90321	404	115690	43%	83%
10-90-4-9	1124	74165	739	96645	440	124873	30%	68%
10-90-4-10	395	117470	895	127786	368	286238	9%	144%

Tabela A.3: Testes Finais - (10-90).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
10-100-3-1	2275	11722	1714	26263	745	58178	124%	396%
10-100-3-2	407	16641	1101	20516	659	65568	23%	294%
10-100-3-3	195	24128	3903	41832	651	69568	73%	188%
10-100-3-4	134	10809	1019	21162	451	44145	96%	308%
10-100-3-5	262	35895	2168	102601	827	94476	186%	163%
10-100-3-6	347	57135	2713	127934	745	127934	124%	124%
10-100-3-7	146	9192	2465	32586	919	48354	255%	426%
10-100-3-8	420	48242	2128	168634	762	170016	250%	252%
10-100-3-9	201	29533	182	29534	268	67665	0%	129%
10-100-3-10	272	67584	2344	172063	691	167505	155%	148%
10-100-4-1	3419	111245	2495	203365	751	200082	83%	80%
10-100-4-2	2689	93456	1998	241453	871	269046	158%	188%
10-100-4-3	562	84580	2364	206792	883	230790	144%	173%
10-100-4-4	1361	81028	3010	129876	777	97618	60%	20%
10-100-4-5	522	122669	342	124708	692	165947	2%	35%
10-100-4-6	1058	156189	2694	196152	722	213061	26%	36%
10-100-4-7	3931	76482	5123	139199	763	260329	82%	240%
10-100-4-8	2970	143754	1870	276741	725	175553	93%	22%
10-100-4-9	494	92384	2396	186806	883	252735	102%	174%
10-100-4-10	865	162369	2267	290946	628	183982	79%	13%

Tabela A.4: Testes Finais - (10-100).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
20-100-3-1	36	21840	34	23493	160	29987	8%	37%
20-100-3-2	85	25460	224	27231	278	43497	7%	71%
20-100-3-3	33	23850	101	25172	253	32258	6%	35%
20-100-3-4	34	23798	275	25845	192	41527	9%	74%
20-100-3-5	38	15126	19	15455	278	26575	2%	76%
20-100-3-6	42	24863	20	25414	217	45149	2%	82%
20-100-3-7	85	22136	750	30357	255	44854	37%	103%
20-100-3-8	65	38709	28	39747	246	51715	3%	34%
20-100-3-9	89	11954	22	12162	298	25578	2%	114%
20-100-3-10	127	25201	134	26784	384	50285	6%	100%
20-100-4-1	45	72636	48	74880	147	88916	3%	22%
20-100-4-2	75	75854	23	77401	101	102686	2%	35%
20-100-4-3	57	65850	39	68170	104	120611	4%	83%
20-100-4-4	69	69199	27	70891	90	95333	2%	38%
20-100-4-5	49	54533	37	56533	404	69117	4%	27%
20-100-4-6	57	75046	52	78734	174	147226	5%	96%
20-100-4-7	62	73855	44	77051	231	108637	4%	47%
20-100-4-8	92	99614	549	105353	384	125104	6%	26%
20-100-4-9	66	60650	39	63044	177	116901	4%	93%
20-100-4-10	63	74436	30	76141	121	87651	2%	18%

Tabela A.5: Testes Finais - (20-100).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
20-120-3-1	155	10415	60	10632	330	40399	2%	288%
20-120-3-2	560	14210	941	56799	582	61722	300%	334%
20-120-3-3	78	20733	652	22155	324	55099	7%	166%
20-120-3-4	544	27438	1069	71379	805	63672	160%	132%
20-120-3-5	42	14956	41	14956	95	37065	0%	148%
20-120-3-6	61	15444	1007	56327	518	56517	265%	266%
20-120-3-7	27	22179	885	45801	448	45662	107%	106%
20-120-3-8	242	35847	496	38243	618	81159	7%	126%
20-120-3-9	111	17293	964	61385	483	57210	255%	231%
20-120-3-10	159	27165	1026	74341	559	65753	174%	142%
20-120-4-1	556	52143	1066	105758	626	105086	103%	102%
20-120-4-2	1008	80823	1073	109111	877	137834	35%	71%
20-120-4-3	289	78183	1059	93354	319	159869	19%	104%
20-120-4-4	876	91180	1023	136642	459	141601	50%	55%
20-120-4-5	144	60945	164	62790	266	105130	3%	72%
20-120-4-6	351	69895	84	72199	216	132254	3%	89%
20-120-4-7	1167	65440	1053	128592	317	128628	97%	97%
20-120-4-8	222	110230	1000	199600	333	180270	81%	64%
20-120-4-9	132	54320	64	54688	149	110237	1%	103%
20-120-4-10	414	89562	1079	178791	514	179495	100%	100%

Tabela A.6: Testes Finais - (20-120).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
30-150-3-1	270	24039	1228	86616	854	78176	260%	225%
30-150-3-2	534	29419	1122	78124	738	78028	166%	165%
30-150-3-3	243	30510	1075	78935	1124	78349	159%	157%
30-150-3-4	160	27594	1316	72431	908	70263	162%	155%
30-150-3-5	177	29023	1040	30733	905	55027	6%	90%
30-150-3-6	448	27334	1317	48103	749	64970	76%	138%
30-150-3-7	44	14259	1164	55428	1023	51918	289%	264%
30-150-3-8	68	29193	1245	45971	444	76887	57%	163%
30-150-3-9	48	10998	1015	11509	1152	33796	5%	207%
30-150-3-10	212	33928	1181	66655	732	59972	96%	77%
30-150-4-1	2931	97558	1218	185316	974	168014	90%	72%
30-150-4-2	662	95103	1224	105290	168	170288	11%	79%
30-150-4-3	833	107114	1293	193830	428	211933	81%	98%
30-150-4-4	4623	89217	1211	184459	758	184397	107%	107%
30-150-4-5	3840	103631	1249	130725	1650	138815	26%	34%
30-150-4-6	157	92646	365	94610	174	164719	2%	78%
30-150-4-7	339	60806	908	63152	729	117508	4%	93%
30-150-4-8	337	85699	1032	89733	958	148542	5%	73%
30-150-4-9	1123	63524	1237	134636	433	133939	112%	111%
30-150-4-10	1103	103541	1250	184691	450	183926	78%	78%

Tabela A.7: Testes Finais - (30-150).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
50-150-3-1	23	32312	10	32487	72	63568	1%	97%
50-150-3-2	14	23655	12	24003	54	53926	1%	128%
50-150-3-3	156	28070	13	28747	481	54775	2%	95%
50-150-3-4	43	24582	13	25090	110	33523	2%	36%
50-150-3-5	49	22301	24	23068	131	35575	3%	60%
50-150-3-6	61	17832	14	18170	88	37895	2%	113%
50-150-3-7	177	11889	21	12139	44	31876	2%	168%
50-150-3-8	51	36691	15	37027	104	58899	1%	61%
50-150-3-9	14	33515	12	33673	48	71344	0%	113%
50-150-3-10	133	40512	22	41518	518	73428	2%	81%
50-150-4-1	25	72128	19	72947	104	91529	1%	27%
50-150-4-2	58	74249	14	75027	51	118628	1%	60%
50-150-4-3	24	67632	11	67670	39	99479	0%	47%
50-150-4-4	22	58275	12	58910	64	78156	1%	34%
50-150-4-5	48	67376	20	67994	77	94149	1%	40%
50-150-4-6	129	62573	19	62964	53	101253	1%	62%
50-150-4-7	143	50924	17	51603	40	91919	1%	81%
50-150-4-8	49	79786	16	80256	119	105756	1%	33%
50-150-4-9	25	80123	11	80380	74	120259	0%	50%
50-150-4-10	29	95550	13	96034	104	123731	1%	29%

Tabela A.8: Testes Finais - (50-150).

Instância	BP		MIPHeur		SearchCol		GAP	
	ttotal	opt	ttotal	valmip	ttotal	valor	MIPHeur	SearchCol
50-180-3-1	120	34501	29	34562	93	75884	0%	120%
50-180-3-2	78	11110	34	11374	124	34708	2%	212%
50-180-3-3	1057	19866	764	20911	1509	57300	5%	188%
50-180-3-4	401	14017	86	14640	110	48040	4%	243%
50-180-3-5	750	20690	678	21801	1226	57227	5%	177%
50-180-3-6	1224	17815	336	18212	237	51998	2%	192%
50-180-3-7	174	17284	401	17886	396	40396	3%	134%
50-180-3-8	85	24049	36	24238	148	61899	1%	157%
50-180-3-9	134	47328	32	47885	243	80677	1%	70%
50-180-3-10	108	26798	56	27248	2198	54827	2%	105%
50-180-4-1	320	94547	166	96105	309	149686	2%	58%
50-180-4-2	533	63012	658	64467	227	116965	2%	86%
50-180-4-3	2422	79349	1105	81459	281	136929	3%	73%
50-180-4-4	113	76016	106	77387	239	151522	2%	99%
50-180-4-5	1694	85729	4024	88518	122	148983	3%	74%
50-180-4-6	818	80973	776	82963	142	140272	2%	73%
50-180-4-7	844	74600	46	75090	261	141412	1%	90%
50-180-4-8	1958	74553	983	75516	234	139197	1%	87%
50-180-4-9	352	112777	78	113529	102	187264	1%	66%
50-180-4-10	1840	86800	4710	128251	1085	138416	48%	59%

Tabela A.9: Testes Finais - (50-180).