UNIVERSIDADE FEDERAL FLUMINENSE
CENTRO TECNOLÓGICO
MESTRADO EM ENGENHARIA DE PRODUÇÃO

JOSÉ MAURICIO BRASIL GONÇALVES

FORMULAÇÕES PARA O PROBLEMA DO *FLOW SHOP* EM DUAS MÁQUINAS COM PENALIDADES POR ATRASO NAS TAREFAS

NITERÓI
2009

JOSÉ MAURICIO BRASIL GONÇALVES

FORMULAÇÕES PARA O PROBLEMA DO *FLOW SHOP* EM DUAS MÁQUINAS COM PENALIDADES POR ATRASO NAS TAREFAS

Dissertação apresentada ao Curso de Mestrado em Engenharia de Produção da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Sistemas, Apoio à Decisão e Logística.

Orientador: Prof. EDUARDO UCHOA BARBOZA, DSc.

Niterói

2009

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

G635 Gonçalves, José Mauricio Brasil.

Formulações para o problema do flow shop em duas máquinas com penalidades por atraso nas tarefas / José Mauricio Brasil Gonçalves. – Niterói, RJ : [s.n.], 2009.
114 f.

Dissertação (Mestrado em Engenharia de Produção) - Universidade Federal Fluminense, 2009.
Orientador: Eduardo Uchoa Barboza.

1. Engenharia de Produção. 2. Escalonamento de tarefa. 3. Programação inteira. 4. Pesquisa operacional. I. Título.

CDD 658.5

JOSÉ MAURICIO BRASIL GONÇALVES

FORMULAÇÕES PARA O PROBLEMA DO *FLOW SHOP* EM DUAS MÁQUINAS COM PENALIDADES POR ATRASO NAS TAREFAS

Dissertação apresentada ao Curso de Mestrado em Engenharia de Produção da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Sistemas, Apoio à Decisão e Logística.

Aprovada em setembro de 2009.

BANCA EXAMINADORA

_____

Prof. Dr. Eduardo Uchoa Barboza – Orientador
Universidade Federal Fluminense

_____

Prof. Dr. Artur Alves Pessoa
Universidade Federal Fluminense

_____

Prof. Dra. Rosiane de Freitas Rodrigues
Universidade Federal do Amazonas

Niterói
2009

AGRADECIMENTOS

A Deus por permitir que eu estudasse.

Aos meus Pais pelo amor com que sempre conduziram minha criação.

Ao meu irmão David, por ser sempre meu melhor amigo.

A minha namorada Cristina pelo companheirismo e por ter aceitado e apoiado os momentos de renúncia ao longo do Mestrado.

Ao Professor Dr. Eduardo Uchoa, pelas aulas particulares de modelagem, pela atenção e comprometimento com o meu aprendizado. E por sempre ter paciência em explicar os conceitos, independentemente de quantas vezes fosse perguntado sobre o tema.

Aos colegas do LOGIS: Hugo Kramer, Marcos Roboredo, Luiz Aizemberg, Prof. Dr. Artur Pessoa, pela disposição em ajudar e pelas idéias apresentadas.

Aos colegas da Petrobras, em especial Cezar Silveira, por entender os momentos que tive de dispensar ao Mestrado.

Aos demais professores do curso de Mestrado em Engenharia de Produção da Universidade Federal Fluminense, pela atenção e pelas aulas ministradas.

RESUMO

Problemas de escalonamento são muito comuns nos ambientes de produção de bens e prestação de serviços. Referem-se à alocação de recursos escassos a tarefas ao longo de determinados períodos de tempo. A meta consiste em otimizar um ou mais objetivos, traduzindo-se na melhor ordem que as tarefas devem ser processadas por cada uma das máquinas. Muito dos problemas de escalonamento enquadram-se na categoria $\mathcal{NP}$ - difícil, sendo o caso do problema desta pesquisa. Este estudo tem por objetivo propor formulações de programação inteira para o problema de escalonamento determinístico *Flow shop* com duas máquinas onde existem penalidades por atraso nas tarefas. Na notação de três campos tem-se: $F_2 \mid\mid \sum w_j T_j$. O fato de buscar a minimização do somatório ponderado dos atrasos das tarefas, faz com que o problema apresente complexidade computacional superior quando comparado ao famoso problema de minimização do tempo de finalização da última tarefa a deixar o sistema: $F_2 \mid\mid C_{max}$ (sendo este último resolvido em tempo polinomial pelo algoritmo de Johnson). Foram testadas formulações com variáveis binárias x_{jit} (que assumem valor unitário se a tarefa *j* finaliza seu processamento na máquina *i* no tempo *t*) e variáveis binárias x_{jt} (que assumem valor unitário se a tarefa *j* tem seu processamento finalizado na segunda máquina no tempo *t*). Em testes elaborados com instâncias de até cinqüenta tarefas, observou-se que a formulação com a variável x_{jit} obteve limites duais iniciais mais próximos do valor ótimo da função objetivo. Por último, foi também proposta uma formulação mais compacta para o problema, não utilizando o índice *t* nas variáveis de decisão, além de uma heurística de busca local para solução do problema e heurísticas que apóiam o algoritmo de *branch and bound* a encontrar uma solução viável inicial.

Palavras-chave: Problemas de Escalonamento. Formulações. Programação Inteira.

ABSTRACT

Scheduling problems are very common in manufacturing and services industries. Scheduling deals with the allocation of scarce resources to jobs over time. The goal is optimizing one or more objectives, translating into the best order in with jobs must be processed by each machine. Many of scheduling problems belong to the $\mathcal{NP}$-hard class of problems, this being the case of the problem studied here. The goal is to propose integer programming formulations for the deterministic two machine flow shop problem, with penalties on the jobs tardiness. In the three-field notation this problem is denoted as F2 | | $\sum w_j T_j$. The fact of seeking to minimize the weighted sum of jobs tardiness makes the problem much more difficult when compared to the famous problem of minimizing the completion time of the last job to leave the system (the makespan), denoted as F2 | | C_{max} (it can be solved in polynomial time by Johnson´s algorithm). Formulations were tested with binary variables x_{jit} (that assume value one when job *j* completes at machine *i* at time *t*) and a formulation with binary variables x_{jt} (that assume value one when job *j* leaves the system at time *t*). Computational experiments with instances with up to 50 jobs have shown that the formulation with variables x_{jit} obtained initial dual limits closer to the optimal value of the objective function. Finally, it is also proposed a more compact formulation for the problem, not using the index *t* in the decision variables and a local search heuristic for solving the problem and heuristics that support the branch and bound algorithms to find a good primal bounds.

Key-words: Scheduling Problems. Formulations. Integer Programming.

LISTA DE TABELAS

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 DEFINIÇÃO DO PROBLEMA

Denominam-se “problemas de escalonamento” uma série de problemas clássicos da área de pesquisa operacional que vêm sendo pesquisados há aproximadamente cinqüenta anos. Possuem elevado número de aplicações nas atividades de produção de bens e prestação de serviços, em situações onde uma série de recursos necessita ser alocado a tarefas ao longo de períodos de tempos. A meta é otimizar um ou mais objetivos (PINEDO, 2008, p. 1). Resolver um problema de escalonamento é, de forma sucinta, determinar a seqüência ótima de execução das tarefas. O estudo de um caso específico desta série de problemas é o objeto principal deste trabalho. A seguir são expostos alguns exemplos hipotéticos onde a formulação e a resolução de um problema de escalonamento têm como resultado uma seqüência de execução das tarefas que otimiza o objetivo escolhido pelo tomador de decisão.

Exemplo 1: Uma fábrica engarrafadora de refrigerantes possui uma variada gama de produtos. Tais produtos diferenciam-se pelo volume da embalagem (um litro, um litro e meio, dois litros, entre outros) e pelo sabor (uva, laranja, cola, limão, etc). Para as linhas de produção das embalagens tipo PET, existe uma unidade de sopro (onde as garrafas são produzidas) e uma unidade de envase (onde o refrigerante é colocado no interior da garrafa). A produção de qualquer refrigerante é iniciada na unidade de sopro, seguida pela unidade de envase. Para cada refrigerante, existem tempos específicos de processamento na unidade de sopro e na unidade de envase. É necessário elaborar um plano de produção semanal que consiste numa lista dos produtos que se deseja produzir, com as datas de início e conclusão da produção de cada um deles no período. Note que existem dois recursos (representado pelas unidades de sopro e envase) que devem ser alocados a tarefas (que, no caso, é a produção dos

diferentes tipos de refrigerantes). A montagem de um plano de produção com os produtos necessários em determinada semana, levando em consideração a otimização de um critério selecionado (que pode ser, por exemplo, a minimização do tempo de conclusão do último produto ou o somatório ponderado dos atrasos em relação a uma data de término prometida) pode ser modelado como um problema de escalonamento.

Exemplo 2: A administração de um aeroporto precisa elaborar um plano de utilização das pontes de desembarque dos aviões (*fingers*, como são chamados) que minimize o tempo total gasto por todas as aeronaves aguardadas no aeroporto, em um determinado período de tempo (um dia, uma semana ou um mês). Cada aeronave necessita de um tempo para ser processada (período necessário para que os passageiros deixem o avião). Por exemplo, aeronaves de grande porte, tais como o BOEING da série 747-400, necessitam usualmente de um tempo superior ao utilizado pelo AIRBUS da série A320-200. É necessário manter uma boa utilização do sistema, visto que atrasos de última hora podem ter impactos no consumo de combustível dos aviões, nas conexões dos passageiros para outros destinos, no tempo de reabastecimento dos suprimentos, entre outros reflexos. As pontes de desembarque dos aviões no aeroporto podem ter a mesma capacidade ou não. Além disso, podem ser aceitáveis pequenos atrasos para alguns tipos de vôos (tais como vôos domésticos que tem como destino final a cidade do aeroporto). Em vôos internacionais, além de conexões ou escalas, o atraso é penalizado. Tomadas as pontes de desembarque como recursos e as aeronaves como tarefas, pode-se dizer que há um problema de escalonamento a ser resolvido. A meta poderia ser a minimização do maior atraso possível.

Exemplo 3: Determinada fábrica produz três tipos de motores. Esta fábrica possui cinco máquinas principais, enumeradas genericamente por máquina um, máquina dois, máquina três, máquina quatro e máquina cinco. Para cada tipo de motor, existe uma seqüência pré-definida de passagem através das máquinas. A produção do motor de menor potência inicia pela máquina dois, passa em seguida pela máquina quatro e por último na máquina cinco. Para o motor de maior potência, a seqüência a ser observada é: máquina um, máquina cinco, máquina quatro e máquina três. Existe um tempo de processamento em cada máquina, que é função do tipo de motor a ser processado. Tomada a produção de um determinado motor como uma tarefa e as máquinas da fábrica como recursos, a necessidade de elaborar um plano de produção periódico, que aloque de maneira ótima (segundo um critério escolhido) os recursos (máquinas da fábrica) às tarefas (produção de motores) configura um problema de escalonamento. Pode ser observado que o ambiente de produção deste exemplo é diferente do ambiente mencionando no exemplo da produção de refrigerantes. Naquele caso, todas as

tarefas inicialmente eram processadas no recurso “unidade de sopro” e depois seguiam para o recurso “unidade de envase”. Já no exemplo acima, a seqüência entre as máquinas é diferente para cada tipo de motor a ser produzido na fábrica.

Exemplo 4: O último exemplo leva em consideração a oferta de serviços financeiros por uma agência bancária. Uma série de serviços é oferecida nestes estabelecimentos, tais como: serviços de seguro, assessoria para investimentos, títulos de capitalização, pagamento de contas, descontos de cheques. Um cliente necessita contratar o serviço de seguro e um título de capitalização. Outro cliente necessita descontar um cheque e tomar conhecimento dos fundos de investimentos oferecidos pela instituição. Não existe uma ordem pré-definida para que um cliente solicite estes serviços. Ele poderá se dirigir inicialmente ao atendimento dos caixas, descontar seu cheque, depois procurar o gerente da sua conta para conhecer as oportunidades de investimentos. O cliente poderia executar esta tarefa de maneira inversa, dirigindo-se primeiro ao seu gerente, para só depois ir aos caixas. Este problema pode ser tomado como um problema de escalonamento, sendo as tarefas representadas pelo “processamento” dos clientes e os recursos sendo as unidades de atendimento no interior do banco. O fato de não existir uma ordem pré-definida para execução das tarefas ilustra o caso de um ambiente de recursos diferente dos ambientes acima descritos. O objetivo a ser perseguido pela administração da agência pode girar em torno da minimização do tempo de processamento dos clientes, priorizando ou não clientes considerados especiais.

Os quatro exemplos acima descrevem situações comuns do dia a dia. Além destes, exemplos de determinação de seqüências ótimas de execução de tarefas possuem aplicações em muitos outros campos. Como exemplo, conforme o trabalho de Butchers et al. (2001), pode se desejar otimizar o escalonamento das tripulações de uma companhia aérea. Aubin (1992) propôs um problema de escalonamento de ambulâncias para a cidade de Montreal, no Canadá. Outro exemplo de aplicação real pode ser observado no trabalho de Portougal (1997).

Dada a variedade de situações práticas, existem muitos tipos diferentes de problemas de escalonamento. Convencionou-se representar na literatura os recursos por “máquinas”, mesmo quando estes são pessoas ou entidades. Incluem-se também como “máquinas” os estágios de produção necessários para efetiva execução de uma tarefa. Por sua vez, o termo “tarefa” é associado tanto aos produtos quanto aos serviços a serem produzidos e ofertados. Assim, a partir deste ponto, os recursos serão denominados por máquinas e os produtos e serviços por tarefas.

Problemas de escalonamento podem ser inicialmente divididos levando em consideração o caráter determinístico ou probabilístico das informações a respeito das tarefas.

Denominam-se problema de escalonamento determinístico os problemas onde as informações a respeito das tarefas são exatas e conhecidas. Em contrapartida, nos casos onde são conhecidas de antemão apenas as distribuições de probabilidade dos dados das tarefas, aplicam-se os problemas de escalonamento probabilístico (PINEDO, 2008, p. 7). No exemplo do aeroporto, o tempo que duas aeronaves idênticas levam para que seus passageiros desembarquem numa mesma ponte pode ser diferente, e poderá variar, por exemplo, tanto em função do número de passageiros que o avião transportou num vôo quanto pelo sucesso ou falha do acoplamento inicial do avião à ponte. Este caso, bem como o exemplo da agência bancária, seria melhor enquadrado a partir da modelagem como um problema de escalonamento probabilístico.

O problema de escalonamento selecionado como objeto deste estudo enquadra-se na categoria dos problemas de escalonamento determinísticos.

A notação convencional dos problemas de escalonamento determinísticos utiliza as letras *n* para representar o número de tarefas, *m* para o número de máquinas, *j* para uma determinada tarefa, *i* para uma determinada máquina, $p_{i,j}$ para o tempo de processamento de uma determinada tarefa *j* em uma máquina *i*, r_j para o tempo que a tarefa *j* encontra-se disponível para iniciar seu processamento (*release date*), d_j para a data de término prometida da tarefa *j* (*due date*) e w_j que representa uma medida de prioridade (*weight*) associada à tarefa *j*. As palavras “término”, “conclusão” e “finalização” são usadas neste trabalho como sinônimos e dizem respeito a um instante de tempo onde um dado processamento foi ou será finalizado. Apenas a título de esclarecimento, a notação utilizada nos problemas de escalonamento probabilísticos não é exatamente igual à anterior, sofrendo algumas modificações (por exemplo, a data que uma tarefa *j* encontra-se disponível para início do processamento é representado por R_j e a data de término prometida por D_j).

Ao se propor e resolver problemas de escalonamentos, existem características intrínsecas às situações reais que necessitam ser levadas em consideração para a exata definição do problema. Tais características podem retratar o ambiente de máquinas, eventuais restrições de processamento e o objetivo que se persegue. Neste sentido, a literatura sobre o tema (LAWLER; LENSTRA; RINNOOY KAN, 1982 apud PINEDO, 2008, p. 14), define um problema de escalonamento utilizando três campos: $\alpha \mid \beta \mid \gamma$, onde α define o ambiente de máquinas, β informa as restrições de processamento e γ expõe o objetivo a ser minimizado. Esta maneira possibilita, com poucos caracteres, transmitir as informações fundamentais da situação real, bem como o objetivo perseguido. A seguir, serão expostos exemplos de

possíveis entradas para cada um dos três campos. A notação utilizada encontra-se em parênteses. Notar a diversidade de aplicações possíveis.

Exemplos de possíveis entradas para o ambiente de máquinas, campo α:

Single Machine (1): Consiste no ambiente de máquinas mais simples possível, caracterizado pela existência de uma máquina apenas.

Identical machines in parallel (*Pm*): Ambiente de *m* máquinas iguais e paralelas. Uma variação deste ambiente pode acontecer quando as máquinas não possuem a mesma velocidade (*machines in parallel with diferent speeds*), cuja notação é *Qm*. Outra variação é aquela onde existem *m* máquinas diferentes em paralelo (o tempo de processamento das tarefas dependem das máquinas). O ambiente é conhecido por *unrelated machines in parallel*, sendo a notação para este caso igual à *Rm*.

Flow shop (*Fm*): Ambiente com *m* máquinas em série. Para ser concluída, cada tarefa *j* precisa ser processada em cada uma das *m* máquinas do ambiente. Todas as tarefas precisam seguir a mesma seqüência através das máquinas, sendo processadas inicialmente na máquina 1, em seguida na máquina 2, e assim sucessivamente até a máquina *m*.

Usualmente, a seqüência de execução das tarefas é representada na literatura através de um gráfico de *Gantt*. Um exemplo de escalonamento num ambiente *flow shop*, pode ser visto no Gráfico de *Gantt* abaixo, onde os tempos de processamento, em minutos, são: $p_{11} = 10, p_{21} = 15, p_{12} = 10, p_{22} = 10, p_{13} = 20$ e $p_{23} = 5$:

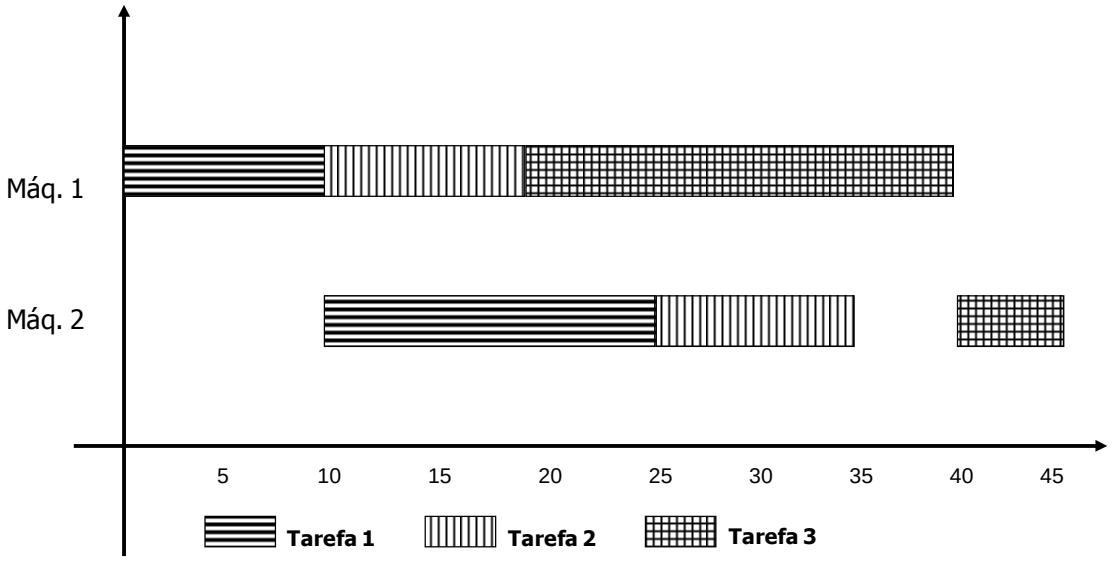

Figura 1 - Exemplo de escalonamento num ambiente de máquinas do tipo *flow shop*.

Nota-se que $j = 1$, $j = 2$ e $j = 3$ iniciam suas operações na máquina 1 e, após o processamento nesta máquina, seguem para a máquina 2.

Job shop (*Jm*): Ambiente com *m* máquinas onde cada tarefa *j*, possui uma seqüência pré-determinada ao longo das *m* máquinas do sistema. Neste sentido, o ambiente de *flow shop* pode ser encarado como um caso especial de ambiente *job shop*, onde todas as tarefas *j* possuem a mesma seqüência entre as *m* máquinas. Uma representação de escalonamento para o ambiente de *job shop* pode ser visto Figura 2. Os tempos de processamentos são os mesmos utilizados no exemplo do ambiente *flow shop*.

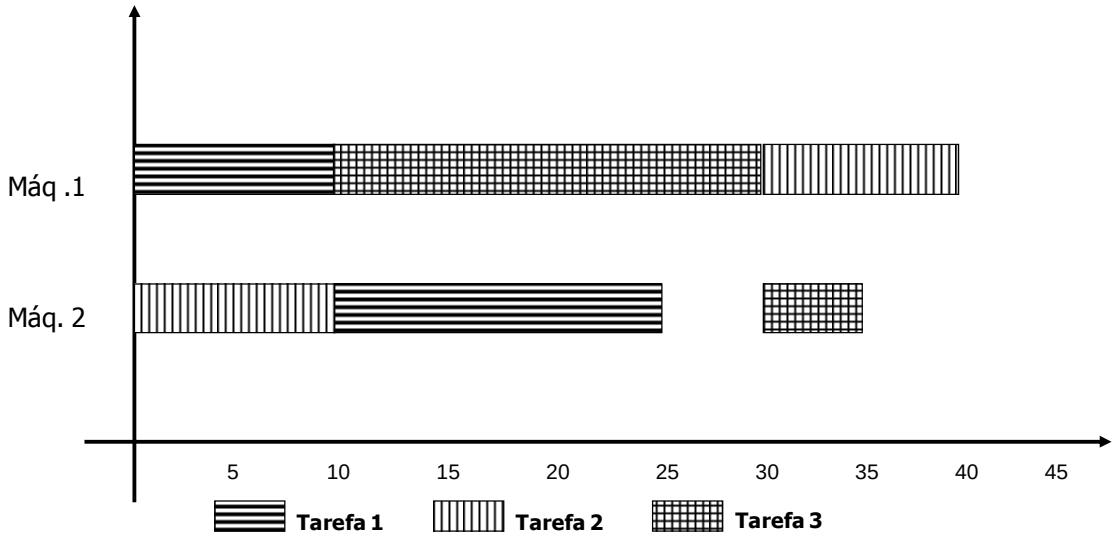

Figura 2 - Exemplo de escalonamento num ambiente de máquinas do tipo *job shop*.

Nota-se que $j = 1$ e $j = 3$ iniciam suas operações na máquina 1 e, após o processamento nesta, seguem para a máquina 2. Por sua vez, $j = 2$ inicia seu processamento no sistema pela máquina 2 e logo após segue para a máquina 1.

Open shop (*Om*): O ambiente de máquinas típico de *open shop* caracteriza-se pelo fato de não existir uma seqüência pré-definida a ser seguida por cada tarefa *j*. Posto de outra forma, é fornecido ao planejador liberdade na escolha da seqüência a ser seguida por cada tarefa *j* ao longo do sistema.

Exemplos de possíveis entradas para as restrições de processamento, campo β:

Release dates (r_j): Conforme exposto anteriormente, informa que a tarefa *j* não pode ser iniciada no ambiente de máquinas antes da data especificada em r_j. Se o campo β não contiver a informação r_j para determinada tarefa *j*, isto quer dizer que a tarefa pode ser

iniciada a qualquer momento. Esta restrição de processamento pode acontecer em qualquer ambiente de máquinas.

No caso de haver datas de términos prometidas para finalização das tarefas (definida anteriormente como *due date*, e representado por d_j), esta entrada não aparece no campo β. Como será visto, o tipo de função objetivo usada fornece informações suficientes a respeito da existência ou não de datas prometidas.

Permutation (*prmu*): Informa que a fila de tarefas em frente às máquinas opera sob o regime de primeiro que entra é o primeiro que saí. Ou seja, não é permitido que determinada tarefa que tenha sido processada após outra em uma determinada máquina, seja processada antes desta na máquina seguinte. É uma restrição típica do ambiente de máquinas *flow shop*.

Blocking (*block*): Em ambientes de máquinas onde há limitação de espaço entre sucessivas máquinas, pode acontecer que o espaço esteja completamente utilizado e, dessa forma, a máquina imediatamente anterior ao espaço não se encontra autorizada a completar seu atual processamento, até que haja espaço para o envio da tarefa à máquina seguinte. É uma restrição característica dos ambientes *flow shop*.

Preemptions (*prmp*): Informa que não é necessário o fato de, iniciado o processamento de uma tarefa *j* numa dada máquina *i*, este seja executado até a sua finalização. Ou seja, o processamento de uma tarefa *j* por uma determinada máquina *i* poderá ser interrompido a qualquer momento. Esta mesma máquina *i* poderá então executar o processamento de outra tarefa, exemplo *j+1*, e depois finalizar o processamento da tarefa *j*.

No-wait (*nwt*): Informa que as *n* tarefas não podem aguardar pelo processamento entre duas máquinas sucessivas. Ou seja, após o processamento de uma dada tarefa *j* na primeira máquina do sistema, a máquina seguinte deve estar disponível para que esta tarefa *j* inicie imediatamente seu processamento, sendo assim até que esta tarefa deixe o sistema.

Outros exemplos de restrições de processamento existem quando há relações obrigatórias de precedência entre as tarefas (*precedence constraints*), quando no caso do ambiente tipo *job shop* uma dada tarefa *j* necessita passar por uma máquina mais de uma vez (*recirculation*) ou quando uma dada máquina não está continuamente disponível (*breakdowns*). Nestes casos, a entrada no campo β é representada por *prec*, *rcrc* e *brkdwn* respectivamente.

O último, campo γ, informa o objetivo que se deseja minimizar. Segundo Pinedo (2008, p. 18), nesta notação, o objetivo a ser minimizado é sempre uma função dos tempos de término das tarefas o que, obviamente, depende da seqüência das tarefas.

Antes de expor exemplos de possíveis entradas para esse campo, é necessário definir alguns elementos: o tempo de finalização de uma dada tarefa *j* em uma máquina *i*, representado por $C_{i,j}$, consiste no tempo que a tarefa foi finalizada na máquina *i*. Já o tempo de finalização da tarefa *j*, representado por C_j, define o tempo que a tarefa deixou o sistema ou seja, é o tempo decorrido após o processamento na última máquina. Para um ambiente do tipo *flow shop*, é imediato que:

$$C_{mj} = C_j \quad \forall \quad j = 1, \ldots . n$$

Define-se ainda:

Lateness (L_j) de uma determinada tarefa *j* como sendo a diferença entre o tempo que a tarefa *j* incorre até deixar o sistema (C_j) e a data de término prometida (d_j). Algebricamente tem-se:

$$L_j = C_j - d_j$$

Tardiness (T_j): quando o *Lateness* de uma tarefa *j* nunca assume valores negativos, tem-se o *Tardiness*. Trata-se do atraso de uma tarefa *j*. Algebricamente apresenta-se da seguinte forma:

$$T_j = Max(C_j - d_j, 0) = Max(L_j, 0)$$

De posse das informações acima, seguem o nome e a descrição de possíveis funções objetivos a serem minimizadas. Seguindo o modelo utilizado para os campos α e β , a notação para o campo γ segue em parênteses:

Makespan (C_{max}): é definido como o tempo de término da última tarefa a deixar o sistema. Segundo Pinedo (2008, p. 18), um mínimo *Makespan* implica usualmente numa boa utilização do sistema.

Maximum Lateness (L_{max}): ou pior violação da data de término prometida (*due-date*).

Total weighted completion time ($\sum w_j C_J$) ou soma ponderada dos tempos de conclusão de todas as *n* tarefas. É um indicador de perda total do sistema.

Total weighted tardiness ($\sum w_j T_J$) ou soma ponderada da diferença entre o tempo de conclusão e a data de término prometida. Representa uma função de custo mais genérica do

que a soma ponderada dos tempos de conclusão, dado que leva em consideração além dos pesos, a data de entrega prometida (*due date*).

Uma maneira de resolver problemas de escalonamento, bem como outros problemas de otimização, seria enumerando todas as soluções possíveis e logo após selecionar a melhor destas segundo o critério eleito pelo tomador de decisão. Por exemplo, o número de soluções (seqüências) possíveis e viáveis para um problema de escalonamento *flow shop*, permutacional, com duas máquinas e *n* tarefas é igual a $n!$. A Tabela 1 exemplifica para um dado número de tarefas *n*, o número de seqüências a serem comparadas

Problema de escalonamento com numero "n" de tarefas	Número de seqüências possíveis e viáveis a serem comparadas
5	$1{,}2\ x\ 10^{2}$
10	$3{,}63 \times 10^{10}$
20	$2{,}43\ x\ 10^{18}$
30	$2{,}65\ x\ 10^{32}$
40	$8{,}16\ x\ 10^{47}$
50	$3{,}04\ x\ 10^{64}$
100	$9{,}33\ x\ 10^{157}$
1000	$4{,}02\ x\ 10^{2567}$

Tabela 1 - Número de soluções utilizando o método da enumeração, para um número *n* de tarefas.

Uma das conclusões que podem ser retiradas da Tabela 1 é a de que apenas problemas com reduzido número de tarefas podem ser resolvidos computacionalmente a partir da enumeração completa das soluções. Entretanto, é fácil diagnosticar que aplicações reais podem facilmente envolver mais de vinte tarefas, fato que torna impraticável o método de enumeração completa.

Outra maneira de encontrar a melhor seqüência seria desenvolver algoritmos para cada tipo de problema. Como exemplo, para solucionar o problema de escalonamento $F_2||C_{max}$ (ambiente tipo *flow shop* com duas máquinas, tendo como objetivo a minimização do tempo de finalização da última tarefa) pode ser empregado a regra de Johnson (1954), que consiste num algoritmo de tempo polinomial. Cormen et al. (2002), define um algoritmo de tempo

polinomial como sendo aqueles em que, sobre entradas de tamanho *n*, seu tempo de execução no pior caso é $O(n^k)$ para alguma constante *k*.

Infelizmente, nem todos os problemas de escalonamento possuem algoritmos de tempo polinomial para serem empregados na sua resolução. Este fato, sem dúvida, influencia diretamente no tempo necessário para o encontro da solução ótima. Os problemas que não podem ser resolvidos em tempo polinomial, porém são verificáveis neste tempo são conhecidos como problemas da classe $\mathcal{NP}$. A questão de ser "resolvido" ou "verificável" pode ser ilustrada por meio de um simples exemplo: suponha o número 6.760.203. Provar que este número pode ser obtido a partir do produto de dois outros números inteiros poderá demandar tempo considerável. Contudo, se for informado que o produto entre 2.501 e 2.703 dá origem ao número 6.760.203, verificar esta afirmativa é simples, e pode ser feita numa simples operação de multiplicação.

Por último, diz-se que determinado problema está na classe dos $\mathcal{NP}$-completos se ele pertence à $\mathcal{NP}$ e é tão difícil quanto qualquer outro problema em $\mathcal{NP}$ (este assunto será abordado com maior profundidade adiante, quando será feita uma revisão dos conceitos de complexidade computacional).

Muitos dos problemas de escalonamento estão na classe dos problemas $\mathcal{NP}$-completos. Um deles é o $F_2||\sum w_j T_j$. Se não é eficiente empregar o método da enumeração para resolver o problema $F_2||\sum w_j T_j$ com, por exemplo, n = 20, e também não existem algoritmos de tempo polinomial disponíveis, de que maneira este problema pode ser resolvido num tempo razoável? A definição do objetivo deste trabalho ajudará a responder a esta questão.

1.2 OBJETIVOS DO TRABALHO

Para Marconi e Lakatos (2007, p. 158), "toda pesquisa deve ter um objetivo determinado para saber o que se vai procurar e o que se pretende alcançar". Segundo Ander-Egg (1978, p. 62, apud MARCONI; LAKATOS, 2007, p. 159), deve-se partir "de um objetivo limitado e claramente definido, sejam estudos formulativos, descritivos ou de verificação de hipóteses".

O objetivo principal deste trabalho é elaborar formulações de programação inteira para o problema $F_2||\sum w_j T_j$ que possibilitem resolver geralmente problemas da ordem de quarenta ou cinqüenta tarefas dentro de um tempo computacional aceitável.

A qualidade da formulação proposta é fundamental para que a resposta seja encontrada dentro de um tempo razoável. A razão para este fato será justificada ao longo deste trabalho.

São propostos dois grupos de formulações.

- O primeiro grupo contém uma formulação com variáveis binárias x_{jit} (que assumem valor unitário se a tarefa *j* finaliza seu processamento na máquina *i* no tempo *t*) e outra formulação com variáveis binárias x_{jt} (que assumem valor unitário se a tarefa *j* tem seu processamento finalizado na última máquina no tempo *t*). Uma terceira formulação, bastante semelhante a anterior, emprega além das variáveis x_{jt}, variáveis auxiliares y_{jt} que representam o quanto uma determinada tarefa *j* já foi processada na primeira máquina até o momento *t*.
- O segundo grupo não utilizará o índice de tempo *t*. Por sua vez, utilizará um índice de posição que representa a ordem que a tarefa entrou no sistema em relação às demais. Esta mudança tem como meta principal reduzir o número de variáveis empregadas na formulação, sendo necessário em casos em que a discretização nos tempos de execução das tarefas é fina.

Uma tabela de resultados irá registrar o desempenho de cada formulação em termos do valor da relaxação linear, tempo utilizado e distância máxima em relação ao valor ótimo nas instâncias selecionadas.

Além das formulações de programação inteira, serão propostos algoritmos heurísticos, alguns dos quais para auxiliar o algoritmo de *branch and bound* a encontrar uma solução viável inicial (limite primal).

O objetivo secundário é o aprofundamento de conhecimentos em programação inteira, uma sub-área da programação matemática, sendo, esta última, um dos métodos da pesquisa operacional. Buscou-se desenvolver a habilidade de formular problemas de programação inteira.

Para realização dos testes nas formulações propostas, foi utilizado o software CPLEX, versão 11.0 (solucionador de problemas de programação linear, programação inteira e programação mista, desenvolvido pela empresa ILOG). O CPLEX foi utilizado juntamente

com o UFFLP, que consiste numa API (*Application Program Interface*) desenvolvido na Universidade Federal Fluminense.

1.3 METODOLOGIA DA PESQUISA

Segundo Ander-Egg (1978, p. 28, apud MARCONI; LAKATOS 2007, p. 157), o trabalho de pesquisa consiste em um "procedimento reflexivo sistemático, controlado e crítico, que permite descobrir novos fatos ou dados, relações ou leis, em qualquer campo do conhecimento". Com base nisso, Marconi e Lakatos (2007, p. 157) relatam que "a pesquisa, portanto, é um procedimento formal, com método de pensamento reflexivo, que requer um tratamento científico e se constitui no caminho para conhecer a realidade ou para descobrir verdades parciais".

Para Booth et al. (2005, p. 7), "pesquisar é simplesmente reunir informações necessárias para encontrar resposta para uma pergunta e assim chegar à solução de um problema". No tocante ao relatório de pesquisa, estes autores asseveram que, "redigir um relatório de pesquisa, enfim, é simplesmente uma questão de pensar por escrito", e defendem que "pensar por escrito pode ser mais meticuloso, sistemático, abrangente, completo e mais adequado àqueles que têm pontos de vistas diferentes – mais ponderado – do que quase todas as outras formas de pensar".

Para o desenvolvimento desta pesquisa, foram utilizados os passos expostos por Marconi e Lakatos (2007, p. 157), descritos a seguir na ordem de sua execução:

1. Seleção do problema para investigação;
2. Definição do problema;
3. Levantamento de hipóteses de trabalho;
4. Coleta, sistematização e classificação dos dados;
5. Análise e interpretação dos dados; e
6. Relatório do resultado da pesquisa

Neste sentido, o Capítulo 2 expõe a revisão da literatura sobre as classes de complexidade computacional, bem como discorre sobre as técnicas de programação linear e programação inteira. Além do objetivo de expor conceitos fundamentais para a elaboração deste trabalho, preocupa-se em demonstrar a importância e a evolução histórica de tópicos

relativos à programação matemática e a teoria da complexidade computacional. O Capítulo 3 introduzirá as formulações desenvolvidas e analisadas no trabalho, bem como os testes de desempenho empregados. O Capítulo 4 traz uma síntese das conclusões do trabalho, bem como sugestões para futuros trabalhos de pesquisa na área.

2 REVISÃO DA LITERATURA

2.1 A CLASSE DE PROBLEMAS $\mathcal{NP}$-COMPLETOS

Na introdução deste trabalho, definiu-se como um problema da classe dos problemas $\mathcal{NP}$-completos como sendo todos àqueles que pertencem à $\mathcal{NP}$ e são tão difíceis quanto qualquer outro problema em $\mathcal{NP}$. Afirmou-se também que $F_2||\sum w_j T_j$ é um problema da classe dos problemas $\mathcal{NP}$-completos. Neste momento, será feita a prova desta afirmação. Porém, antes disso, faz-se necessário revisar alguns conceitos, começando pelo conceito de algoritmo, exposto por Cormen et al. (2002, p. 3):

> Informalmente, um **algoritmo** é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como **entrada** e produz algum valor ou conjunto de valores como **saída**. Portanto, um algoritmo é uma seqüência de passos computacionais que transformam a entrada na saída.

Os algoritmos são utilizados para resolver uma série de problemas computacionais. Tais problemas podem estar relacionados a diversas questões práticas, que variam desde o projeto genoma humano até os problemas de tomada de decisão empresariais e governamentais.

Fundamental também é o conceito de "instância" de um problema que, segundo Cormen et al. (2002, p. 3) "uma instância de um problema consiste na entrada (que satisfaz a quaisquer restrições impostas no enunciado do problema) necessária para se calcular uma solução para o problema".

Uma medida de eficiência dos algoritmos é a velocidade de execução, ou seja, quanto tempo um algoritmo demora para produzir seu resultado. Por sua vez, existem problemas para

os quais não se conhece nenhum algoritmo eficiente. Um subconjunto interessante desses problemas é conhecido por problemas "$\mathcal{NP}$-completos" (CORMEN et al., 2002, p. 6). Uma lista de razões fazem com que estes tipos de problemas sejam importantes:

> Por que os problemas $\mathcal{NP}$-completos são interessantes? Primeiro, embora ainda não tenha sido encontrado nenhum algoritmo eficiente para um problema $\mathcal{NP}$-completo, ninguém jamais provou que não é possível existir um algoritmo eficiente para este fim. Em outras palavras, desconhecemos se existem ou não algoritmos eficientes para problemas $\mathcal{NP}$-completos. Em segundo lugar, o conjunto de problemas $\mathcal{NP}$-completos tem a propriedade notável de que, se existe um algoritmo eficiente para qualquer um deles, então existem algoritmos eficientes para todos. Esse relacionamento entre problemas $\mathcal{NP}$-completos torna a falta de soluções eficientes ainda mais torturante. Em terceiro lugar, vários problemas $\mathcal{NP}$-completos são semelhantes, mas não idênticos, a problemas para os quais conhecemos algoritmos eficientes. Uma pequena mudança no enunciado do problema pode provocar uma grande alteração na eficiência do melhor algoritmo conhecido (CORMEN et al., 2002, p. 6).

> É valioso conhecer os problemas $\mathcal{NP}$-completos, porque alguns deles surgem com freqüência surpreendente em aplicações reais. Se for chamado a produzir um algoritmo eficiente para um problema $\mathcal{NP}$-completo, é possível que você perca muito tempo em uma busca infrutífera. Por outro lado, se conseguir mostrar que o problema é $\mathcal{NP}$-completo, você poderá em vez disso dedicar seu tempo ao desenvolvimento de um algoritmo eficiente que ofereça uma boa solução, embora não seja a melhor possível (CORMEN et al., 2002, p. 6).

Nota-se acima o emprego acentuado da palavra "eficiente" associada a algoritmo. A área de conhecimento da complexidade computacional realiza uma distinção entre problemas de "otimização" e problemas de "decisão". Isso não é comum nas bibliografias de pesquisa operacional, onde muitas vezes trata-se um problema de otimização como sinônimo a um problema de "tomada de decisão", etc. De qualquer forma, esta distinção é importante para que se possa definir a complexidade de um determinado problema de otimização.

Seja o problema de otimização a seguir (os problemas de otimização serão definidos com maior formalidade no próximo item):

$$max\{cx : x \in S\}$$

Seja uma instância do problema, tal que c esteja associado aos custos de uma representação linear padrão das soluções x em S. Tem-se o seguinte problema de decisão associado ao problema de otimização:

Existe algum $x \in S$ com valor $cx \geq k$? A resposta é "sim" ou "não".

A relação entre um problema de otimização e seu problema de decisão torna-se muito útil quando se tem por objetivo mostrar que o problema de otimização é “difícil”. Isso acontece porque o problema de decisão é de certo modo “mais fácil” ou, pelo menos, “não mais difícil” que o problema de otimização relacionado (CORMEN et al., 2002, p. 765).

É necessário definir de maneira mais formal as classes de complexidade computacional de um problema. As definições abaixo foram retiradas de Wolsey (1998, p. 83):

Definição 1: Seja X a instância de um problema. O comprimento de entrada $L = L(X)$ é o tamanho da representação binária da instância.

Definição 2: Dado um problema P, um algoritmo A para o problema e uma instância X seja $f_A(X)$ o número de cálculos elementares (adição, subtração, divisão, comparações, etc) requeridos para que o algoritmo A resolva a instancia X. $f_A^*(l) = max_X\{f_A(X): L(X) = l\}$ é o tempo de execução do algoritmo A. O algoritmo A é polinomial para o problema P se $f_A^*(l) = O(l^p)$ para algum inteiro positivo p.

Definição 3: $\mathcal{NP}$ é a classe dos problemas de decisão que possuem a seguinte propriedade: para toda a instância onde a resposta é sim, existe uma “curta” (leia-se polinomial) prova do “sim”.

Definição 4: $\mathcal{P}$ é a classe de problemas de decisão que estão em $\mathcal{NP}$ e que para estes existe um algoritmo de tempo polinomial.

Para definir formalmente outras classes de complexidade, é necessário expor o conceito de redução polinomial:

Definição 5: Se os problemas $P\ e\ Q\ \in \mathcal{NP}$, e uma instância do problema P, pode ser convertida em tempo polinomial numa instância do problema Q, então P pode ser reduzido polinomialmente a Q. Representa-se da seguinte forma: $P \propto Q$ e diz-se que P reduz a Q.

O conceito de redução polinomial é importante no sentido de que se existe um algoritmo eficiente que solucione um dado problema Q, este logicamente poderia ser empregado na resolução de um problema P, se P reduz a Q.

Definem-se agora as últimas duas classes de complexidade. As definições abaixo também foram extraídas de Wolsey (1998, p. 84 e 88):

Definição 6: $\mathcal{NPC}$, a classe dos problemas $\mathcal{NP}$-completos, é um subconjunto de problemas $P \in \mathcal{NP}$, que, para todo $Q \in \mathcal{NP}$, Q é polinomialmente reduzido a P.

Conforme exposto na introdução, diz-se que um problema está na classe de problemas $\mathcal{NP}$-completos se ele está em $\mathcal{NP}$ e é tão "difícil" quanto qualquer outro problema em $\mathcal{NP}$. Mostrar que um problema está na classe de problemas $\mathcal{NP}$-completos consiste em mostrar o quanto ele é difícil e não o quanto ele é fácil (CORMEN et al., 2002, p. 764-766).

Definição 7: Um problema de otimização cuja sua versão de decisão associada encontra-se na classe dos problemas $\mathcal{NP}$-completos é denominado $\mathcal{NP}$-difícil.

Definidas as classes de complexidade, é chegado o momento de provar que $F_2||\sum w_jT_j$ é $\mathcal{NP}$-difícil. Pinedo (2008, p. 26) assevera que "um algoritmo utilizado num problema de escalonamento pode, muitas vezes, ser aplicado em outro problema deste tipo, quando este último é encarado como um caso especial do primeiro". Nota-se que o autor está baseado no conceito de redução polinomial. Por exemplo, o problema $1||\sum w_jT_j$ é um caso especial do problema $F_2||\sum w_jT_j$, dado que os tempos de processamento na segunda máquina podem ser nulos. Assim, $1||\sum w_jT_j$ se reduza $F_2||\sum w_jT_j$. Um algoritmo que resolva o problema $F_2||\sum w_jT_j$ poderá ser utilizado para resolver, por exemplo: $1||\sum w_jT_j$, $1||\sum T_j$ ou $F_2||\sum T_j$,

Teorema 1: $F_2||\sum w_jT_j$ é $\mathcal{NP}$-difícil.

A prova do Teorema 1, segundo o qual $F_2||\sum w_jT_j$ é $\mathcal{NP}$-difícil será feita em duas etapas:

1º: Analisar o problema $1||\sum w_jT_j$ e provar que este é $\mathcal{NP}$-difícil;

2º: Verificar que $1||\sum w_jT_j \propto F_2||\sum w_jT_j$. Assim, se $1||\sum w_jT_j$ é $\mathcal{NP}$-difícil então $F_2||\sum w_jT_j$ também será.

A prova de que $1||\sum w_jT_j$ é $\mathcal{NP}$-difícil foi estabelecida por Lawler (1977) bem como por Lenstra, Rinnoy Kan e Brucker (1977). Apoiando-se nestes trabalhos, Pinedo (2008, p. 58) apresenta e justifica o teorema 2 a seguir:

Teorema 2: O problema $1||\sum w_jT_j$ é $\mathcal{NP}$-difícil.

Para provar o Teorema 2, será utilizada a definição de um problema sabidamente $\mathcal{NP}$-difícil (GAREY; JOHNSON, 1979), denominado problema de 3-PARTIÇÃO. A prova será feita a partir do fato de que o problema de 3-PARTIÇÃO reduz a $1||\sum w_jT_j$.

O problema de 3-PARTIÇÃO é definido da seguinte maneira: seja um conjunto A de 3t número inteiros, $a_1, a_2, \ldots, a_{3t}$, tais que:

$$\sum_{t=1}^{3t} a_j = tb;$$

$$\frac{b}{4} < a_j < \frac{b}{2} \ \forall j = 1, 2, \ldots, 3t$$

A questão que se coloca é: "É possível particionar o conjunto A em t subconjuntos disjuntos $A_1, A_2, \ldots A_t$, tais que para $1 \leq i \leq t$ tem-se:

$$\sum_{a \in A_J} a = b$$

A redução do problema de 3-PARTIÇÃO a $1||\sum w_j T_j$ é baseada na transformação a seguir. Notar os valores assumidos pelos parâmetros *n*, w_j e d_j .

$$n = 4t - 1$$

$$d_j = 0, \ p_j = a_j, w_j = a_j, \ \forall \ j = 1, \ldots \ldots 3t,$$

$$d_j = (j - 3t)(b + 1), p_j = 1, w_j = 2, \ \forall \ j = 3t + 1, \ldots \ldots, 4t - 1.$$

Tomando:

$$z = \sum_{1 \leq j \leq k \leq 3t} a_j a_k + \frac{1}{2}(t - 1)t_b.$$

Pode ser demonstrado que existe um escalonamento com valor da função objetivo igual a z se e somente se existe uma solução para o problema de 3-PARTIÇÃO. As primeiras $3t$ tarefas apresentam quociente w_j / p_j igual a 1 (pois, conforme a transformação acima, para as primeiras $3t$ tarefas tem-se que $p_j = a_j$ e $w_j = a_j$) e $d_j = 0$. Existem $t - 1$ tarefas com quociente w_j / p_j igual a 2 (pois, no segundo caso, para as $t - 1$ tarefas restantes tem-se que

p_j= 1 e w_j = 2), estando suas datas de término prometidas em $(j - 3t).(b + 1)$. A solução com valor z pode ser obtida se as $t - 1$ tarefas podem ser processadas exatamente durante os $t - 1$ intervalos:

$$[b,\ b + 1], [2b + 1, 2b + 2],..., [(t - 1)b + t - 2, (t - 1)b + t - 1]$$

De forma a adaptar as $t - 1$ tarefas nos $t - 1$ intervalos, as primeiras $3t$ tarefas devem ser particionados dentro de t subconjuntos, que possuem três tarefas cada um, com a soma dos três tempos de processamento em cada subconjunto sendo igual a b. Neste caso a soma ponderada dos atrasos é igual a z.

Se esta partição não é possível, então existe no mínimo um subconjunto onde a soma dos três tempos de processamento é maior do que b ou um subconjunto onde a soma dos três tempos de processamento é inferior a b. Neste caso, prova-se que o custo obtido é necessariamente maior do que *z*.

Por último, como uma instância do problema $1||\sum w_j T_j$ pode ser transformada em tempo polinomial por redução numa instância do problema $F_2||\sum w_j T_j$, onde esta última apresente p_{2j}= 0 para toda tarefa *j*, tem-se que $F_2||\sum w_j T_j$ é $\mathcal{NP}$-difícil.

Permanece em aberto a questão de como obter a resposta para um problema de escalonamento $F_2||\sum w_j T_j$. Como demonstrado, este problema situa-se na classe dos problemas $\mathcal{NP}$-completos e sua resolução, por si só, já estabelece um desafio (para instâncias com $n = 20$ por exemplo), independentemente do tempo empregado.

Wolsey (1998, p. 87 e 88) fornece uma explicação, a respeito das conseqüências de $\mathcal{P} = \mathcal{NP}$ ou $\mathcal{P} \neq \mathcal{NP}$ e dos desafios que se colocam ao tratar os problemas $\mathcal{NP}$-completos.

> Para muitos problemas de interesse, já foi demonstrado que estes estão na classe $\mathcal{P}$ ou $\mathcal{NPC}$. Mais que isso, ninguém obteve sucesso ainda em provar que $\mathcal{P} = \mathcal{NP}$ ou mostrar que $\mathcal{P} \neq \mathcal{NP}$. Entretanto, dado o imenso número de problemas em $\mathcal{NP}$ para os quais nenhum algoritmo de tempo polinomial foi encontrado, torna-se prática a hipótese de que $\mathcal{P} \neq \mathcal{NP}$. Então, como nós devemos interpretar as observações acima? A primeira observação importante preocupa-se com a classe $\mathcal{NP}$. Tipicamente, problemas nesta classe possuem um tamanho (exponencialmente grande) para o conjunto de soluções viáveis e estes problemas poderiam, em teoria, serem resolvidos por enumeração das soluções viáveis. Porém, isso é impraticável para instancias com qualquer tamanho considerável.
> Um pessimista poderia dizer que para a maioria dos problemas que são difíceis (seus problemas de decisão estão em $\mathcal{NPC}$), não temos esperança em resolver instâncias

> de grandes dimensões. (porque no pior caso nos não podemos fazer mais do que enumerar) e por isso deve-se desistir.
> Um matemático (otimista) poderia tornar-se famoso provando que $\mathcal{P} = \mathcal{NP}$
> Um matemático (pessimista) poderia tornar-se famoso provando que $\mathcal{P} \neq \mathcal{NP}$.
> Um matemático (pensativo) talvez decida fazer uma pergunta diferente: seria possível eu encontrar um algoritmo onde é garantido encontrar uma solução "perto da ótima" em tempo polinomial em todos os casos?
> Um probabilista (pensativo) talvez também faça uma pergunta diferente: Posso descobrir um algoritmo que é, com alta probabilidade, executado em tempo polinomial e garantir que seja encontrada uma solução ótima ou (próxima de ótima) com grande probabilidade?
> Um engenheiro iria começar a procurar por um algoritmo heurístico que produzisse soluções utilizáveis.
> Seu chefe poderia dizer: Eu não me importo nem um pouco com programação inteira, você deve se preocupar com nosso problema de escalonamento. Me de uma programação de produção factível para amanhã em que a encomenda de William Brown and Daughters' esteja fora da porta às 16:00hs.
> Um professor empenhado talvez dissesse: Ótimo, anteriormente eu estava tentando desenvolver um algoritmo que resolvesse todos os problemas de programação inteira e publicando um artigo a cada dois anos explicando porque eu não obtive sucesso. Agora eu sei que eu poderia também estudar cada problema da classe $\mathcal{NP}$ individualmente. Como existem milhares deles, eu deveria ser capaz de escrever vinte artigos por ano.
> Não é necessário dizer que estão todos quase certos. Não existe solução fácil e rápida, mas os problemas não desaparecem, e cada vez mais e mais fascinantes e importantes problemas práticos estão sendo formulados como problemas de programação inteira. Assim, embora a teoria de $\mathcal{NP}$-completude, usando uma apropriada combinação de teoria, algoritmos, experiência e intensivos cálculos, verificáveis boas soluções para grandes instâncias podem e devem ser encontradas.

O texto acima endossa o objetivo desta dissertação. Tomado um problema situado na classe dos $\mathcal{NP}$-completos (no caso, $F_2||\sum w_j T_j$), torna-se necessário o desenvolvimento de meios que viabilizem a solução de instâncias com elevado número de tarefas em tempos computacionais razoáveis. Trata-se de um esforço de pesquisa que é empreendido mundo afora por outros pesquisadores, para este e outros problemas $\mathcal{NP}$-completos.

Um meio de resolver este problema é utilizar a técnica de programação inteira (como citado acima, esta técnica tem sido utilizada para resolver muito problemas práticos importantes). A seguir, serão expostos os conceitos de formulação além dos conceitos básicos das técnicas de Programação Linear e Programação Inteira, bem como os algoritmos por elas empregados.

2.2 PROGRAMAÇÃO LINEAR

Segundo Colin (2007, p. 2), "a Programação Linear é uma das principais descobertas da matemática aplicada. Se tomados em consideração os benefícios econômicos gerados ao

ser humano, é bastante provável que a Programação Linear seja a maior descoberta da matemática aplicada de todos os tempos".

Para Puccini et al. (1989, p. 47) a técnica de Programação Linear é muito importante para a área de pesquisa operacional tanto que, a publicação por George Dantzig, em 1947, do método simplex para a resolução de problemas de programação linear é considerada como sendo o "marco definitivo na afirmação da pesquisa operacional".

Dada a relevância da programação linear para a área de pesquisa operacional, segue abaixo um pouco da história do surgimento desta técnica da matemática aplicada (e, por conseqüência, da pesquisa operacional). O fragmento abaixo foi extraído de Colin (2007, p. 2 e 3):

> É intrigante notar que, embora a matemática seja uma ciência relativamente madura desde pelo menos o século XVIII, a PL da forma como a conhecemos hoje em dia foi iniciada apenas após a Segunda Guerra Mundial especialmente por George B. Dantzig (1914-2005). Entre 1941 e 1945 Dantzig trabalhou no Pentágono, órgão de defesa americano, como especialista em planejamento e programação de atividades militares, época que trabalhava intensivamente com calculadoras de mesa (Dantzig, 1991, pp.19-31). Posteriormente, ele continuou trabalhando na mesma organização em outras funções como conselheiro em matemática da Força Aérea. Sabendo que Dantzig estava interessado em mudar de emprego, D Hitchcock e M. Wood, seus colegas de Pentágono, o desafiaram a que desenvolvesse um método que "mecanizasse" o processo de planejamento que era utilizado na época. Na ocasião, Dantzig estava fascinado com o trabalho de Wassily Leontief (1906-1999), russo que havia proposto em 1932 uma estrutura matricial denominada "Modelo Interindustrial de Entrada-Saída" que lhe renderia em 1973 o Prêmio Nobel de Economia. Aparentemente o modelo de Leontief serviu de inspiração para Dantzig, mas ele necessitava de algumas características para a solução do problema da Força Aérea: ser dinâmico (capacidade de adequar mudanças ao longo do tempo), ter flexibilidade para poder escolher entre diversas alternativas de alocação de recursos e, finalmente, ser computacionalmente tratável. O primeiro grande avanço de Dantzig foi a formulação do problema como a minimização de uma função linear sujeita a equações e inequações também lineares. O próximo passo de Dantzig foi solucionar o problema. A primeira idéia foi procurar ajuda junto a economistas matemáticos, tendo em vista que seu problema era de alocação de recursos escassos e ele imaginava que o problema já poderia estar resolvido. A primeira opção de Dantzig foi o professor Tjalling C. Koopmans (1910-1985), holandês que ficou muito interessado no problema, mas também trouxe a má notícia de que não havia uma solução conhecida até aquela data, 1947. Após a confirmação de que os economistas não tinham um método de solução, Dantzig resolveu por si só tentar uma solução, iniciativa que acabou culminando na criação do algoritmo simplex. O algoritmo simplex é o mecanismo matemático que serve para resolver problemas de programação linear. Desde então, e até hoje, o algoritmo simplex se mostrou extremamente poderoso nos quesitos mais importantes em praticamente todas as aplicações: simplicidade, rapidez e precisão na solução computacional. Na mesma época, Dantzig continuou procurando por outras sugestões de solução do problema e o apresentou para John von Neumann (1903-1957), nascido na Hungria, naquela ocasião fazendo pesquisas na Universidade de Princeton e considerado por muitos o maior matemático da época. John Von Neumann adquiriu fama por resolver de cabeça problemas para os quais outros matemáticos utilizavam papel e lápis, ou mesmo calculadoras (Bergamini, 1969, p.175). Von Neumann foi o principal criador da Teoria dos Jogos e esteve envolvido em muitos dos principais avanços científicos

> do século XX como a Física Atômica, a Mecânica Quântica, o computador digital e vários outros campos da Matemática Pura e Aplicada. No encontro, após uma breve exposição de Dantzig sobre o problema, Von Neuman discursou por uma hora e meia sobre teoria matemática de programas lineares, tendo em vista que ele encontrara uma analogia entre o problema apresentado e sua Teoria dos Jogos, publicada entre 1928 e 1944. Questões que seriam de suma importância para o desenvolvimento da Programação Linear foram discutidas pela primeira vez naquele encontro. A pedido de Dantzig, após algum tempo, Von Neumann propôs uma solução para o problema que se mostrou de capacidade muito inferior ao algoritmo simplex. Apresentando o problema a vários dos principais pesquisadores da época, ajudado pelo aparecimento do computador e considerando as diversas aplicações dos modelos, a difusão da Programação Linear foi ampla e rápida. Ela não só floresceu como também serviu de inspiração e base para diversas outras disciplinas, como a programação não-linear, a programação inteira e outras técnicas de otimização. A primeira aplicação não-militar aconteceu em 1952, com a mistura ótima de produtos na produção de gasolina. O primeiro código computacional comercial foi criado em 1954 na Rand Corporation por William Orchard-Hays e possivelmente a primeira publicação brasileira sobre Programação Linear é devida a Leme (1956). Desde então, têm sido incontáveis o número e a diversidade de aplicações que a técnica permite.

De acordo com o fragmento, o primeiro passo de Dantzig foi formular o problema como a minimização de uma função linear sujeita a equações e inequações também lineares. É comum na literatura encontrar a sigla PL denominando tanto a técnica de programação linear quanto um problema específico (sendo que isto ocorre também com a técnica de programação inteira, que será vista mais a frente). Neste trabalho, a sigla PL poderá significar tanto um problema em específico quanto a técnica propriamente dita.

Traduzir a descrição do problema numa formulação deve ser feita de forma sistemática e, uma clara distinção deve ser estabelecida entre os dados da instância do problema e as variáveis usadas na formulação (WOLSEY, 1998, p. 5).

O processo de formulação segue os passos a seguir: definição das variáveis de decisão, emprego destas variáveis para definir um grupo de equações de restrições onde uma solução viável do sistema corresponda a uma solução viável do problema formulado e, por último, utilização das variáveis para definir uma função objetivo, a qual irá medir o quão boa é cada solução viável pesquisada.

Neste sentido, Goldbarg e Luna (2000, p. 10) afirmam que as fórmulas ou equações do modelo não existem prontas e acabadas na natureza, elas têm que ser identificadas ou criadas. Contudo, para estes autores, o desenvolvimento de formulações se dá através de processos pouco rigorosos ou conhecidos, envolvendo, entre outros fatores: intuição, experiência, criatividade e poder de síntese.

Uma formulação para um problema de programação linear pode ser observada a seguir:

$$min\ c^t x$$

Sujeito a:

$$Ax \geq b$$

$$x \geq 0$$

Onde c e x são vetores pertencentes a R^n, $b \in R^m$, A é uma matriz $m\ x\ n$. A função $f(x) = c^t x$ é chamada de função objetivo.

Existem outras formulações gerais, tendo como o objetivo a maximização ao invés de minimização.

$$max\ c^t x$$

Sujeito a:

$$Ax \leq b$$

$$x \geq 0$$

Onde, igualmente ao anterior, c e x são vetores pertencentes a R^n, $b \in R^m$, A é uma matriz $m\ x\ n$. A função $f(x) = c^t x$ é chamada de função objetivo.

Para ilustrar a formulação, será utilizado o exemplo 5 a seguir: uma fábrica necessita definir os níveis de produção de trinta produtos que serão produzidos em determinado período de tempo. O objetivo primordial é maximizar a receita oriunda da venda destes produtos. O processo de fabricação incorre em algumas limitações, tais como: níveis mínimos de atendimento da demanda (estipulados via contratos com clientes), limitações relacionadas à capacidade de matérias primas, mão de obra disponível, além de outros fatores.

A formulação correspondente ao exemplo 5 assume a seguinte forma: o vetor x representa o vetor das variáveis de decisão. As variáveis de decisão podem ser definidas como sendo a quantidade de produzida de um produto p (variável "D_p"). Neste caso, existiriam as variáveis $D_1, D_2, \dots, D_{30}$. O vetor c contém o incremento unitário na função objetivo para cada variável de decisão contida no vetor x. No caso, seria a receita obtida com a comercialização de cada produto p (a constante que representa a receita unitária correspondente ao produto p poderia ser denominada por "R_P"). Dado que existem limitações de recursos (mão de obra, recursos físicos, atendimento mínimo, demanda máxima estimada), estes estariam representados no vetor b o vetor que contém quantidades máximas que devem ser observadas

para cada restrição (estas constantes poderiam ser denominadas por L_d, L_a, L_m, L_o e representariam as limitações de demanda, atendimento mínimo, matérias-primas e mão de obra. Por último, a matriz A contém as quantidades de recursos consumidos e outros limites por unidade de produção do produto p, sendo que as constantes poderiam ser denominadas por $C_{pd}, C_{pa}, C_{pm}, C_{po}$. Este é um problema bastante simples, com reduzido número de variáveis de decisão (igual ao número de produtos, 30).

A formulação de um problema precisa ser resolvida para que cumpra o objetivo de eleger a melhor solução dentre as soluções viáveis. Neste sentido, o algoritmo simplex foi desenvolvido para resolver formulações de programação linear. Um aprofundamento na teoria do método simplex fugiria aos objetivos desta dissertação. Serão expostos apenas os principais pontos de funcionamento do algoritmo. A relevância de expor estes pontos se evidenciará no momento que se estabelecer a diferença entre formulações boas e formulações ruins, no tópico de programação inteira.

Segundo Wolsey (1998, p. 12), um poliedro consiste num subconjunto de R^n descrito por um conjunto finito de restrições lineares:

$$P = \{x \in R^n \mid Ax \leq b\}$$

As Proposições 1 e 2 são fundamentais para entender o funcionamento do método simplex:

Proposição 1: O conjunto de todas as soluções viáveis de um problema de programação linear é um poliedro.

Proposição 2: Se um problema de programação linear tem solução ótima, então existe uma solução ótima que corresponde a um vértice do seu poliedro.

O método simplex parte de uma solução básica factível (uma solução que se encontra num dos vértices do poliedro) e examina se há uma solução melhor (que favoreça a função objetivo) utilizando a *condição de otimalidade*. Se a condição de otimalidade for verificada, a solução básica factível atual é a solução ótima. Caso contrário, o algoritmo pesquisa outro vértice do poliedro, até que a condição de otimalidade seja verificada.

A condição ou teste de otimalidade funciona a partir da análise dos vértices adjacentes ao vértice atual. Se uma solução básica factível não tiver nenhuma solução básica factível adjacente melhor (conforme medido pelo valor da função objetivo), então esta tem de ser a solução ótima do problema.

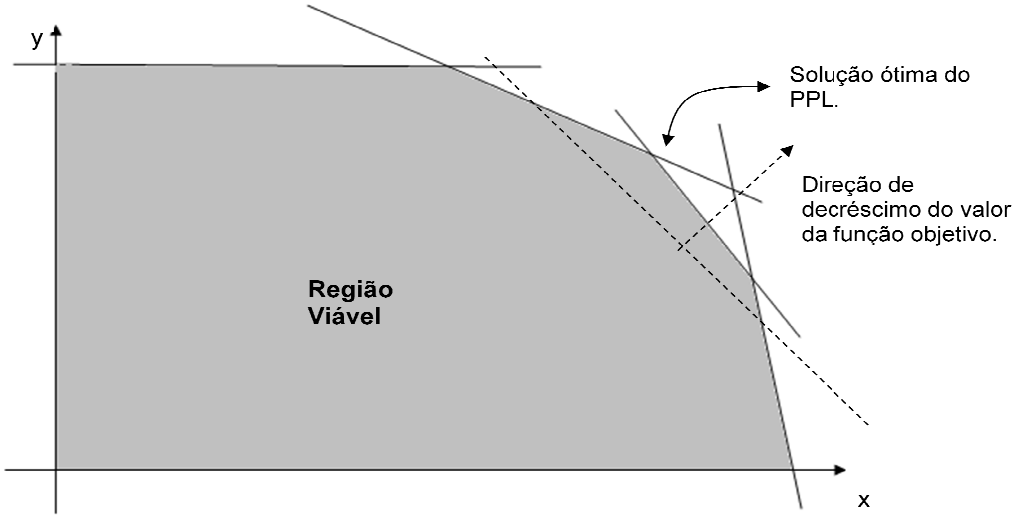

Figura 3 - Poliedro referente à formulação de programação linear

Na prática, o método simplex empregado em softwares é o método simplex revisado (que trabalha com a inversa da matriz básica). O método simplex original "não é o procedimento computacional mais eficiente, pois ele calcula e armazena muitos números que não são necessários na iteração atual e podem não ser mais relevantes nas iterações seguintes" (HILLIER; LIEBERMAN, 2006, p. 180). O método simplex revisado, por sua vez, processa e armazena somente as informações necessárias (coeficientes das variáveis não básicas, os coeficientes da variável básica que entra nas demais equações e os lados direitos das equações). Pode-se dizer que ele é uma versão aprimorada do procedimento original (HILLIER; LIEBERMAN, 2006, p. 180).

Entretanto, a essência do algoritmo permanece: a pesquisa dentre um número finito de pontos (vértices do poliedro) até encontrar a melhor solução (que minimize ou maximize) a função objetivo.

O algoritmo simplex quase sempre soluciona o problema em um número de iterações que é um pequeno múltiplo de dimensão do problema. Entretanto, é sabido que o método pode visitar cada vértice do poliedro. Conseqüentemente, o pior caso do método faz com que o algoritmo se torne de tempo exponencial (WRIGHT, 2004, p. 41).

Com o advento da teoria da complexidade computacional na segunda metade do século XX, buscou-se desenvolver um algoritmo de tempo polinomial que solucionasse problemas de programação linear. Em 1979, Leonid Khachian da União Soviética, baseando-se em trabalhos de outros eminentes matemáticos soviéticos (Shor, Yudin e Nemirovskii) publicou o primeiro algoritmo de tempo polinomial para problemas de programação linear.

Apesar de ser um algoritmo de tempo polinomial, sua implementação prática se mostrou bastante inferior ao método simplex.

Coube ao indiano Narendra Karmarkar em 1984, a criação de outro algoritmo de tempo polinomial para solução de problemas de programação linear que não tem as limitações computacionais do algoritmo de Khachian e que para problemas grandes e muito grandes (em especial com muitas restrições) chega a ser dezenas de vezes mais rápido que o algoritmo simplex (COLIN, 2007, p. 3).

Um trabalho interessante sobre o desenvolvimento histórico do método de pontos interiores foi elaborado por Wright (2004).

2.3 PROGRAMAÇÃO INTEIRA

"A característica que distingue a Programação Inteira da Programação Linear reside no fato de que algumas das variáveis pertencem a um conjunto discreto, tipicamente, um subconjunto dos números inteiros", Arenales et al. (2007, p. 163).

Abaixo segue a formulação de um problema de programação inteira:

$$min\ c^t x$$

Sujeito a:

$$Ax \geq b$$

$$x \geq 0 \quad .$$

Onde c é um vetor pertencente a R^n, x é um vetor pertencente a Z_+^n , $b \in R^m$, A é uma matriz $m\ X\ n$. A função $f(x) = c^t x$ é chamada de função objetivo.

Outra forma comum de representar o problema é:

$$min\{cx: Ax \geq b, x \geq 0\}$$

Onde A é uma matriz $m\ X\ n$, c é um vetor de dimensão n, x é um vetor coluna de variáveis com dimensão n e pertencente a Z_+^n. Quando adicionado nas restrições a obrigatoriedade de que algumas das variáveis de decisão assumam apenas valores inteiros, tem-se um problema de Programação Inteira Mista (abreviado por PIM). Caso seja obrigatório que todas as variáveis de decisão assumam somente valores inteiros, tem-se um problema de

Programação Inteira (PI) e, por último, caso as variáveis possam assumir apenas valores iguais a 0 e 1, tem-se um problema de Programação Inteira Binária (PIB).

Na prática, muitas vezes utiliza-se somente o termo Problema de Programação Inteira, para descrever qualquer um dos três casos acima.

O fato que diferencia a programação inteira da programação linear pode parecer um elemento simplificador porém, na maioria dos casos, “a solução de problemas de programação inteira é extremamente difícil do ponto de vista computacional” (COLIN, 2007, p. 173).

A dificuldade em resolver problemas de programação inteira pode ser analisada também de forma intuitiva. Na programação linear a Proposição 2 afirma que: se um problema de programação linear tem solução ótima, então existe um vértice ótimo no poliedro. Grosseiramente, isto significa que existe um “local” onde procurar a solução ótima (mesmo que o número de vértices assuma dimensão extremamente grande em instâncias com elevado número de restrições e variáveis). A questão é que existem algoritmos de tempo polinomial (pontos interiores) e exponencial (simplex, no seu pior caso) que possibilitam a resolução do problema dentro de razoável tempo computacional. Em contrapartida, a avaliação das soluções que se encontram nos vértices do poliedro não pode ser utilizada para resolver problemas de Programação Inteira, dado que, as variáveis de decisão, que são as coordenadas para o vértice, podem não ser inteiras. Ou seja, o vértice do poliedro pode não representar uma solução do problema, pois viola a restrição de integralidade.

Uma opção seria limitar o espaço de soluções de tal maneira que os vértices correspondessem a pontos inteiros e, assim, resolver o problema utilizando o algoritmo simplex ou de pontos interiores. Entretanto, descrever todas as equações necessárias de um poliedro que contém apenas coordenadas inteiras nos vértices é algo extremamente difícil para muitos problemas na prática. Por outro lado, quanto mais próximo é o espaço delimitado pelas equações dos pontos inteiros subjacentes, mais forte é a formulação. Isto será definido com mais rigor um pouco mais a frente.

Uma segunda opção seria resolver o problema “relaxando” a restrição de integralidade (o problema originado é conhecido por relaxação linear do problema de programação inteira), resolver o problema originado e arredondar o valor encontrado para as variáveis para o número inteiro mais próximo. Infelizmente, este procedimento não é eficaz, uma vez que a solução encontrada pode estar muito afastada da solução ótima. A Figura 4 ilustra a situação:

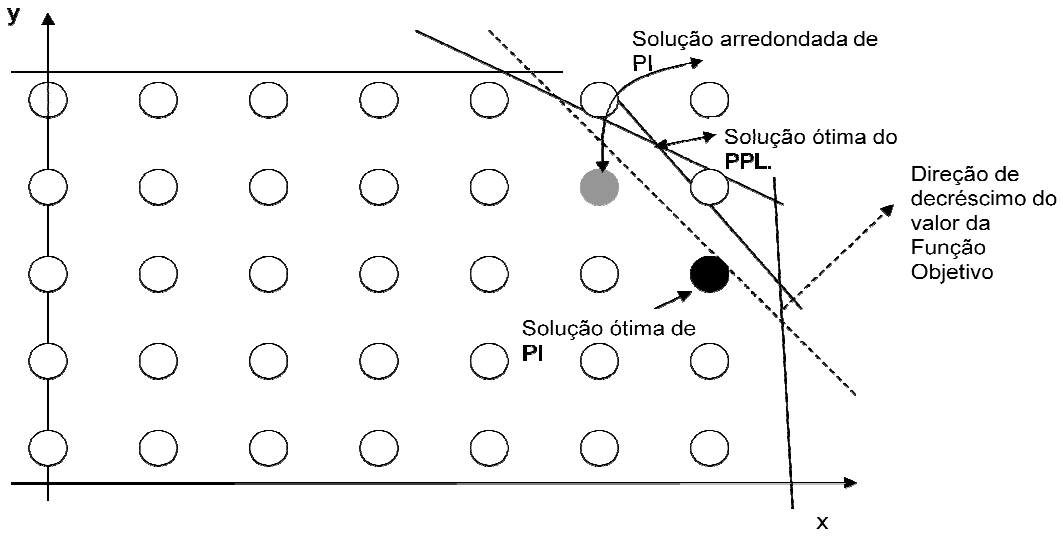

Figura 4 - Tentativa de arredondamento para o problema de PI

Dessa forma, se não é possível ter uma região de "busca", qual o número de soluções possíveis a serem pesquisadas? Todas as soluções viáveis encontram-se no interior do poliedro correspondente a relaxação linear do problema de Programação Inteira. O número de soluções possíveis tende a ser extremamente grande, conforme demonstrado na Tabela 1, no primeiro capítulo deste trabalho.

No tocante a resolução dos problemas de programação inteira, Arenales et al. (2007, p. 163) destacam que:

> Os problemas de programação inteira e combinatória são resolvidos por métodos ótimos (exatos), que fornecem uma solução ótima; por algoritmos aproximados, que garantem a distância máxima entre o valor da solução subótima encontrada e o valor ótimo; ou por métodos heurísticos, que, em geral, fornecem uma solução subótima, sem conhecimento de qualidade desta em relação a uma solução ótima.

Os métodos heurísticos também são utilizados como apoio aos métodos ótimos (exatos) no sentido de fornecer uma solução viável inicial. Neste sentido, alguns algoritmos heurísticos serão empregados com o intuito de apoiar a resolução do problema .

2.3.1 Métodos Exatos de Resolução

Kallrath e Wilson (1997, p. 90) frisam que, dentre os melhores algoritmos exatos que surgiram nas últimas décadas para resolução de problemas de programação inteira, destacam-se: algoritmos de enumeração implícita, algoritmos de planos de cortes, algoritmos de *branch and cut* e a Programação Dinâmica.

Posto que não existe um local específico, representado pelos vértices da região delimitada pelas restrições do problema (poliedro) para busca de uma solução ótima, o que pode ser feito para resolver o problema de programação inteira? Como um algoritmo poderia ser desenvolvido? Colocado de outra maneira, para um problema de programação inteira com valor da função objetivo dado por *z*, como é possível afirmar que uma das soluções factíveis é uma solução ótima?

Wolsey (1998, p. 23) destaca que é necessário encontrar um limite inferior $\underline{z} \leq$ z (*lower bound*) e um limite superior $\overline{z} \geq$ z (*upper bound*), para o valor z tal que $\underline{z} = \overline{z} = z$. O autor assevera que, praticamente, qualquer algoritmo irá buscar uma seqüência decrescente de limites superiores tal que: $\overline{z_1} > \overline{z_2} > \cdots > \overline{z_s} \geq z$ e uma seqüência crescente de limites inferiores $\underline{z_1} < \underline{z_2} < \cdots < \underline{z_t} \leq z$. A condição de parada do algoritmo se dá quando: $\overline{z_s} - \underline{z_t} \leq \varepsilon$, onde ε é um valor pequeno e não negativo.

Uma questão fundamental diz respeito a como encontrar os limites $\overline{z}$ e $\underline{z}$. Para um problema de minimização, qualquer solução viável fornece um limite superior (sendo que, para um problema de maximização, qualquer solução viável informa um limite inferior). Os limites superiores num problema de minimização e inferiores num problema de maximização são denominados limites primais (*primal bounds*). Neste sentido, Wolsey (1998, p. 24) transmite que para muitos problemas de programação inteira, encontrar soluções viáveis é uma questão simples sendo que, o ponto principal é encontrar boas soluções viáveis (o que se traduz em valores mais próximos ao valor ótimo). Ainda segundo o autor, para outros problemas, encontrar uma solução viável é tão difícil quanto encontrar uma solução ótima para o PI.

Em contrapartida aos limites primais, apresenta-se como desafio maior encontrar limites inferiores nos problemas de minimização ou limites superiores em problemas de maximização. Estes limites são conhecidos por limites duais (*dual bounds*). A abordagem mais importante é dada pela relaxação, a idéia de substituir um problema de programação inteira por um simples problema de otimização onde o valor ótimo dado pela função objetivo

é, no mínimo, menor (problema de minimização) ou maior (problema de maximização) do que o valor do problema original de PI (WOLSEY, 1998, p. 24).

Para que o problema de otimização associado apresente esta propriedade, é necessário que, ou o conjunto das soluções viáveis do problema associado seja maior do que o conjunto do problema original, ou a função objetivo é modificada por uma função que apresente valor superior (problema de maximização) ou valor inferior (problema de minimização), ou o mesmo valor (problemas de maximização e minimização).

Exemplos de relaxações são: a relaxação linear, a relaxação combinatória e a relaxação lagrangeana. Para esta dissertação, por motivos que se mostrarão óbvios mais a frente, será exposta com mais detalhe apenas a relaxação linear.

Seja o problema: $min\{cx : x \in P \cap Z^n\}$ e a formulação $P = \{x \in R_+^n : Ax \geq b\}$. A relaxação linear é dada pelo problema: $min\{cx : x \in P\}$. Uma vez que $P \cap Z^n \subseteq P$ e a função objetivo não foi alterada, o problema $min\{cx : x \in P\}$ é uma relaxação linear do problema original: $min\{cx : x \in P \cap Z^n\}$.

A proposição a seguir será bastante utilizada no momento da análise e comparação das formulações para o problema $F_2||\sum w_j T_j$.

Proposição 3: Suponha que existem duas formulações P_1 e P_2, para o problema $min\{cx : x \in X \subseteq Z^n\}$, sendo P_1 uma formulação melhor do que P_2 ($P_1 \subseteq P_2$). Seja z_1^{LP} e z_2^{LP} os valores das relaxações lineares associadas às formulações P_1 e P_2. Assim, $z_1^{LP} \geq z_2^{LP}$, para todo c.

Ou seja, quanto mais forte é a formulação, mais próximo do valor ótimo da função objetivo será o valor da relaxação linear.

A proposição 3, tomada de forma intuitiva, origina a idéia de que, para um mesmo problema, existem formulações “mais apertadas” do que outras (graficamente, suponha um problema D e duas formulações S_1 e S_2 expostas na Figura 5 a seguir).

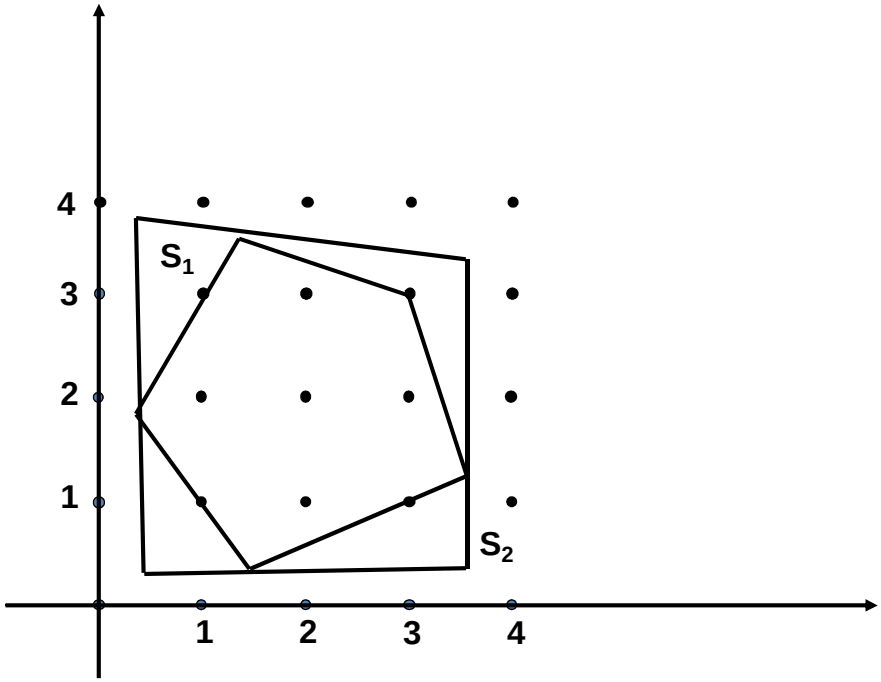

Figura 5 - Duas formulações para um mesmo problema

Nota-se que a formulação S_1 é mais apertada, no sentido que envolve os pontos inteiros de forma mais próxima do que a formulação S_2.

De fato, uma formulação ideal seria a formulação S_3, ilustrada na Figura 6.

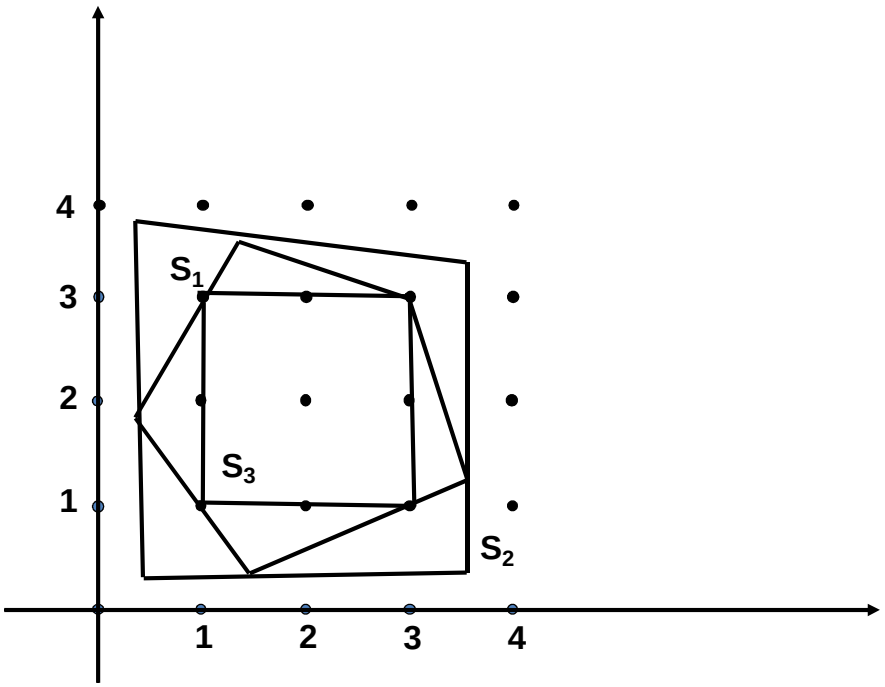

Figura 6 - Uma formulação S_3 “ideal”

Porque a formulação S_3 é ideal? Por que ao se resolver a relaxação linear, o ponto extremo corresponde a uma solução que contém somente variáveis inteiras. Assim, ao resolver a relaxação linear do problema original, o problema estará solucionado.

Na prática, descrever completamente todas as restrições de uma formulação ideal incorre num elevado número de restrições, e este número tende a ser extremante grande na maioria dos problemas (WOLSEY, 1998, p. 15). Uma formulação ideal equivale a envoltória convexa do conjunto de pontos que são solução inteira, é representado por $conv(X)$, onde X é o conjunto de soluções, $X \subseteq R^n$.

Apesar do fato de que o conjunto das soluções viáveis de um problema de programação inteira geralmente compreender um conjunto finito (fato este distinto na PL), a completa enumeração das soluções é inviável, mesmo para instâncias pequenas (um exemplo deste fato foi exposto na Tabela 1). Assim, enumerar todas as soluções não é uma opção inteligente.

Uma segunda opção é utilizar o método de enumeração implícita. O método parte da premissa de que é possível dividir o problema inicial numa série de problemas menores que são mais fáceis de resolver e utilizar as informações destes subproblemas para resolver o problema inicial. Em particular, as informações fundamentais dos subproblemas são os limites (*bounds*) fornecidos através das resoluções (conforme exposto anteriormente, praticamente todos os algoritmos para resolução de problemas de programação inteira baseiam-se em estabelecer uma seqüência crescente de limites inferiores e uma seqüência decrescente de limites superiores). A proposição a seguir, extraída de Wolsey (1998, p. 93) é essencial para compreender o método de enumeração implícita, ou "*branch and bound*", proposto por Land e Doing (1960).

Proposição 4: Seja $S = S_1 \cup S_2 \ldots\ldots \cup S_k$ a decomposição de S numa série de conjuntos menores e seja $z^k = max\{cx: x \in S_k\}$ para $k = 1, \ldots, K$, $\overline{z}^k$ é um limite superior de z^k e $\underline{z}^k$ é o limite inferior de z^k. Então, $\overline{z} = max_k \overline{z}^k$ é um limite superior do problema inicial z e $\underline{z} = max_k \underline{z}^k$ é um limite inferior do problema inicial z.

O método de *branch and bound* é comumente representado a partir de uma árvore de enumeração. Uma ilustração do funcionamento do método será feita a seguir. O exemplo foi extraído de Arenales et al. (2007, p. 239).

Seja o problema P:

$$z = max\ 5x_1 - x_2$$
$$7x_1 - 5x_2 \leq 13$$
$$3x_1 + 2x_2 \leq 17$$
$$x \in Z_+^2$$

A relaxação linear do problema P obteve valor z = 16,38 para a função objetivo, além dos valores para x_1 = 3,83 e x_2 = 2,76. Denomina-se o problema inicial como "nó raiz":

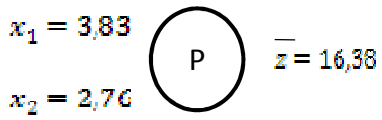

Figura 7 – Relaxação linear do problema *P*.

A partir daí, seleciona-se uma variável e efetua-se sua ramificação. Para o exemplo, optou-se por ramificar a variável x_1 = 3,83, gerando dois subproblemas P_1 e P_2 a partir da inclusão da restrição $x_1 \leq 3$ no subproblema P_1 e $x_1 \geq 4$ no subproblema P_2.

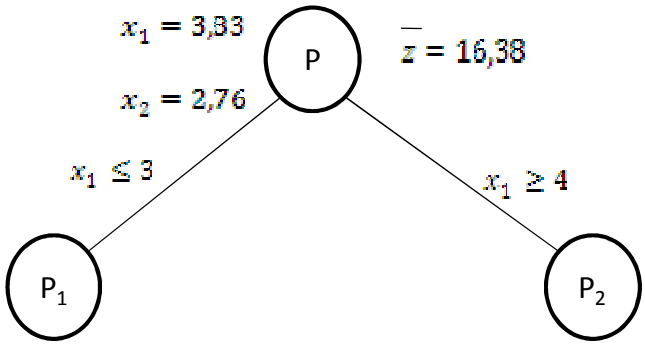

Figura 8 - Ramificação da variável x_1.

O problema P_2 é inviável, pois a restrição adicionada, "$x_1 \geq 4$", faz com que o conjunto das soluções factíveis de P_2 seja vazio. Já o problema P_1, ao ser resolvido, obtém-se

$\overline{z_1}$ = 13,40 (limite dual). Notar que este limite é mais "apertado" que o valor de $\overline{z}$ = 16,38. Para o problema P_1, tem-se x_1 = 3,00 e x_2 = 1,60:

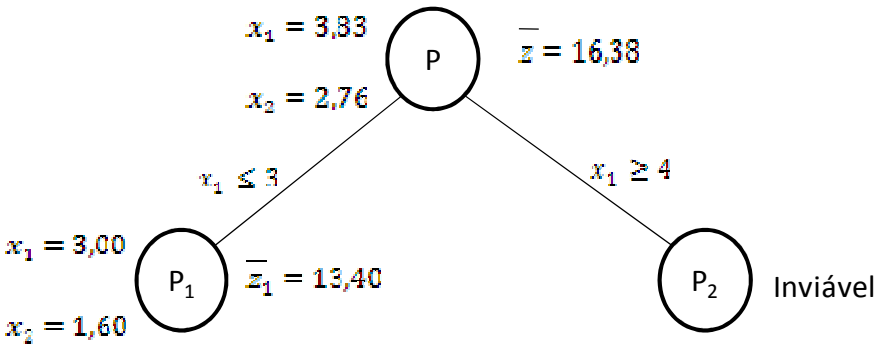

Figura 9 - Resultados dos subproblemas P_1 e P_2.

Novamente uma variável necessita ser ramificada. Seleciona-se a variável $x_2 = 1{,}60$, fazendo $x_2 \geq 2$ e $x_2 \leq 1$. Tem-se agora a seguinte árvore:

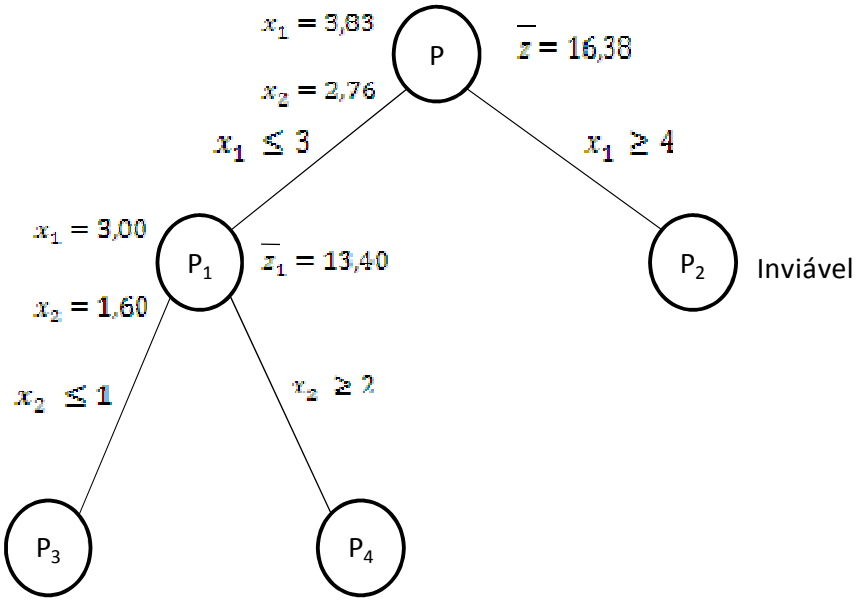

Figura 10 - Ramificação da variável x_2.

Resolvendo o problema P_3 encontra-se como valor da relaxação linear o valor de $\overline{z_3}$ = 11,86 (limite superior). Os valores obtidos pelas variáveis foram: x_1 = 2,57 e x_2 = 1,00.

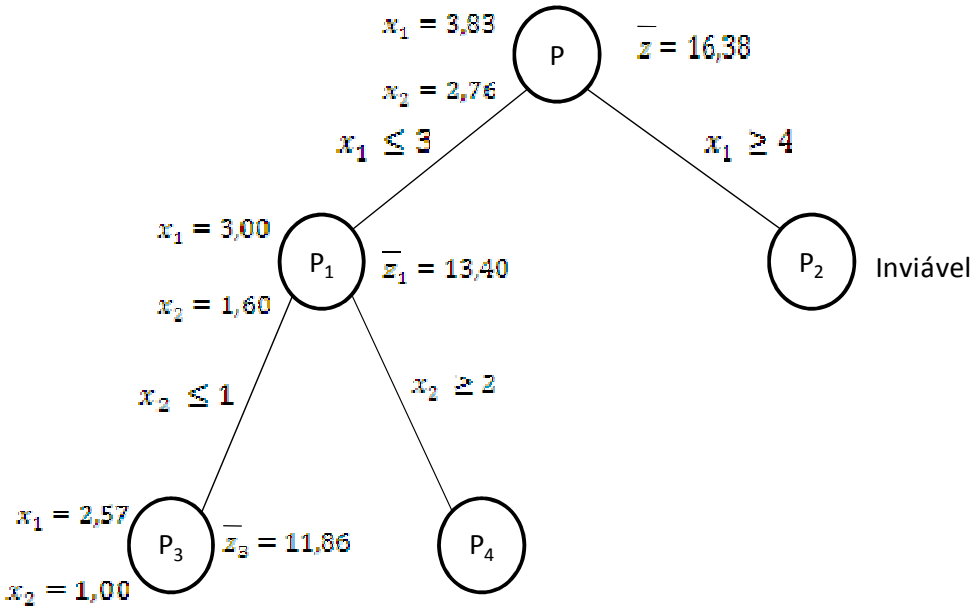

Figura 11 - Resolução do subproblema P_3.

Resolve-se também o problema P_4. O valor obtido pela relaxação linear foi $\overline{z_4}$ = 13,00 (limite superior). Já as variáveis assumem os valores inteiros x_1 = 3,00 e x_2 = 2,00.

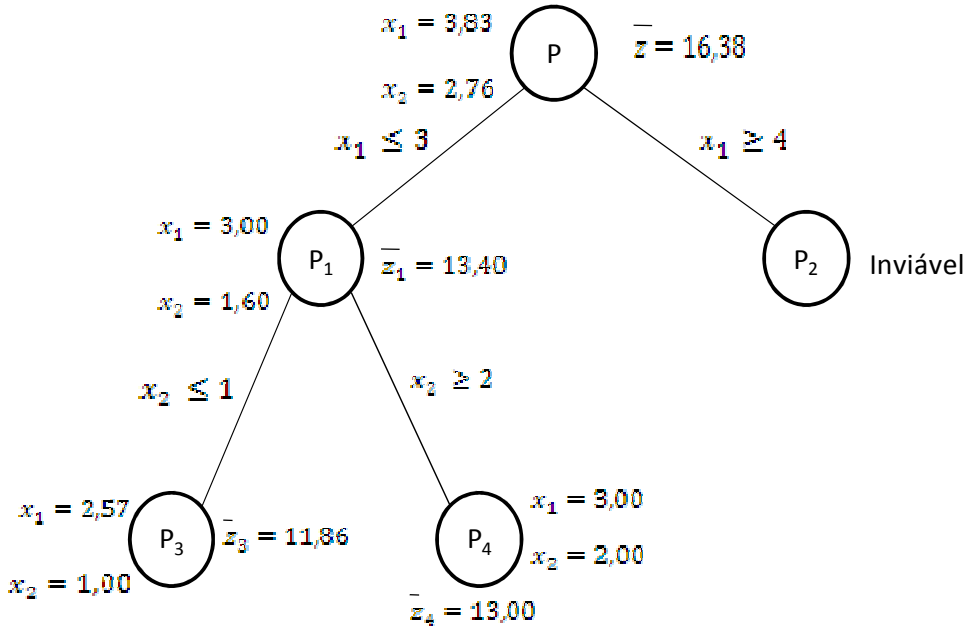

Figura 12 - Resolução do subproblema P_4.

A solução ótima é fornecida pelo subproblema P_4. Notar que, dentre os subproblemas P_3 e P_4 este último apresenta valor maior da relaxação linear.

Como o algoritmo atingiu a condição de parada? Alguns pontos se destacam acima: o primeiro deles é o fato de que, caso alguma variável do nó representado pelo subproblema P_3 fosse ramificada, o valor da relaxação linear não ultrapassaria o valor de $\overline{z_3}$ = 11,86, que foi o valor obtido pela relaxação linear do nó pai (subproblema P_3). Se já se possui uma solução inteira de valor z = 13,00, não faz sentido investigar um nó que não originará valores superiores.

Um segundo ponto é: como os coeficientes das variáveis na função objetivo são todos números inteiros, a solução ótima z é, necessariamente, um número inteiro. Como o subproblema P_1 apresenta valor da relaxação linear igual a 13,40, o valor máximo que a função objetivo poderia obter é 13,00. Justamente é o valor obtido pelo subproblema P_4. Caso o subproblema P_4 tivesse sido resolvido antes do subproblema P_3, esta questão já seria uma condição de parada suficiente.

Outro ponto importante é: ao resolver o subproblema P_4, obtiveram-se valores inteiros para as variáveis x_1 e x_2. Ou seja, mesmo com a restrição de integralidade relaxada, já foram obtidos valores inteiros para as variáveis neste nó. Assim, não faz nenhum sentido ramificar qualquer variável no subproblema P_4.

Hillier e Lieberman (2006, p. 491) asseveram que existem três condições que descartam um nó de considerações posteriores: se a relaxação não contiver soluções viáveis, se a solução ótima da relaxação for uma solução que apresente todas as variáveis inteiras e, por último, se a solução da relaxação do subproblema em julgamento for inferior a melhor solução inteira conhecida (problema de maximização) ou superior a melhor solução inteira já obtida (problema de minimização).

É notório que o método de *branch and bound* pode acabar por ter de resolver um grande número de subproblemas. No simples problema P acima, com duas variáveis de decisão, foi necessário resolver 4 subproblemas, além do nó raiz (ao todo, 5 problemas foram resolvidos). Este número tende a ser extremamente grande para problemas de decisão envolvendo elevado número de variáveis de decisão.

Para problemas de dimensões elevadas, uma questão importante é como escolher o próximo nó a ser resolvido. Arenales et al. (2007, p. 248) destacam que existem duas alternativas básicas: regras a priori, que determinam de antemão a ordem de seleção dos nós, e regras adaptativas, que seleciona o nó a partir de informação dos nós ativos.

A busca em profundidade é uma das regras a priori. Na busca em profundidade, caso o nó corrente não seja eliminado, então o próximo nó a ser examinado é um de seus filhos, e assim sucessivamente (ARENALES et al., 2007, p. 248). Uma regra pode também empregar a estratégia de *backtracking* o que significa que, quando um nó é eliminado, retorna-se ao longo do caminho em direção ao nó raiz até encontrar o primeiro nó que tem um filho ainda não examinado.

Pode-se determinar também que o próximo nó a ser aberto esteja à esquerda ou a direita do nó pai. Dessa forma, especifica-se exatamente o desenvolvimento da árvore. Um fator importante é a seleção da primeira variável que será ramificada no nó raiz (problema original com a restrição de integralidade relaxada).

Outros elementos podem auxiliar no encontro a solução ótima em menores tempos. O primeiro deles é a adição de planos de cortes.

Os algoritmos de planos de cortes buscam obter uma aproximação da envoltória convexa da região factível de um problema de programação inteira que contenha um ponto extremo correspondente a uma solução ótima. Essa aproximação é obtida por meio de cortes ou desigualdades válidas (ARENALES et al., 2007, p. 250).

Kallrath e Wilson (1997, p. 93) lembram que os algoritmos de planos de corte puros tendem as ser lentos no encontro da solução ótima e, por este fato, não é comum a sua utilização pelos pacotes de softwares comerciais.

A utilização dos planos de cortes juntamente ao *branch and bound* deu origem ao método, (ou algoritmo) de *branch and cut*. Wolsey (1998, p. 157) afirma que o *branch and cut* é o algoritmo de *branch and bound* em que alguns planos de cortes são gerados ao longo da árvore de *branch and bound*.

Arenales et al. (2007, p. 256) lembra que o método tem por objetivo reduzir o número de nós na árvore de *branch and bound*. Em cada nó da árvore de *branch and bound*, adicionam-se desigualdades válidas de modo a obter um limitante mais apertado no nó.

O método de *branch and cut* é largamente utilizado nos pacotes comerciais. Usualmente, os pacotes empregam uma etapa de pré-processamento, onde são identificadas redundâncias nas variáveis e restrições, além de apertar os limitantes de variáveis. Se o problema resultante de programação inteira é menor e ou mais apertado, então, provavelmente será resolvido com mais rapidez (ARENALES et al., 2007, p. 260).

2.3.2 Heurísticas

Devido ao fato de que, para muitos problemas $\mathcal{NP}$-difíceis, encontrar uma solução ótima incorre em elevado tempo computacional (sendo que, para outros problemas desta classe, a descoberta de uma solução “apenas” viável já demanda considerável tempo), o desenvolvimento de regras para a descoberta de soluções não ótimas de maneira rápida representa um importante fator, sendo as heurísticas os atores principais neste campo. Exemplos das situações onde uma heurística pode auxiliar seguem abaixo.

No caso de um tomador de decisão aceitar incorrer num determinado “prejuízo” em relação à solução ótima, caso a resposta fornecida por uma heurística transmita uma solução viável de forma rápida. O tomador de decisão pode até mesmo não dispor de tempo necessário para que a solução ótima seja conhecida. Neste caso, uma heurística cumpre papel fundamental.

Uma segunda função que uma heurística pode cumprir é a sua utilização juntamente ao método de *branch and bound* ou *branch and cut*, Uma heurística pode ser utilizada para fornecer um limite primal inicial. Quanto mais “apertado” for este limite, mais a heurística terá auxiliado o algoritmo na busca pela solução ótima.

Mas no que consiste uma heurística? Nicholson (1971, apud ARENALES et al., 2007, p. 269) define uma heurística como sendo um procedimento para resolver problemas por meio de um enfoque “intuitivo”, em geral racional, no qual a estrutura do problema possa ser interpretada e explorada inteligentemente para se obter uma solução razoável.

Como finalização do capítulo, uma reflexão se coloca: são vários os fatores que influenciam na viabilidade ou velocidade em encontrar uma solução para o problema $F_2||\sum w_j T_j$. Como observado neste capítulo, cumprem papel fundamental: a qualidade da formulação desenvolvida, uma vez que quanto mais “apertada” for a formulação, mais rápido os métodos disponíveis convergem para uma solução ótima, o tipo de regra de seleção dos nós nas árvores do algoritmo de *branch and cut*, os planos de cortes adicionados, a passagem e a qualidade de uma solução viável inicial, fornecido por uma heurística. Sem dúvida, todos estes elementos podem ser objetos de estudos aprofundados.

3 FORMULAÇÕES PARA O PROBLEMA DO *FLOW SHOP* COM DUAS MÁQUINAS E PENALIDADES POR ATRASO NAS TAREFAS

O conceito de formulação foi abordado anteriormente, juntamente aos conceitos que possibilitam comparar formulações diferentes para um mesmo problema.

Uma questão que pode ser levantada em relação ao problema $F_2||\sum w_j T_j$ é a seguinte: há algum impacto negativo no valor da função objetivo de um problema $F_2||\sum w_j T_j$ se for adicionada a restrição de processamento $prmu$ (permutação), segundo a qual a ordem de execução das tarefas não pode ser alterada ao longo das máquinas? A importância em investigar esta questão se justificará mais a frente, ao serem apresentadas as formulações desenvolvidas neste trabalho.

A questão será ilustrada utilizando a instância 1, exposta na Tabela 2 a seguir:

j	p_{1j}	p_{2j}	w_j	d_j
1	2	4	4	5
2	2	2	3	8
3	2	1	5	9
4	3	2	1	11

Tabela 2 - Dados da instância 1

Um escalonamento ótimo pode ser observado na Figura 13. Notar que, para cada máquina, há uma ordem diferente de execução das tarefas.

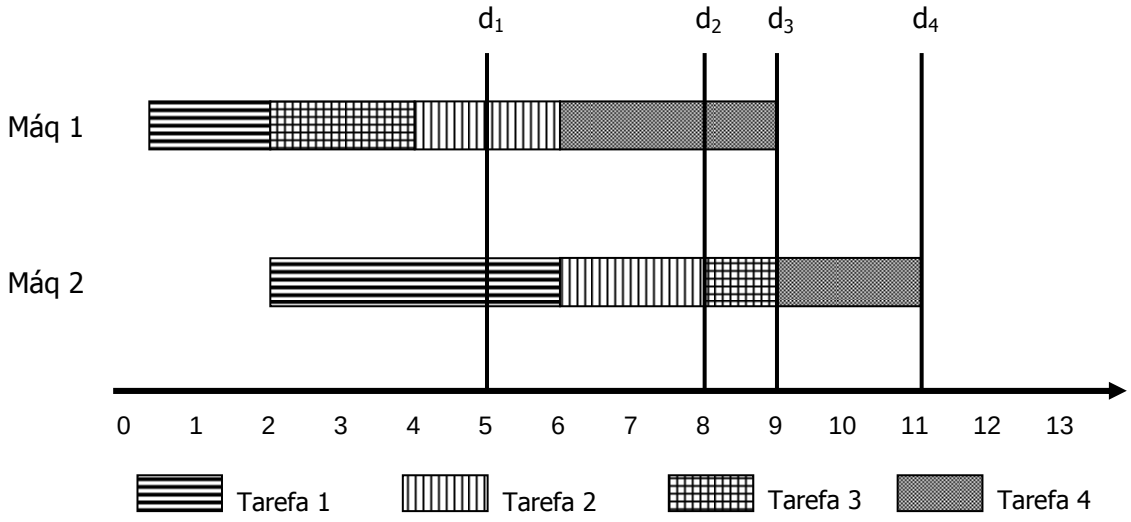

Figura 13 - Escalonamento *Flow shop* não permutacional

A tarefa 3 foi a segunda a ser processada na máquina 1. Contudo, há uma troca de posição com a tarefa 2 no momento do processamento na última máquina.

A alteração da seqüência de processamento na máquina 1 é ilustrada na Figura 14. Nota-se que não ocorre prejuízo à solução ótima, uma vez que as tarefas continuam sendo finalizadas nos mesmos horários.

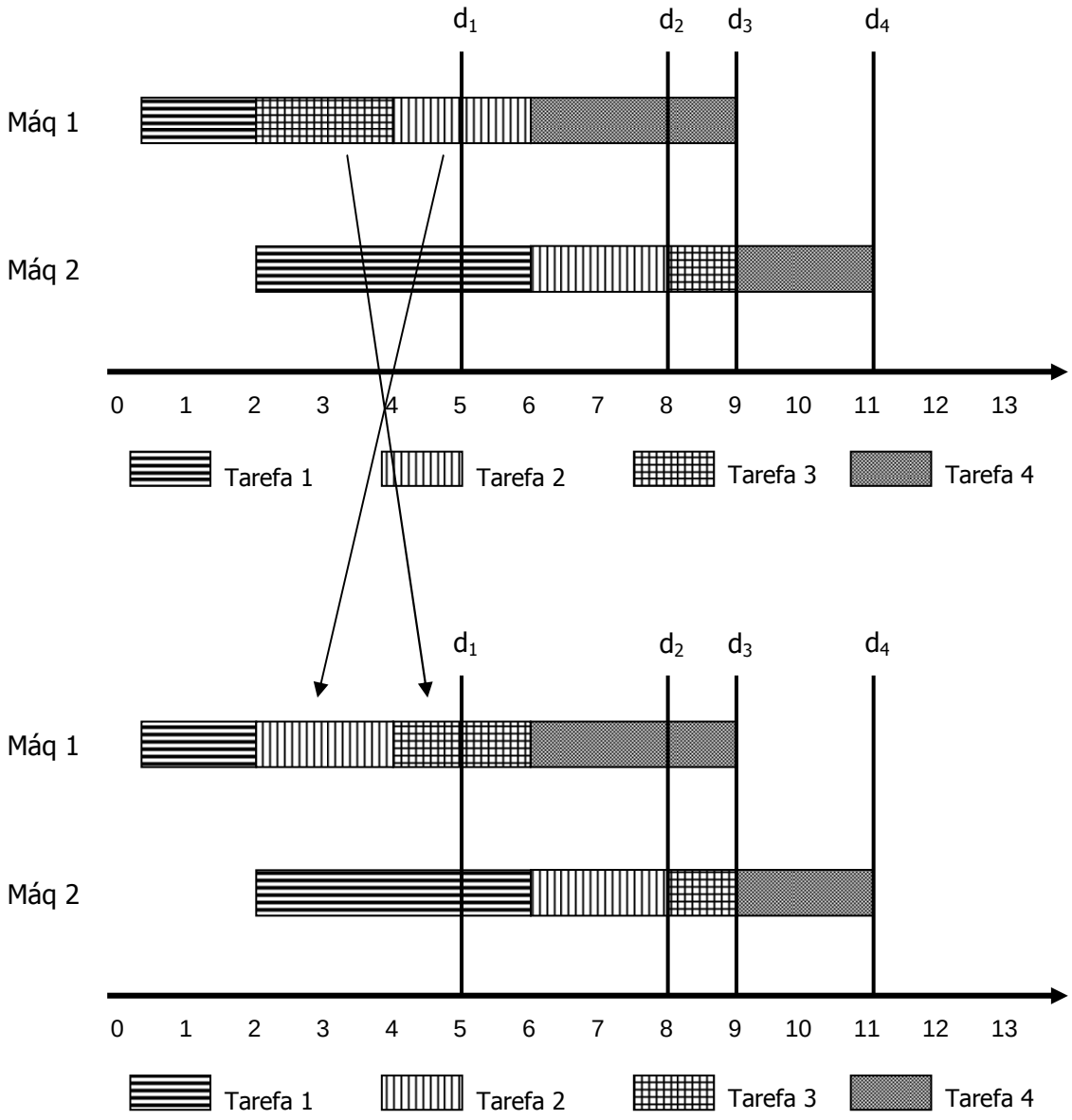

Figura 14 - Alteração da seqüência de execução das tarefas na primeira máquina

Teorema 3: Para toda solução ótima, de uma entrada do problema $F_2||\sum w_j T_j$ onde o escalonamento é não permutacional, é possível encontrar um seqüenciamento permutacional, a partir da mudança da ordem de execução das tarefas na primeira máquina, onde não há piora do valor da função objetivo.

Prova:

Se existe uma troca de ordem de execução entre duas tarefas (na passagem da primeira para a segunda máquina), tanto a tarefa que passou a frente na segunda máquina, quanto a tarefa que foi processada primeiro na máquina 1, já tiveram seu processamento finalizado na

1º máquina, até o instante que a segunda tarefa processada na máquina 2 tenha sido finalizada. Sendo assim, não há problemas em alterar a seqüência de execução da máquina 1, de forma que estas duas tarefas sigam a mesma seqüência na máquina 2. Repetindo este procedimento, um escalonamento permutacional pode ser obtido. □

O Teorema 3 implica no fato de que os problemas $F_2||\sum w_j T_j$ e $F_2|prmu|\sum w_j T_j$ são, em suma, equivalentes. Uma formulação para o problema $F_2||\sum w_j T_j$ pode ser aplicada a qualquer instância do problema $F_2|prmu|\sum w_j T_j$ que produzirá solução ótima com igual custo ao de uma formulação desenvolvida para $F_2|prmu|\sum w_j T_j$ e vice versa.

A seguir serão expostas as formulações de programação inteira, desenvolvidas para resolver o problema $F_2||\sum w_j T_j$. As formulações foram divididas em dois grupos, segundo a utilização de certos índices nas variáveis de decisões.

O primeiro grupo de formulações propostas possui três formulações, que serão intituladas por Formulações “1”, “2” e “3”. A característica comum destas formulações é a utilização de um índice de tempo *t* nas variáveis. Na Formulação 1 o índice *t* representa o tempo de conclusão de uma determinada tarefa *j* numa máquina *i*. Na Formulação 2, o índice *t* representa o tempo de conclusão da tarefa *j* na última máquina. A Formulação 3 parte da Formulação 2 e adiciona variáveis auxiliares.

O segundo grupo de formulações propostas não utiliza índices de tempo *t* nas variáveis. Por sua vez, utiliza variáveis com índices de posição da tarefa em relação às demais. Neste grupo, serão propostas duas formulações, uma para o caso geral do problema $F_2||\sum w_j T_j$ (que será denominada por Formulação 4) e outra para um caso especial do problema $F_2||\sum w_j T_j$, onde todos os pesos associados às tarefas são unitários. Esta última formulação será denominada por Formulação 5.

Para avaliar o desempenho das formulações propostas neste trabalho, foram feitos testes com instâncias que contemplam de 4 a 50 tarefas. Para cada instância com o mesmo número de tarefas, existe uma letra (A ou B) que, ao lado do número de tarefas, identifica totalmente qual instância está sendo utilizada.

As instâncias que contêm a letra A apresentam os tempos de processamento nas máquinas 1 e 2 (p_{1j} e p_{2j}) entre os valores 1 e 5. Já as instâncias com letra B, os valores destes dois parâmetros situam-se entre 1 e 10. Em relação aos pesos associados às tarefas (w_j), em todas as instâncias os valores mínimos e máximos estão entre 1 e 5.

A data de término prometida (d_j) apresenta como valor mínimo a soma dos tempos de processamento da tarefa *j* nas máquinas 1 e 2 ($p_{1j} + p_{2j}$) e como máximo o maior valor entre a soma dos tempos de processamento nas máquinas 1 e 2 de todas as tarefas.

Todos esses números foram gerados a partir de uma distribuição aleatória uniforme utilizando a fórmula "ALEATÓRIOENTRE" do *MS-Excel 2007*.

As Tabelas 16 e 17, disponíveis no Apêndice, informam os valores de p_{1j}, p_{2j}, w_j, d_j de cada tarefa *j*, para as instâncias da Série A e da Série B.

De forma a ilustrar as formulações, será utilizada uma instância com reduzido número de tarefas (três), descrita a seguir. Esta instância será denominada genericamente por "instância de teste".

J	p_{1j}	p_{2j}	w_j	d_j
1	5	4	4	9
2	5	1	3	10
3	2	3	5	5

Tabela 3 - Instância de teste para ilustração das formulações.

Para todas as formulações expostas, admite-se que não há benefício algum em manter a máquina 1 ociosa (até que todas as tarefas tenham sido processadas nesta máquina). O somatório dos tempos de processamento de todas as tarefas na máquina 1 será representado pela constante S_1. Assim:

$$S_1 = \sum_{j=1}^{n} p_{1j} \qquad (1)$$

Para a instância de teste, tem-se S_1 = 12.

Além disso, é fornecido um tempo máximo de execução das tarefas "t_{max}", calculado como sendo o maior valor entre o somatório dos tempos de processamento da máquina 1 e da máquina 2, multiplicado por 1,3 e arredondado para cima quando necessário. Algebricamente:

$$t_{max} = Max\left(\sum_{j=1}^{n} p_{1j}\,, \sum_{j=1}^{n} p_{2j}\right).1{,}3 \qquad (2)$$

O acréscimo de 30% ao valor máximo foi derivado a partir de experimentos. Notou-se que, com este acréscimo, há tempo suficiente para conclusão de todas as tarefas. A necessidade de empregar o "t_{max}" será vista a seguir.

O valor do t_{max} para a instância de teste é igual a 16.

3.1 FORMULAÇÕES COM VARIÁVEIS CONTENDO ÍNDICE DE TEMPO

Para o desenvolvimento deste grupo de formulações, utilizou-se como ponto de partida o trabalho de Dyer e Wolsey (1990) o qual introduziu na literatura uma formulação para o problema de escalonamento com uma máquina, empregando variáveis com índices *t* de tempo. A formulação pode ser vista abaixo:

Função Objetivo:

$$Min\ Z = \sum_{j=1}^{n} \sum_{t=1}^{t_{max}-p_j+1} c_{jt} x_{jt} \qquad (3)$$

Define-se a variável x_{jt}, que assume valor igual a 1 se a tarefa *j* é finalizada no tempo *t*, recebendo valo nulo se a tarefa *j* é finalizada em qualquer instante de tempo diferente de *t*.

O coeficiente c_{jt} pode representar qualquer coeficiente de custo que seja associado a um tempo de finalização *t*. Neste sentido, pode-se fazer $c_{jt} = w_j.max\,(0\,, t - d_j)$, o que é equivalente a $w_j.T_j$.

Sujeito a:

$$\sum_{t=1}^{t_{max}-p_j+1} x_{jt} = 1 \quad \forall \quad j = 1, \dots, n \qquad (4)$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \quad \forall \quad t = 1, \dots, t_{max} \qquad (5)$$

$$x_{jt} \in \{0{,}1\} \quad \forall \quad j = 1, \dots, n; \quad \forall \quad t = 1, \dots, t_{max} - p_j + 1 \qquad (6)$$

As restrições (4) encarregam-se de que uma dada tarefa *j* seja finalizada uma e somente uma única vez. As restrições (5) impedem que para um dado tempo *t*, a máquina processe simultaneamente mais de uma tarefa.

Neste sentido, tomando por base a formulação de Dyer e Wolsey (1990), serão apresentadas a seguir as Formulações 1, 2 e 3, para o problema $F_2||\sum w_j T_j$, que empregam índices de tempo nas variáveis de decisão.

3.1.1 Formulação 1

A Formulação 1 emprega a variável de decisão binária x_{jit}, onde os índices *j*, *i* e *t* representam uma tarefa *j*, uma máquina *i* e tempo *t* de finalização da tarefa *j* na máquina *i*. A variável de decisão assume valor 1 se a tarefa *j* finaliza seu processamento na máquina *i* no tempo igual a *t* ou valor nulo se isto não ocorre. Por exemplo, a variável $x_{\_2\_1\_10}$ assume valor igual a 1 se a tarefa 2, finaliza o processamento na máquina 1 no tempo 10. Caso a tarefa 2 finalize o processamento na máquina 1 em qualquer outro instante de tempo, a variável $x_{\_2\_1\_10}$ assume valor nulo.

Notar que algumas combinações de índices são impossíveis. Ilustrando com a instância de teste, uma hipotética variável $x_{\_1\_1\_1}$ seria inviável pois, a tarefa 1 não pode ser finalizada na máquina 1 no tempo igual a 1 pois seu p_{11} é igual a 5. Esta variável de decisão simplesmente não existe e, por isso, sequer integra a formulação.

O número de variáveis na formulação é dado por:

$$\sum_{j=1}^{n}(S_1 - p_{1j} + 1) + \sum_{j=1}^{n} t_{max} - p_{1j} - p_{2j} + 1 \qquad (7)$$

Analisando a expressão (7), é possível verificar que o valor do t_{max} influencia diretamente no número de variáveis empregadas.

Para a instância de teste, o número de variáveis é igual a 58.

O objetivo de minimizar o somatório dos atrasos ponderados deve levar em consideração que cada tarefa *j* é finalizada uma e somente uma única vez em cada uma das duas máquinas, que cada máquina não poderá executar mais de uma tarefa ao mesmo tempo e que a seqüência do *flow shop* deve ser respeitada. A função objetivo e as restrições são apresentadas a seguir.

3.1.1.1 **Função Objetivo – Formulação 1**

$$Min\ Z = \sum_{j=1}^{n} \sum_{t\ =p_{1j}+\ p_{2j}}^{t_{max}} custo(j,t).x_{j2t} \qquad (8)$$

$$Onde: custo(j,t) = T_j.w_j$$

A função (8) consiste no somatório de todas as variáveis que representam possíveis finalizações de uma tarefa *j* na máquina 2. São incluídas somente variáveis com índices de finalização na máquina 2 pois o custo $T_j.w_j$ é uma função da data de finalização da tarefa *j* no sistema. Isto é o mesmo que o tempo de finalização da tarefa *j* na máquina 2.

O coeficiente de cada variável x_{j2t} consiste na multiplicação do atraso T_j, (diferença entre o tempo *t* de conclusão da tarefa e a data de término prometida d_j) pelo peso w_j associado à tarefa. Caso a variável represente a finalização de uma tarefa não atrasada, o coeficiente assume valor nulo.

Ilustrando com a instância de teste, a função objetivo pode ser observada no Quadro abaixo. Lembrar que, na instância de teste, t_{max} = 16.

$Min\ Z$ = 4 $x_{\_1\_2\_10}$ + 8 $x_{\_1\_2\_11}$ + 12 $x_{\_1\_2\_12}$ + 16 $x_{\_1\_2\_13}$ + 20 $x_{\_1\_2\_14}$ + 24 $x_{\_1\_2\_15}$ + 28 $x_{\_1\_2\_16}$ + 3 $x_{\_2\_2\_11}$ + 6 $x_{\_2\_2\_12}$ + 9 $x_{\_2\_2\_13}$ + 12 $x_{\_2\_2\_14}$ + 15 $x_{\_2\_2\_15}$ + 18 $x_{\_2\_2\_16}$ + 5 $x_{\_3\_2\_6}$ + 10 $x_{\_3\_2\_7}$ + 15 $x_{\_3\_2\_8}$ + 20 $x_{\_3\_2\_9}$ + 25 $x_{\_3\_2\_10}$ + 30 $x_{\_3\_2\_11}$ + 35 $x_{\_3\_2\_12}$ + 40 $x_{\_3\_2\_13}$ + 45 $x_{\_3\_2\_14}$ + 50 $x_{\_3\_2\_15}$ + 55 $x_{\_3\_2\_16}$

Quadro 1: função objetivo da Formulação 1 (instância de teste).

Os índices das variáveis foram separados por traços, de forma a não confundir as informações a respeito das tarefas, máquinas e tempos de finalização. Tomando o termo 20.$x_{\_1\_2\_14}$ como exemplo, tem-se que: o coeficiente apresenta valor igual a 20 pois consiste no produto entre o T_1 (tempo *t* de finalização da tarefa 1 subtraído de d_1, data prometida para tarefa 1, $14 - 9 = 5$) e o peso w_1 (o peso atribuído a tarefa 1 foi 4). Dessa forma, $w_1T_1 = 5.4 = 20$. Como já citado, a variável assume valor igual a 1 se a tarefa 1 finaliza seu processamento na máquina 2 no instante de tempo *t* igual a 14. Assume valor nulo caso a tarefa 1 seja finalizada num instante de tempo *t* diferente de 14.

O número de variáveis com coeficiente não nulo na função objetivo é igual a:

$$n.t_{max} - \sum_{j=1}^{n} d_j \qquad (9)$$

Na instância de teste, o número de variáveis com coeficiente não nulo é igual a 24.

3.1.1.2 **Primeiro Grupo de Restrições – Formulação 1**

$$\sum_{t=p_{1j}}^{S_1} x_{j1t} = 1 \quad \forall \quad j = 1, \dots, n \qquad (10)$$

$$\sum_{t=p_{1j}+p_{2j}}^{t_{max}} x_{j2t} = 1 \quad \forall \quad j = 1, \dots, n \qquad (11)$$

O primeiro grupo de restrições faz com que cada tarefa *j* seja finalizada exatamente uma única vez em cada uma das duas máquinas. Isto é feito determinando que, de todas as variáveis que representam términos possíveis de uma dada tarefa *j* numa dada máquina *i*, apenas uma delas tenha valor não nulo.

Estas restrições são necessárias uma vez que cada tarefa *j*, deve ser processada uma e somente uma única vez na máquina 1 e uma e somente uma única vez na máquina 2.

Ilustrando com a instância de teste, tomados a tarefa 1 e a máquina 1, tem-se:

$x\_1\_1\_5 + x\_1\_1\_6 + x\_1\_1\_7 + x\_1\_1\_8 + x\_1\_1\_9 + x\_1\_1\_10 + x\_1\_1\_11 + x\_1\_1\_12 = 1$

Quadro 2: exemplo de uma restrição do primeiro grupo

O número de restrições no grupo é dado por 2.n.

Para a instância de teste, o número de restrições do 1º grupo é igual a 6.

3.1.1.3 **Segundo Grupo de Restrições – Formulação 1**

$$\sum_{j=1}^{n} \sum_{t=t_1}^{t_1+p_{ij}-1} x_{jit} \leq 1 \quad \forall \quad t_1 = 0, \ldots, t_{max}, \quad \forall \quad i = 1, \ldots, m \qquad (12)$$

O segundo grupo de restrições impede que uma máquina processe simultaneamente mais de uma tarefa. Isto é restringido da seguinte forma: dadas as *n* tarefas e um intervalo de tempo limitado entre um valor de tempo t_1 (valor mínimo) e o tempo t_1 somado aos tempos de processamento de todas as *n* tarefas (valor máximo), não é possível que exista mais de uma variável que represente a finalização da tarefa *j* na máquina *i* com valor não nulo neste intervalo.

Notar que não basta restringir o fato de duas tarefas diferentes finalizarem seus processamentos na máquina *i* no mesmo tempo *t*. Utilizando a instância de teste, se a tarefa 1 finaliza seu processamento na máquina 1 no instante *t* = 7, a tarefa 2 nunca poderia finalizar seu processamento na máquina 1 em, por exemplo, *t* = 8 pois, seu p_{12}= 5. No mínimo, a tarefa 2 será finalizada em *t* = 12. O grupo de restrições acima leva em consideração estas limitações.

Para *i* = 1 e t_1 = 3, uma das restrições do segundo grupo pode ser visualizada no quadro abaixo. Nota-se que, dentre os tempos *t* considerados, somente uma variável poderá representar o término de uma tarefa *j* na máquina 1.

$x\_1\_1\_5 + x\_1\_1\_6 + x\_1\_1\_7 + x\_2\_1\_5 + x\_2\_1\_6 + x\_2\_1\_7 + x\_3\_1\_3 + x\_3\_1\_4 \leq 1$

Quadro 3: exemplo de uma restrição do segundo grupo

O número de restrições deste grupo é igual a:

$$(m.t_{max}) - \left(t_{max} - \sum_{j=1}^{n} p_{1j} \right) - m \qquad (13)$$

Justifica-se a expressão (13) da seguinte maneira: o produto do número de máquinas pelo t_{max} (há, de modo geral, uma restrição que elimine para cada máquina e dado tempo t_1

sobreposições de tarefas) deve ser subtraído por um número de equações onde t_1 compreenda um tempo onde o processamento de todas as tarefas na máquina 1 já tenha sido finalizado (lembrando que foi adotado como premissa que não há tempo ocioso na primeira máquina) Ilustrando: quando $t_1 = 13$, não existe nenhuma variável que represente finalizações de tarefas na máquina 1. Assim, as equações que restringem o processamento na máquina 1 para um valor t_1 superior a $\sum_{j=1}^{n}(p_{1j})$ não existem, pois não é possível finalizar uma tarefa na máquina 1 após o tempo *t* fornecido por este somatório. Fazendo $\left(t_{max} - \sum_{j=1}^{n} p_{1j}\right)$, chega-se ao número que deve ser diminuído de $(m.t_{max})$.

Também é subtraído o valor igual a *m* pois, a expressão $\sum_{j=1}^{n}\sum_{t=t_1}^{t_1+p_{ij}-1} x_{jit}$ não adiciona nenhuma restrição quando t_1 = 0, para cada uma das máquinas. Como exemplo, a variável $x_{\_1\_1\_4}$ não existe pois, p_{11} = 5. No tempo *t* = 4, seria impossível finalizar esta tarefa. Este é outro caso segundo o qual as variáveis simplesmente não existem.

O número de restrições no segundo grupo, utilizando a expressão (13), para a instância de teste, é igual a 26.

3.1.1.4 **Terceiro Grupo de Restrições – Formulação 1**

$$\sum_{t=t_1-p_{2j}+1}^{S_1} x_{j1t} + \sum_{t=0}^{t_1} x_{j2t} \leq 1 \quad \forall \quad j = 1, \dots, n\,; \quad \forall \quad t_1 = 0, \dots, t_{max} \qquad (14)$$

O terceiro grupo de restrições determina que as tarefas sigam a ordem de execução do *flow shop*: todas as tarefas devem ser processadas inicialmente na máquina 1 e, em seguida, na máquina 2. Isto é feito da seguinte forma: dentre todas as variáveis que representam possíveis finalizações na máquina 2, até um tempo t_1, somada a todas as variáveis que representam eventuais finalizações da tarefa na máquina 1 em tempos de t_1 até t_{max} (ou em tempos onde a tarefa não tivesse sido terminada na máquina 1, quando comparadas as demais variáveis com índice *i* = 2 na mesma equação) apenas uma variável poderá representar a finalização de uma tarefa *j* numa máquina *i* no tempo *t*.

Ilustrando com a instância de teste, restrição para t_1 = 10 e *j* = 3:

$x\_3\_1\_8 + x\_3\_1\_9 + x\_3\_1\_10 + x\_3\_1\_11 + x\_3\_1\_12 + x\_3\_2\_5 + x\_3\_2\_6 + x\_3\_2\_7 + x\_3\_2\_8 + x\_3\_2\_9 + x\_3\_2\_10 \leq 1$

Quadro 4: exemplo de uma restrição do terceiro grupo

Para que seja mantida a seqüência adequada da tarefa 3, em torno do tempo t_1, somente uma das variáveis acima poderá ser não nula.

O número de restrições no terceiro grupo é dado por:

$$n.(t_{max} + 1) \qquad (15)$$

Para a instância de teste, existem no terceiro grupo 51 restrições.

3.1.1.5 **Restrições de Integralidade e Não Negatividade – Formulação 1**

$$x_{jit} \in \{0,1\}, \quad \forall \quad j = 1, \dots, n; \quad \forall \quad t = 0, \dots, t_{max}; \quad \forall \quad i = 1, \dots, m \qquad (16)$$

3.1.1.6 **Formulação 1 – Completa**

Função Objetivo:

$$Min\ Z = \sum_{j=1}^{n} \sum_{t\ =p_{1j}+\ p_{2j}}^{t_{max}} custo(j,t).x_{j2t} \qquad (8)$$

$$Onde: custo(j,t) = T_j.w_j$$

Sujeito a:

$$\sum_{t=p_{1j}}^{S_1} x_{j1t} = 1 \quad \forall \quad j = 1, \dots, n \qquad (10)$$

$$\sum_{t=p_{1j}+p_{2j}}^{t_{max}} x_{j2t} = 1 \quad \forall \quad j = 1, \dots, n \qquad (11)$$

$$\sum_{j=1}^{n} \sum_{t=t_1}^{t_1+p_{ij}-1} x_{jit} \leq 1 \quad \forall \quad t_1 = 0, \ldots, t_{max}, \quad \forall \quad i = 1, \ldots, m \qquad (12)$$

$$\sum_{t=t_1-p_{2j}+1}^{S_1} x_{j1t} + \sum_{t=0}^{t_1} x_{j2t} \leq 1 \quad \forall \quad j = 1, \ldots, n\,; \quad \forall \quad t_1 = 0, \ldots, t_{max} \qquad (14)$$

Restrições de Integralidade e Não Negatividade:

$$x_{jit} \in \{0,1\}, \quad \forall \quad j = 1, \ldots, n; \quad \forall \quad t = 0, \ldots, t_{max}; \quad \forall \quad i = 1, \ldots, m \qquad (16)$$

3.1.2 Formulação 2

A diferença entre a Formulação 2 e a Formulação 1 é o fato de que a Formulação 2 elabora primordialmente um seqüenciamento para a última máquina do sistema. Qual a idéia por trás disto? A Formulação 2 baseia-se no Teorema 3, segundo o qual os problemas $F_2||\sum w_j T_j$ e $F_2|prmu|\sum w_j T_j$ são equivalentes. A lógica é estabelecer a seqüência da última máquina e, a partir disso, replicar este mesmo seqüenciamento para a 1º máquina do sistema. Dessa forma, é possível eliminar das variáveis de decisão o índice *i*, que representa uma determinada máquina. Nesta formulação, o índice *t* representará o tempo de conclusão da tarefa *j* no sistema (o que, de fato, é o mesmo que a conclusão da tarefa na segunda máquina). Define-se como variável de decisão a variável binária x_{jt} que assume valor unitário se a tarefa *j* conclui o processamento no sistema no tempo *t*, ou valor nulo se isto não ocorre. É importante lembrar que, assim como na Formulação 1, algumas combinações de índices não existem. Utilizando a instância de teste, a variável $x_{\_1\_7}$ não existe pois, $p_{11} + p_{12} = 9$. Assim, $x_{\_1\_7}$ nunca poderia representar a conclusão da tarefa 1 no sistema.

O número total de variáveis na Formulação 2 é dado por (17):

$$\sum_{j=1}^{n} t_{max} - p_{1j} - p_{2j} + 1 \qquad (17)$$

Para a instância de teste, o número de variáveis de decisão é igual a 31. Nota-se uma redução significativa no número de variáveis empregadas, quando comparada a Formulação 1

(58 variáveis). Na Formulação 1, há uma variável para cada máquina *i*, tarefa *j* e tempo de finalização possível *t*. Na Formulação 2, há uma variável para cada tarefa *j* e tempo possível de finalização *t*.

A redução de variáveis terá impacto no tempo de resolução do problema, quando se tomam algumas instâncias em comparação. Este reflexo será analisado um pouco mais a frente.

O objetivo de minimizar o somatório dos atrasos ponderados deve levar em consideração que cada tarefa *j* é finalizada uma e somente uma única vez no sistema, que a máquina 2 não poderá executar duas tarefas ao mesmo tempo e que a seqüência do *flow shop* deve ser respeitada. As variáveis de decisão só podem assumir valores iguais a zero ou um. Estas questões são impostas pelas restrições. A formulação 2 é descrita a seguir:

3.1.2.1 **Função Objetivo – Formulação 2**

$$Min\ Z = \sum_{j=1}^{n} \sum_{t=p_{1j}+\,p_{2j}}^{t_{max}} custo(j,t).x_{jt} \qquad (18)$$

$$Onde: custo(j,t) = T_j.w_j$$

A função (18) consiste no somatório de todas as variáveis que representam possíveis tempos de conclusão da tarefa *j* no sistema.

O coeficiente de cada variável x_{jt} consiste na multiplicação do atraso T_j, (diferença entre o tempo *t* de conclusão da tarefa e a data prometida d_j) pelo peso w_j associado à tarefa. Caso a variável represente a finalização de uma tarefa não atrasada, o coeficiente assume valor nulo.

Ilustrando com a instância de teste, a função objetivo pode ser observada no Quadro abaixo.

$Min\ Z = 4\ x\_1\_10 + 8\ x\_1\_11 + 12\ x\_1\_12 + 16\ x\_1\_13 + 20\ x\_1\_14 + 24\ x\_1\_15 + 28\ x\_1\_16 + 3\ x\_2\_11 + 6\ x\_2\_12 + 9\ x\_2\_13 + 12\ x\_2\_14 + 15\ x\_2\_15 + 18\ x\_2\_16 + 5\ x\_3\_6 + 10\ x\_3\_7 + 15\ x\_3\_8 + 20\ x\_3\_9 + 25\ x\_3\_10 + 30\ x\_3\_11 + 35\ x\_3\_12 + 40\ x\_3\_13 + 45\ x\_3\_14 + 50\ x\_3\_15 + 55\ x\_3\_16$

Quadro 5: exemplo de função objetivo da Formulação 2 utilizando a instância de teste

3.1.2.2 **Primeiro Grupo de Restrições – Formulação 2**

$$\sum_{t=p_{1j}+p_{2j}}^{t_{max}} x_{jt} = 1 \quad \forall \quad j = 1, \dots, n \qquad (19)$$

O primeiro grupo de restrições (19) faz com que cada tarefa *j* seja finalizada exatamente uma única vez na última máquina. Isto é feito determinando que, de todas as variáveis que representam términos possíveis de uma dada tarefa *j* no sistema, uma e somente uma única delas tenha valor não nulo. Este grupo de restrições tem o mesmo propósito do 1º grupo de restrições da Formulação 1, sendo seu funcionamento similar.

Ilustrando com a instância de teste, tomados a tarefa 1, tem-se que:

$x\_1\_9 + x\_1\_10 + x\_1\_11 + x\_1\_12 + x\_1\_13 + x\_1\_14 + x\_1\_15 + x\_1\_16 = 1$

Quadro 6: exemplo de restrição de finalização das tarefas com a instância de teste

O número de restrições no grupo é dado por *n*, pois para toda tarefa *j*, deve haver um somatório que restrinja a uma única vez a finalização da tarefa *j* no sistema.

Para a instância de teste, o número de restrições que garantem a finalização de uma dada tarefa uma e somente uma única vez é igual a 3.

3.1.2.3 **Segundo Grupo de Restrições – Formulação 2**

$$\sum_{j=1}^{n} \sum_{t=t_1}^{t_1+p_{2j}-1} x_{jt} \leq 1 \quad \forall \quad t_1 = 0, \dots, t_{max} \qquad (20)$$

O segundo grupo de restrições impede que a máquina 2 processe simultaneamente mais de uma tarefa. Isto é restringido da seguinte forma: dada as *n* tarefas e um intervalo de tempo limitado entre um valor de tempo t_1 (valor mínimo) e o tempo t_1 somado aos tempos de processamento de todas as *n* tarefas (valor máximo), não é possível que exista mais de uma

variável que represente a finalização da tarefa *j* na máquina *i = 2* com valor não nulo neste intervalo.

Igualmente à Formulação 1, notar que não basta restringir o fato de duas tarefas diferentes finalizarem seus processamentos na máquina 2 no mesmo tempo *t*. Utilizando a instância de teste, se a tarefa 1 finaliza seu processamento no sistema no instante *t* = 9, a tarefa 3 nunca poderia finalizar seu processamento no sistema em, por exemplo, *t* = 10 pois, seu p_{12}= 3. No mínimo, a tarefa 3 será finalizada em *t* = 12.

Para t_1 = 13, uma das restrições do segundo grupo pode ser visualizada no quadro abaixo. Nota-se que, dentre os tempos *t* considerados, somente uma variável poderá representar a conclusão da tarefa *j* no sistema.

$x\_1\_13 + x\_1\_14 + x\_1\_15 + x\_1\_16 + x\_2\_13 + x\_3\_13 + x\_3\_14 + x\_3\_15 \leq 1$

Quadro 7: exemplo de restrição, utilizando a instância de teste, para restringir o processamento na máquina 2 a apenas uma tarefa de cada vez.

O número de restrições no segundo grupo é dado pela expressão (21):

$$t_{max} - min(p_{1j}) \qquad (21)$$

Há, de modo geral, uma restrição para cada instante t_1. Contudo, para aqueles instantes onde não existem variáveis de decisão (ou seja, para os tempos onde seria impossível finalizar uma tarefa *j*), não há restrição alguma. Este tempo é igual ao menor tempo de processamento na máquina 1.

Para a instância de teste, o número de restrições do grupo 2 é igual a 14.

3.1.2.4 **Terceiro Grupo de Restrições – Formulação 2**

$$\sum_{j=1}^{n} \sum_{t=0}^{t_1+p_{2j}} p_{1j}.x_{jt} \leq t_1 \quad \forall \quad t_1 = 0, \dots, t_{max} \qquad (22)$$

O terceiro grupo de restrições encarrega-se de manter o seqüenciamento característico do ambiente *flow shop*, (tarefas sendo processadas inicialmente na máquina 1 e, em seguida, na máquina 2). As restrições deste grupo, apesar de voltarem-se ao mesmo propósito do 3º grupo da Formulação 1, utilizam lógica diferente de elaboração.

A montagem das restrições utiliza o raciocínio a seguir: tomado um tempo t_1 o somatório de todos os tempos de processamento na máquina 1 das tarefas que eventualmente já tenham sido finalizadas no sistema deve ser menor ou igual ao tempo t_1. Ou seja, para que a tarefa possa ser finalizada até $t_1 + p_{2j}$, esta deve ter sido finalizada na máquina 1 até o tempo t_1.

Utilizando na instância de teste t_1 = 10, tem-se a seguinte restrição:

5 x_1_9 + 5 x_1_10 + 5 x_1_11 + 5 x_1_12 + 5 x_1_13 + 5 x_1_14 + 5 x_2_6 + 5 x_2_7 + 5 x_2_8 + 5 x_2_9 + 5 x_2_10 + 5 x_2_11 + 2 x_3_5 + 2 x_3_6 + 2 x_3_7 + 2 x_3_8 + 2 x_3_9 + 2 x_3_10 + 2 x_3_11 + 2 x_3_12 + 2 x_3_13 ≤ 10

Quadro 8: exemplo de restrição para manter o seqüenciamento correto entre as máquinas

O número de restrições responsáveis por manter o seqüenciamento correto das tarefas é igual a:

$$t_{max} - min(p_{1j}) + 1 \qquad (23)$$

Aqui, o princípio que orienta o número de restrições é o mesmo da restrição anterior. Devem ser incluídas restrições para todos os tempos t_1 considerados, exceto para àqueles onde não há variável de decisão viável.

Tomado como exemplo a instância de teste, o número de restrições é igual a 15.

3.1.2.5 **Restrições de Integralidade e Não Negatividade – Formulação 2**

$$x_{jt} \in \{0,1\} \quad \forall \quad j = 1, \dots, n. \quad \forall \quad t = 0, \dots, t_{max} \qquad (24)$$

3.1.2.6 **Formulação 2 – Completa**

Função Objetivo:

$$Min\ Z = \sum_{j=1}^{n} \sum_{t=p_{1j}+\ p_{2j}}^{t_{max}} custo(j,t).\, x_{jt} \qquad (18)$$

$$Onde: custo(j,t) = \ T_j.\, w_j$$

Sujeito a:

$$\sum_{t=p_{1j}+p_{2j}}^{t_{max}} x_{jt} = 1 \quad \forall \quad j = 1, \dots, n \qquad (19)$$

$$\sum_{j\ =\ 1}^{n} \sum_{t=t_1}^{t_1+p_{2j}-\ 1} x_{jt}\ \leq 1 \quad \forall \quad t_1 = 0, \dots, t_{max} \qquad (20)$$

$$\sum_{j=1}^{n} \sum_{t=0}^{t_1+p_{2j}} p_{1j}.\, x_{jt}\ \leq t_1 \quad \forall \quad t_1 = 0, \dots, t_{max} \qquad (22)$$

Restrições de Integralidade e Não Negatividade:

$$x_{jt}\ \in \{0,1\} \quad \forall \quad j = 1, \dots, n. \quad \forall \quad t = 0, \dots, t_{max} \qquad (24)$$

3.1.3 Formulação 3

A Formulação 3 assemelha-se bastante à Formulação 2, partindo do princípio que os problemas $F_2||\sum w_j T_j$ e $F_2|prmu|\sum w_j T_j$ são equivalentes. Emprega as variáveis x_{jt}, que assim como na Formulação 2, representam o tempo *t* de finalização da tarefa *j* no sistema. Contudo, de forma a reduzir a repetição de variáveis nas restrições, notadamente naquelas restrições que garantem o cumprimento da seqüência correta das tarefas ao longo das máquinas, incluem-se na formulação as variáveis auxiliares y_{jt} as quais representam o quanto da tarefa *j* já pode ter sido processada na máquina 1, num dado instante *t*. Estas variáveis podem assumir valores inteiros (não somente binários).

3.1.3.1 **Função Objetivo – Formulação 3**

A função objetivo da Formulação 3 é representada pela função (18), ou seja, é idêntica a função objetivo empregada na Formulação 2.

3.1.3.2 **Primeiro Grupo de Restrições – Formulação 3**

O primeiro grupo de restrições da Formulação 3 é igual ao primeiro grupo de restrições da Formulação 2 (expressão 19). O objetivo é fazer com cada tarefa *j* seja finalizada exatamente uma única vez na última máquina. Isto é feito determinando que, de todas as variáveis que representam términos possíveis de uma dada tarefa *j* no sistema, uma e somente uma única delas tenha valor não nulo. O número de restrições neste grupo é igual a *n*. Para a instância de teste, existem 3 restrições neste grupo.

3.1.3.3 **Segundo Grupo de Restrições – Formulação 3**

Outro grupo igual ao da Formulação 2 (expressão 20). Impede que a máquina 2 execute mais de uma tarefa simultaneamente. O número de restrições deste grupo também é dado por (21). Para a instância de teste, o número de restrições do grupo 2 é igual a 14.

3.1.3.4 **Terceiro Grupo de Restrições – Formulação 3**

$$(y_{jt_1} - y_{jt_1-1}) - \sum_{t=t_1+p_{2j}}^{t_1+(p_{1j}+p_{2j})-1} x_{jt} = 0 \quad \forall \quad j = 1, \dots, n\,; \quad \forall \quad t_1 = 1, \dots, \left(t_{max} - \left(p_{1j} + p_{2j}\right) + 1\right) \qquad (25)$$

O terceiro grupo de restrições encarrega-se de manter o seqüenciamento correto das *n* tarefas (processamento na máquina 1 e, depois, processamento na máquina 2). Neste grupo, a variável y_{jt} atua no sentido de substituir uma série de variáveis x_{jt} em cada uma das restrições. Assim, diminui-se o número total de variáveis nas restrições. A lógica envolvida na

equação é a seguinte: para um dado tempo t_1 e tarefa *j*, a diferença entre o quanto a tarefa *j* já foi processada na máquina 1 em relação ao tempo *t* imediatamente anterior (esta diferença será igual a 0 ou 1) é igual ao somatório das variáveis x_{jt} que representam possíveis finalizações da tarefa *j*, considerando o tempo t_1.

Analisando com a instância de testes, para o tempo t_1= 5, *j* = 3 tem-se:

y_3_5 - y_3_4 - x_3_8 - x_3_9 = 0

Quadro 9: exemplo de restrição do terceiro grupo, para t_1 = 5

Notar que se a tarefa 3 não foi iniciada na máquina 1 até o tempo 5, seria impossível a tarefa ser concluída no tempo *t* = 9, pois $p_{13} + p_{23}$ = 5, o que satisfaria a igualdade.

Se a tarefa foi finalizada no tempo *t* = 9, esta deve, no mínimo, ter sido iniciada na máquina 1 no início de *t* = 5. Ou, caso tenha sido concluída em t = 8, esta foi iniciada na máquina 1, no mínimo, no início de *t* = 4.

Uma questão fundamental para entender estas restrições é observar que os valores atribuídos à y_{jt} são funções dos valores de x_{jt}, e dos valores permitidos pelo 4º grupo de restrições, que será visto logo a seguir.

O número de restrições deste grupo é dado por (26):

$$\sum_{j=1}^{n} t_{max} - (p_{1j} + p_{2j}) + 1 \qquad (26)$$

Justifica-se a expressão (26) da seguinte maneira: deve haver uma restrição para cada intervalo onde é possível finalizar a tarefa *j* no sistema, considerando a finalização do processamento na máquina 1.

Para a instância de teste, existem ao todo 31 restrições neste grupo.

3.1.3.5 **Quarto Grupo de Restrições – Formulação 3**

$$\sum_{j=1}^{n} y_{jt} \leq t \qquad \forall \quad t = 1, \dots, t_{max} \qquad (27)$$

O quarto grupo de restrições limita o somatório dos valores das variáveis y_{jt}, para um dado tempo *t*. Isto precisa ser feito uma vez que, o somatório dos processamentos na máquina 1 de todas as tarefas, realizados até o tempo *t* não pode ser, de fato, maior do que o próprio tempo *t*.

Utilizando a instância de teste, para *t* = 5, tem-se um exemplo deste grupo de restrições:

$y\_1\_5 + y\_2\_5 + y\_3\_5 \leq 5$

Quadro 10: exemplo de restrição do quarto grupo, para t_1 = 5

O número de restrições deste grupo é dado por:

$$t_{max} - min(p_{1j}) \qquad (28)$$

Explica-se o número de restrições, dado por (28), da seguinte forma: deve existir uma restrição que limite o somatório dos valores das variáveis y_{jt} para cada tempo *t*, exceto para aqueles tempos *t* onde todas as tarefas necessariamente já deveriam ter sido processadas na máquina 1.

Utilizando a instância de teste, tem-se que o número de restrições do 4º grupo é igual a 14.

3.1.3.6 **Restrições de Integralidade e Não Negatividade – Formulação 3**

$$x_{jt} \in \{0,1\} \quad \forall \quad j = 1, \dots, n. \qquad \forall \qquad t = 0, \dots, t_{max} \qquad (24)$$

$$y_{jt} \geq 0, \quad \forall \quad j = 1, \dots, n. \qquad \forall \qquad t = 0, \dots, t_{max} \qquad (29)$$

3.1.3.7 **Formulação 3 – Completa**

Função Objetivo:

$$Min\ Z = \sum_{j=1}^{n} \sum_{t=p_{1j}+\ p_{2j}}^{t_{max}} custo(j,t).\ x_{jt} \qquad (18)$$

$$Onde: custo(j,t) =\ T_j.\ w_j$$

Sujeito a:

$$\sum_{t=p_{1j}+p_{2j}}^{t_{max}} x_{jt} = 1 \qquad \forall \qquad j = 1, \dots, n \qquad (19)$$

$$\sum_{j\ =\ 1}^{n} \sum_{t=t_1}^{t_1+p_{2j}-\ 1} x_{jt}\ \leq 1 \qquad \forall \qquad t_1 = 0, \dots, t_{max} \qquad (20)$$

$$(y_{jt_1} - y_{jt_1-1}) - \sum_{t=t_1+p_{2j}}^{t_1+(p_{1j}+p_{2j})-1} x_{jt} = 0 \qquad (25)$$

$$\forall \quad j = 1, \dots, n\ ; \quad \forall t_1 = 1, \dots, \left(t_{max} - \left(p_{1j} + p_{2j}\right) + 1\right)$$

$$\sum_{j=1}^{n} y_{jt} \leq t \qquad \forall \quad t = 1, \dots, t_{max} \qquad (27)$$

Restrições de Integralidade e Não Negatividade:

$$x_{jt}\ \in \{0,1\} \quad \forall \quad j = 1, \dots, n. \qquad \forall \qquad t = 0, \dots, t_{max} \qquad (24)$$

$$y_{jt}\ \geq 0, \ \forall \quad j = 1, \dots, n. \qquad \forall \qquad t = 0, \dots, t_{max} \qquad (29)$$

3.2 FORMULAÇÕES COM VARIÁVEIS CONTENDO ÍNDICE DE POSIÇÃO

O segundo grupo de formulações exclui da variável de decisão o índice *t* de tempo de conclusão da tarefa e inclui um índice de posição, que será denominado por *k*. O índice de posição informa a ordem, quando comparada às demais, que uma dada tarefa é executada. A exclusão do índice de tempo *t* das variáveis tem por objetivo reduzir o número total de variáveis de decisão empregadas na formulação.

Para a elaboração das formulações com variáveis contendo índice de posição, utilizou-se como ponto de partida uma formulação para o problema $F_m|prmu|C_{\max}$, desenvolvida por Wilson (1989), (que na verdade é uma melhoria de uma formulação proposta por Wagner em 1959). Esta formulação contendo variáveis com índice de posição, denominada por Formulação de Wilson, será a apresentada a seguir.

Função Objetivo:

$$Min\ Z = C_m^n \qquad (30)$$

Sujeito a:

$$\sum_{k=1}^{n} x_j^k = 1 \quad \forall \quad j = 1, \dots, n \qquad (31)$$

$$\sum_{j=1}^{n} x_j^k = 1 \quad \forall \quad k = 1, \dots, n \qquad (32)$$

$$P_i^k = \sum_{j=1}^{n} p_{ij} x_j^k \quad \forall \quad i = 1, \dots, m; \quad \forall \quad k = 1, \dots, n \qquad (33)$$

$$C_i^k = S_i^k + P_i^k \quad \forall \quad k = 1, \dots, n; \quad \forall \quad i = 1, \dots, m \qquad (34)$$

$$S_i^k \geq C_i^{k-1} \quad \forall \quad k = 2, \dots, n \quad \forall \quad i = 1, \dots, m \qquad (35)$$

$$S_i^k \geq C_{i-1}^k \quad \forall \quad k = 1, \dots, n \quad \forall \quad i = 2, .., m \qquad (36)$$

$$S_i^k = 0 \quad \forall \quad k = 1 \quad \forall \quad i = 1 \qquad (37)$$

Restrições de Integralidade e Não Negatividade:

$$x_j^k \in \{0,1\}, \quad \forall \quad j = 1, \dots, n, \quad \forall \quad k = 1, \dots, n \qquad (38)$$

$$S_i^k \geq 0 \quad \forall \quad i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (39)$$

$$C_i^k \geq 0 \quad \forall \quad i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (40)$$

$$P_i^k \geq 0 \quad \forall \quad i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (41)$$

A variável binária x_j^k assume valor unitário se a tarefa *j* ocupa a posição *k* (assumindo valor nulo no caso contrário). A variável contínua C_i^k representa o tempo de finalização da tarefa que ocupa a posição *k* na máquina *i*. A variável contínua S_i^k representa o tempo de início na máquina *i* da tarefa que ocupa a posição *k*. A variável contínua P_i^k representa o tempo de processamento na máquina *i* da tarefa que ocupa a posição *k*.

A função objetivo (30) busca a minimização do tempo de conclusão da última tarefa, "tarefa *n*", na última máquina, "máquina *m*" (conforme exposto na introdução, este objetivo a ser perseguido é conhecido por *Makespan*). As restrições de atribuição (31) e (32) fazem com que uma tarefa *j* ocupe uma e somente uma posição *k* e uma posição *k* seja ocupada por uma e somente uma tarefa *j*. As restrições (33) atribuem o tempo de processamento P_i^k da tarefa que ocupa a posição *k* na máquina *i*. As restrições (34) encarregam-se de associar o tempo de finalização de uma dada tarefa que ocupa a posição *k* na máquina *i* como sendo igual ao tempo de início desta tarefa na máquina *i*, somada ao tempo de processamento necessário para que a tarefa seja finalizada na máquina *i*, As restrições (35) fazem com que duas tarefas não sejam executadas simultaneamente na mesma máquina, ao imporem que o tempo de início da tarefa *k* seja superior ou igual ao tempo de finalização da tarefa que ocupa a posição *k*-1, para uma dada máquina *i*. As restrições (36) submetem todas as tarefas à seqüência característica do *flow shop*, processamento inicial na máquina 1, para só depois ser processadas na máquina 2 (para um *flow shop* com duas máquinas). A restrição (37) impõe que a primeira tarefa a ser executada (k =1) seja iniciada no instante t = 0 na máquina 1.

3.2.1 Formulação 4

Ao analisar a formulação de Wilson, desenvolvida para o problema da minimização do *Makespan*, é possível notar o emprego de variáveis que cumprem funções diferentes, quando comparadas entre si. Nas Formulações 1 e 2, existia apenas um tipo de variável (x_{jit}, x_{jt} respectivamente) sendo que na Formulação 3 definiu-se, além da variável x_{jt}, a variável auxiliar y_{jt}. Basicamente, nas Formulações 1, 2 e 3, o que se têm é um conjunto de variáveis que cumprem a mesma função, possuindo diferentes índices. Por sua vez, a Formulação de Wilson apresenta variáveis que indicam o início de uma tarefa *k* numa máquina *i*, outras variáveis representam o tempo de finalização de uma tarefa que ocupa a posição *k* numa máquina *i*. Existem ainda as variáveis x_j^k e P_i^k.

No caso da Formulação 4, além das variáveis utilizadas na formulação de Wilson, são incluídas na formulação variáveis contínuas "T_j" que representam eventuais atrasos das *n* tarefas (tempo de finalização subtraído da data de término prometida).

O propósito de minimizar o somatório dos atrasos ponderados leva em consideração as mesmas restrições anteriormente impostas nas Formulações 1, 2 e 3: cada tarefa *j* é finalizada uma e somente uma única vez em cada uma das duas máquinas, cada máquina não poderá executar mais de uma tarefa ao mesmo tempo e a seqüência do *flow shop* deve ser respeitada. Entretanto, a maneira como isto é implementado na Formulação 4 é bastante diferente quando comparado às formulações anteriores (de fato, isto é justificável na medida em que as variáveis de decisão empregadas são diferentes).

O número de variáveis x_j^k empregadas é dado pelo produto do número total de tarefas *n* pelo total de posições *k* possíveis. Dado que toda tarefa *j* deve ocupar uma e somente uma posição *k*, o número de posições *k* possíveis é igual a *n*. Dessa forma, tem-se que o número de variáveis x_j^k é igual a n^2.

O número de variáveis T_j utilizadas é igual a *n*, uma vez que para todas as *n* tarefas, poderá haver um atraso em relação à data de término prometida.

O número de variáveis C_i^k na formulação é dado por *n.m* pois, para cada máquina *i*, todas as *k* posições devem ser preenchidas. Utilizando o mesmo raciocínio, define-se também o número de variáveis do tipo S_i^k sendo igual a *n.m*.

Por último, o número de variáveis P_i^k é dado também por *n.m*, uma vez que qualquer tarefa pode ocupar a posição *k* na máquina *i*.

Para a instância de teste, o número de variáveis de decisão é igual a 30 (9 variáveis do tipo x_j^k, 3 variáveis do tipo T_j, 6 variáveis do tipo C_i^k, 6 variáveis do tipo S_i^k e 6 variáveis do tipo P_i^k).

Os componentes da Formulação 4 são apresentados a seguir:

3.2.1.1 **Função Objetivo – Formulação 4**

$$Min \sum_{j=1}^{n} w_j T_j \qquad (42)$$

A função (42) busca, como não poderia deixar de ser, a minimização do somatório dos atrasos ponderados.

Utilizando a instância de teste, tem-se a ilustração da função objetivo no quadro 11.

$4\ T_1 + 3\ T_2 + 5\ T_3$

Quadro 11: exemplo da função objetivo da Formulação 4, utilizando a instância de teste

3.2.1.2 **Primeiro Grupo de Restrições – Formulação 4**

$$T_j \geq C_2^k - d_j - M_j + M_j x_j^k \quad \forall \quad j = 1, \dots, n; \quad \forall \quad k = 1, \dots, n \qquad (43)$$

O primeiro grupo de restrições (43) faz com que o atraso (*Tardiness*) seja maior ou igual à diferença entre o tempo de finalização da tarefa *j* na segunda máquina e a data de término prometida. A forma escolhida para garantir isto (dado que a variável que informa o tempo de conclusão C_i^k não contém índice *j* de tarefa) é a seguinte: define-se uma constante M_j, tal que $M_j = t_{max} - d_j$. Esta constante aparece duas vezes na restrição, primeiramente com valor negativo e, depois multiplicada pela variável x_j^k (binária). Se a tarefa *j* ocupar a posição *k*, a constante M_j é subtraída por ela mesma, o que elimina seu impacto na restrição. Se a tarefa *j* não ocupar a posição *k* , o valor $-M_j$ faz com que T_j seja um valor mínimo.

Utilizando a instância de teste, para $j = 1$ e $k = 1$ tem-se um exemplo deste grupo de restrições:

$$T_1 - C_{\_2\_1} - 7\, x_{\_1\_1} \geq -15$$

Quadro 12: exemplo de restrição do primeiro grupo, utilizando a instância de teste

O número de restrições deste grupo é dado por n^2, uma vez que existe uma restrição para cada tarefa *j* e posição *k*

3.2.1.3 **Segundo Grupo de Restrições – Formulação 4**

O segundo grupo contém restrições idênticas as restrições (31) e (32) da Formulação de Wilson. Fazem com que uma e somente uma posição *k* seja ocupada por uma e somente uma tarefa *j* e uma e somente uma tarefa *j* ocupe uma e somente uma posição *k*. Ilustrando com a instância de teste, tem-se para $j = 1$ e $k = 1$ as equações do quadro 13 abaixo:

$$x_{\_1\_1} + x_{\_2\_1} + x_{\_3\_1} = 1$$

$$x_{\_1\_1} + x_{\_1\_2} + x_{\_1\_3} = 1$$

Quadro 13: exemplo de restrição do segundo grupo, utilizando a instância de teste

O número de restrições no segundo grupo é dado por $2n$, pois existe uma restrição para cada posição possível *k* e cada tarefa *j* existente.

3.2.1.4 **Terceiro Grupo de Restrições – Formulação 4**

O terceiro grupo de restrições contém as equações representadas pela expressão (33) da Formulação de Wilson. Atribui à variável P_i^k o tempo de processamento na máquina *i* da tarefa que ocupa a posição *k*. Isto é feito na medida em que, se a tarefa *j* ocupa a posição *k*, então a variável x_j^k assume valor 1 e, dessa forma, o tempo de processamento na máquina *i* da

tarefa que ocupa a posição *k* passa a ser p_{ij}. Sendo *i* = 1 e *k* = 3, segue a ilustração de uma restrição deste tipo, quando utilizada a instância de teste:

$$P\_1\_3 - 5\ x\_1\_3 - 5\ x\_2\_3 - 2\ x\_3\_3 = 0$$

Quadro 14: exemplo de restrição do terceiro grupo, utilizando a instância de teste

O número de restrições do terceiro grupo é dado por $n.m$. Uma vez que existe um tempo de processamento para cada máquina *i* e possível posição *k*.

3.2.1.5 **Quarto Grupo de Restrições – Formulação 4**

O quarto grupo de restrições, representado por uma expressão idêntica a expressão (34) da Formulação de Wilson, amarra os tempos de início e de finalização de uma dada tarefa que ocupa a posição *k*, numa máquina *i*. Isto é feito da seguinte maneira: como a variável C_i^k representa o tempo de conclusão da tarefa que ocupa a posição *k* na máquina *i*, esta pode ser escrita como sendo igual ao tempo de início do seu processamento na mesma máquina (variável S_i^k), somada ao tempo de processamento P_i^k. De maneira mais direta, o tempo de conclusão é igual ao tempo de início somado ao tempo de processamento, para uma tarefa que ocupa a posição *k* na máquina *i*. Utilizando a instância de teste, para *i* = 2 e *k* = 3, tem-se a ilustração do quadro 15:

$$-\ C\_2\_3 + S\_2\_3 + P\_2\_3 = 0$$

Quadro 15: exemplo de restrição do quarto grupo, utilizando a instância de teste

O número de restrições do quarto grupo é dado por $n.m$, dado que existe um tempo de conclusão para cada tarefa que ocupa uma posição *k* e para cada máquina *i*.

3.2.1.6 **Quinto Grupo de Restrições – Formulação 4**

O quinto grupo de restrições, representado por uma expressão idêntica a expressão (35) da Formulação de Wilson, faz com que, para uma mesma máquina *i*, uma determinada tarefa que ocupa a posição *k* só seja iniciada quando a tarefa que ocupa a posição imediatamente anterior seja completada. Isto é feito na medida em que o tempo de início da tarefa que ocupa a posição *k*, na máquina *i*, deve ser igual ou superior ao tempo de conclusão da tarefa que ocupa a posição imediatamente anterior. Utilizando a instância de teste, para $i = 2$ e $k = 3$, tem-se a seguinte ilustração:

$- C_{\_2\_2} + S_{\_2\_3} \geq 0$

Quadro 16: exemplo de restrição do quinto grupo, utilizando a instância de teste

O número de restrições do quinto grupo é dado por (44):

$$(n-1)m \qquad (44)$$

Pois existe uma restrição para cada máquina *i* e tarefa que ocupa a posição *k*, exceto para a tarefa que ocupa a primeira posição.

3.2.1.7 **Sexto Grupo de Restrições – Formulação 4**

O sexto grupo de restrições, representado pela expressão (36) da Formulação de Wilson, faz com que as tarefas sigam a seqüência determinada pelo ambiente de máquinas característico do *flow shop* (processamento primeiro na máquina 1, em seguida na máquina subseqüente, e assim sucessivamente, até a máquina m). Neste sentido, tem-se que a tarefa que ocupa a posição *k* na máquina *i*, tem seu tempo de início de processamento maior ou igual ao tempo de finalização desta mesma tarefa na máquina anterior. Ilustrando com a instância de teste, para $i = 2$ e $k = 1$, tem-se a seguinte restrição:

$- C_{\_1\_1} + S_{\_2\_1} \geq 0$

Quadro 17: exemplo de restrição do sexto grupo, utilizando a instância de teste.

O número de restrições do último grupo é dado por:

$$(m-1)n \qquad (45)$$

Uma vez que existe uma restrição, a partir da segunda máquina em diante, para cada tarefa *j* que ocupa uma possível posição *k*.

3.2.1.8 **Sétimo Grupo de Restrições – Formulação 4**

Este grupo, representado por apenas uma equação de restrição, idêntica a equação (37) da Formulação de Wilson, transmite que a tarefa que ocupa a primeira posição (k = 1) deve iniciar seu processamento na primeira máquina do sistema (i = 1) no instante t = 0. A ilustração desta restrição, independentemente da instância a ser utilizada, é igual a S_1^1 = 0.

3.2.1.9 **Restrições de Integralidade e Não Negatividade – Formulação 4**

Incluem-se no grupo das restrições de integralidade e não negatividade as expressões (38), (39), (40) e (41) da Formulação de Wilson. Alem destas, adiciona-se também a restrição (46), exposta abaixo:

$$T_j \geq 0 \quad \forall \quad j = 1, \dots, n \qquad (46)$$

A Formulação 4 pode ser observada na íntegra a seguir:

3.2.1.10 **Formulação 4 - Completa**

Função Objetivo:

$$Min \sum_{j=1}^{n} w_j T_j \qquad (42)$$

Sujeito a:

$$T_j \geq C_2^k - d_j - M_j + M_j x_j^k \ \ \forall \ \ j = 1, \dots, n; \ \ \forall \ \ k = 1, \dots, n \qquad (43)$$

$$\sum_{k=1}^{n} x_j^k = 1 \ \ \forall \ \ j = 1, \dots, n \qquad (31)$$

$$\sum_{j=1}^{n} x_j^k = 1 \ \ \forall \ \ k = 1, \dots, n \qquad (32)$$

$$P_i^k = \sum_{j=1}^{n} p_{ij} x_j^k \quad \forall \quad i = 1, \dots, m; \quad \forall \quad k = 1, \dots, n \qquad (33)$$

$$C_i^k = S_i^k + P_i^k \ \ \forall \ \ k = 1, \dots, n; \ \ \forall \ \ i = 1, \dots, m \qquad (34)$$

$$S_i^k \geq \ C_i^{k-1} \ \ \forall \ \ k = 2, \dots, n \quad \forall \quad i = 1, \dots, m \qquad (35)$$

$$S_i^k \geq \ C_{i-1}^k \ \ \forall \ \ k = 1, \dots, n \ \ \forall \ \ i = 2, .., m \qquad (36)$$

$$S_1^1 = 0 \ \forall \ \ k = 1 \ \ e \ \ i = 1 \qquad (37)$$

Restrições de Integralidade e Não Negatividade:

$$x_j^k \ \in \ \{0,1\}, \ \ \forall \quad j = 1, \dots, n, \quad \forall \quad k = 1, \dots, n \qquad (38)$$

$$S_i^k \ \geq 0 \quad \forall \ \ i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (39)$$

$$C_i^k \ \geq 0 \quad \forall \ \ i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (40)$$

$$P_i^k \ \geq 0 \ \forall \ \ i = 1, \dots, m, \ \ \forall \ \ k = 1, \dots, n \qquad (41)$$

$$T_j \ \geq 0 \ \ \forall \quad j = 1, \dots, n \qquad (46)$$

3.2.2 Formulação 5

Ao longo deste trabalho, ao se buscar estabelecer uma formulação para o $F_2||\sum w_j T_j$ que não empregasse índices de tempo *t* nas variáveis de decisão, foi desenvolvida uma

formulação para o problema da minimização do somatório do *Tardiness*. Na notação de três campos, representa-se este problema da seguinte forma: $F_2||\sum T_j$. Como este problema pode ser tomado como um caso especial do problema $F_2||\sum w_j T_j$, se $w_j = 1,\ \forall\ j = 1, \dots, n$, a formulação para minimização do somatório do *Tardiness* será apresentada a seguir, sendo denominada genericamente por Formulação 5.

A Formulação 5 também foi baseada na Formulação de Wilson. Contudo, diferentemente da Formulação 4, a Formulação 5 não utiliza o método "M grande". Definem-se as variáveis x_j^k, C_i^k, S_i^k e P_i^k, com os mesmos papéis que estas variáveis desempenham na Formulação de Wilson. Além destas, define-se a variável T_k, que representa o atraso da tarefa que ocupa a posição *k* na seqüência de tarefas e a variável d_j^k, que representa a data de término prometida da tarefa *j* que ocupa a posição *k*.

A Formulação 5 é apresentada a seguir:

3.2.2.1 **Função Objetivo – Formulação 5**

$$Min \sum_{k=1}^{n} T_k \qquad (47)$$

A função objetivo preocupa-se em minimizar o somatório dos atrasos de todas as tarefas, sendo o atraso da tarefa que ocupa a posição *k* representado pela variável T_K.

3.2.2.2 **Primeiro Grupo de Restrições – Formulação 5**

$$T_k \geq C_2^k - \sum_{j=1}^{n} d_j^k \quad \forall \quad k = 1, \dots, n \qquad (48)$$

O primeiro grupo de restrições encarrega-se de associar à variável que representa o atraso da tarefa que ocupa a posição *k* o seu valor, que compreende o tempo de finalização da tarefa subtraído da data de término prometida. Caso a tarefa que ocupa a posição *k* finalize seu processamento antes da data de término prometida (ou seja, não esteja atrasada), o valor de T_k será igual a 0, pois, pela própria definição do *Tardiness*, $T_k \geq 0$, para todas as *k* posições possíveis.

3.2.2.3 **Segundo Grupo de Restrições – Formulação 5**

O segundo grupo de restrições é representado pelas expressões (31) e (32), da Formulação de Wilson (e, por conseqüência, é igual ao segundo grupo da Formulação 4). Fazem com que uma e somente uma posição *k* seja ocupada por uma e somente uma tarefa *j* e uma e somente uma tarefa *j* ocupe uma e somente uma posição *k*.

3.2.2.4 **Terceiro Grupo de Restrições – Formulação 5**

O terceiro grupo de restrições é representado pela expressão (33) da Formulação de Wilson (e, por conseqüência, é igual ao terceiro grupo da Formulação 4). Atribui à variável P_i^k o tempo de processamento na máquina *i* da tarefa que ocupa a posição *k*.

3.2.2.5 **Quarto Grupo de Restrições – Formulação 5**

$$d_j^k = d_j . x_j^k \quad \forall \quad j = 1, \dots, n; \quad \forall \quad k = 1, \dots, n \qquad (49)$$

O quarto grupo de restrições associa à variável d_j^k a data de término prometida da tarefa *j* que ocupa a posição *k*. Esta variável, ao ser diminuída do tempo de finalização C_i^k, da origem ao *Tardiness* da tarefa *j*.

3.2.2.6 **Quinto Grupo de Restrições – Formulação 5**

O quinto grupo de restrições é representado pela expressão (34) (e, por conseqüência, é idêntico ao quarto grupo da Formulação 4). Encarrega-se de associar o tempo de conclusão de uma dada tarefa que ocupa a posição *k* numa máquina *i*, como sendo igual ao tempo de início na máquina *i*, somado ao tempo de processamento desta tarefa na máquina *i*.

3.2.2.7 **Sexto Grupo de Restrições – Formulação 5**

O sexto grupo de restrições, igualmente ao quinto grupo de restrições da Formulação 4, é representado pela expressão (35) da Formulação de Wilson. Faz com que, para uma mesma máquina *i*, uma determinada tarefa que ocupa a posição *k* só possa ser iniciada quando a tarefa que ocupa a posição imediatamente anterior (*k* - 1) seja finalizada.

3.2.2.8 **Sétimo Grupo de Restrições – Formulação 5**

O sétimo grupo de restrições faz com que as tarefas sigam a seqüência do *flow shop* (processamento primeiro na máquina 1, em seguida na máquina subseqüente, e assim sucessivamente, até a máquina m). É representado pela expressão (36) da Formulação de Wilson, o que denota que este grupo é igual ao sexto grupo de restrições da Formulação 4.

3.2.2.9 **Restrições de Integralidade e Não Negatividade – Formulação 5**

As restrições de integralidade e não negatividade são representadas pelas expressões (38), (39), (40) e (41), da Formulação de Wilson. Somam-se a este grupo as restrições (50) (que fazem com que a data de término prometida de uma tarefa *j* que ocupa uma posição *k* seja sempre maior ou igual a zero) e (51) (a definição do *Tardiness* não admite que seu valor seja negativo). Estas duas últimas restrições podem ser observadas a seguir.

$$d_j^k \geq 0 \quad \forall \quad j = 1, \dots, n, \quad \forall \quad k = 1, \dots, n \qquad (50)$$

$$T_k \geq 0 \quad \forall \quad k = 1, \dots, n \qquad (51)$$

3.2.2.10 **Formulação 5 – Completa**

Função Objetivo:

$$Min \sum_{k=1}^{n} T_k \qquad (47)$$

Sujeito a:

$$T_k \geq C_2^k - \sum_{j=1}^{n} d_j^k \quad \forall \quad k = 1, \dots, n \qquad (48)$$

$$\sum_{j=1}^{n} x_j^k = 1 \quad \forall \quad k = 1, \dots, n \qquad (31)$$

$$\sum_{k=1}^{n} x_j^k = 1 \quad \forall \quad j = 1, \dots, n \qquad (32)$$

$$P_i^k = \sum_{j=1}^{n} p_{ij} x_j^k \quad \forall \quad i = 1, \dots, m; \quad \forall \quad k = 1, \dots, n \qquad (33)$$

$$d_j^k = d_j . x_j^k \quad \forall \quad j = 1, \dots, n; \quad \forall \quad k = 1, \dots, n \qquad (49)$$

$$C_i^k = S_i^k + P_i^k \quad \forall \quad k = 1, \dots, n; \quad \forall \quad i = 1, \dots, m \qquad (34)$$

$$S_i^k \geq C_i^{k-1} \quad \forall \quad k = 2, \dots, n \quad \forall \quad i = 1, \dots, m \qquad (35)$$

$$S_i^k \geq C_{i-1}^k \quad \forall \quad k = 1, \dots, n \quad \forall \quad i = 1, .., m \qquad (36)$$

Restrições de Integralidade e Não Negatividade:

$$x_j^k \in \{0,1\}, \quad \forall \quad j = 1, \dots, n, \quad \forall \quad k = 1, \dots, n \qquad (38)$$

$$S_i^k \geq 0 \quad \forall \quad i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (39)$$

$$C_i^k \geq 0 \quad \forall \quad i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (40)$$

$$P_i^k \geq 0 \quad \forall \quad i = 1, \dots, m, \quad \forall \quad k = 1, \dots, n \qquad (41)$$

$$d_j^k \geq 0 \quad \forall \quad j = 1, \dots, n, \quad \forall \quad k = 1, \dots, n \qquad (50)$$

$$T_k \geq 0 \quad \forall \quad k = 1, \dots, n \qquad (51)$$

3.3 ANÁLISE DE DESEMPENHO DAS FORMULAÇÕES

Dado que a Formulação 5 só pode ser empregada para um caso especial do problema $F_2||\sum w_j T_j$, esta terá de ser analisada em separado das demais. Dessa forma, a análise de desempenho das formulações propostas neste trabalho será feita em duas partes. Inicialmente serão analisadas as Formulações 1, 2, 3 e 4, desenvolvidas para o caso geral do problema $F_2||\sum w_j T_j$. Após isto, será feita uma análise da Formulação 5.

O computador utilizado foi um *Intel Core 2 duo* de 2.13 GHz, com 2,00 GB de memória RAM. Para cada formulação, foi produzida uma "tabela de resultados" com oito campos, com os significados abaixo.

CAMPO	INFORMAÇÃO DO CAMPO
Instância	Informa a instância resolvida
Valor da Relaxação Linear	Informa o valor da função objetivo ao se fazer a relaxação linear do problema original
Valor da Relaxação Linear + Cortes	Informa o valor da função objetivo após a relaxação linear do problema e a aplicação do pré-processamento e dos cortes do CPLEX 11.0
Valor da função objetivo	Transmite o melhor valor da função objetivo até o limite de tempo estabelecido para execução do algoritmo
Valor da função objetivo x Relaxação linear	Registra a diferença percentual entre o valor da relaxação linear do problema original e o valor da função objetivo após o limite de tempo estabelecido.
Tempo (s)	Informa o tempo utilizado na resolução. Caso o tempo seja igual ao valor limite estabelecido, isto significa que o valor do campo "valor da função objetivo" não é provadamente ótimo
Número de nós	Informa o número de nós abertos na árvore de *branch and bound*
Distância máxima do valor ótimo	Para as instâncias onde não foi encontrada solução ótima informa-se, percentualmente, a distância máxima entre a melhor solução encontrada e a solução ótima. Caso o valor deste campo seja igual a 0,00%, isto significa que a solução do campo "valor da função objetivo" é ótima.

Tabela 4 - Campos das tabelas de resultados e seus significados

3.3.1 Análise das Formulações para o Caso Geral do Problema

As Tabelas 5, 6, 7 e 8, expõem os resultados obtidos ao serem utilizadas as Formulações 1, 2, 3 e 4, respectivamente. Para todas as instâncias utilizadas, determinou-se o limite máximo de 14.400 segundos (4 horas) para encontro da solução ótima.

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4A	26,00	26,00	26	0,00%	0,22	1	0,00%
4B	43,00	43,00	43	0,00%	0,02	1	0,00%
6A	57,00	58,00	58	1,72%	0,08	1	0,00%
6B	54,00	54,00	54	0,00%	0,09	1	0,00%
8A	71,75	77,00	77	6,82%	0,28	1	0,00%
8B	262,50	266,00	266	1,32%	1,19	1	0,00%
10A	99,57	102,00	102	2,38%	1,06	1	0,00%
10B	364,05	376,00	376	3,18%	21,58	1	0,00%
12A	137,02	144,00	144	4,85%	53,2	1	0,00%
12B	432,77	455,00	455	4,89%	178,33	113	0,00%
14A	190,55	204,00	204	6,59%	25,14	1	0,00%
14B	672,29	683,25	693	2,99%	455,98	79	0,00%
16A	232,84	239,42	252	7,60%	106,36	63	0,00%
16B	861,50	873,66	886	2,77%	515,56	79	0,00%
18A	259,77	265,36	274	5,19%	153,16	43	0,00%
18B	1.094,68	1.105,91	1.134	3,47%	4.955,83	1.062	0,00%
20A	478,25	486,38	493	2,99%	175,81	31	0,00%
20B	1.294,84	1.305,91	1.346	3,80%	8.921,28	4.313	0,00%
25A	673,93	684,96	688	2,04%	249,64	3	0,00%
25B	1.720,88	1.735,59	1.764	2,44%	13.684,49	835	0,00%
30A	946,61	953,76	968	2,21%	546,06	22	0,00%
30B	2.609,26	2.620,95	2.795	6,65%	14.400	26	5,62%
35A	1.221,64	1.230,68	1.278	4,41%	14.400	2.241	2,66%
35B	3.662,13	3.673,26	4.094	10,55%	14.400	3.625	9,84%
40A	1.430,40	1.438,34	1.479	3,29%	14.400	1.410	1,83%
40B	4.664,20	4.679,12	7.349	36,53%	14.400	7	36,14%
45A	2.093,20	2.102,54	2.302	9,07%	14.400	183	8,21%
45B	5.878,39	5.894,69	-	-	14.400	3	-
50A	2.492,77	2.502,04	2.817	11,51%	14.400	30	11,04%
50B	6.521,98	6.537,46	7.352	11,29%	14.400	3	10,91%

Tabela 5 – Resultados obtidos pela Formulação 1

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4A	22,54	26,00	26	13,31%	0,01	1	0,00%
4B	43,00	43,00	43	0,00%	0,02	1	0,00%
6A	50,19	58,00	58	13,47%	0,05	1	0,00%
6B	54,00	54,00	54	0,00%	0,06	1	0,00%
8A	66,50	77,00	77	13,64%	0,11	1	0,00%
8B	242,88	266,00	266	8,69%	0,39	1	0,00%
10A	87,60	102,00	102	14,12%	0,34	1	0,00%
10B	338,26	376,00	376	10,04%	8,02	1	0,00%
12A	119,20	144,00	144	17,23%	12,03	1	0,00%
12B	405,19	427,49	455	10,95%	74,27	462	0,00%
14A	177,74	204,00	204	12,87%	10,00	1	0,00%
14B	622,69	648,19	693	10,15%	264,08	2.712	0,00%
16A	219,85	237,02	252	12,76%	22,63	68	0,00%
16B	809,58	838,28	886	8,63%	358,61	2.780	0,00%
18A	243,07	265,72	274	11,29%	31,95	19	0,00%
18B	1.028,41	1.064,05	1.134	9,31%	3.355,73	30.708	0,00%
20A	442,48	483,93	493	10,25%	30,19	30	0,00%
20B	1.224,42	1.262,36	1.348	9,17%	14.400	118.101	0,37%
25A	630,93	672,18	688	8,29%	56,48	218	0,00%
25B	1.617,15	1.672,55	1.764	8,32%	14.400	92.018	0,34%
30A	895,53	930,13	968	7,49%	1.738,66	24.714	0,00%
30B	2.502,89	2.546,14	2.732	8,39%	14.400	17.763	4,21%
35A	1.173,29	1.200,38	1.264	7,18%	14.400	189.783	0,63%
35B	3.530,73	3.578,42	3.815	7,45%	14.400	6.225	4,38%
40A	1.396,32	1.416,95	1.459	4,30%	10.536,48	56.108	0,00%
40B	4.504,20	4.548,74	4.923	8,51%	14.400	3.987	5,53%
45A	2.058,12	2.082,32	2.149	4,23%	14.400	46.389	1,21%
45B	5.687,38	5.735,52	6.268	9,26%	14.400	1.491	7,18%
50A	2.452,06	2.468,27	2.582	5,03%	14.400	14.788	3,10%
50B	6.337,42	6.380,52	7.048	10,08%	14.400	783	8,61%

Tabela 6 – Resultados obtidos pela Formulação 2

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4A	26,00	26,00	26	0,00%	0,02	1	0,00%
4B	43,00	43,00	43	0,00%	0,05	1	0,00%
6A	55,00	58,00	58	5,17%	0,05	1	0,00%
6B	54,00	54,00	54	0,00%	0,09	1	0,00%
8A	70,64	77,00	77	8,26%	0,70	1	0,00%
8B	258,80	266,00	266	2,71%	0,39	1	0,00%
10A	98,25	102,00	102	3,68%	1,48	1	0,00%
10B	361,81	376,00	376	3,77%	13,36	1	0,00%
12A	135,80	144,00	144	5,69%	14,73	1	0,00%
12B	431,15	440,22	455	5,24%	76,17	292	0,00%
14A	190,33	204,00	204	6,70%	25,06	1	0,00%
14B	670,24	680,50	693	3,28%	139,17	397	0,00%
16A	232,60	239,13	252	7,70%	29,98	81	0,00%
16B	859,87	868,45	886	2,95%	203,09	444	0,00%
18A	258,44	267,89	274	5,68%	28,06	3	0,00%
18B	1.091,27	1.105,84	1.134	3,77%	1.007,55	4.400	0,00%
20A	475,62	486,09	493	3,53%	41,67	10	0,00%
20B	1.290,59	1.304,24	1.346	4,12%	3.572,88	19.329	0,00%
25A	671,43	680,18	688	2,41%	227,78	3.212	0,00%
25B	1.719,24	1.733,06	1.764	2,54%	3.691,86	6.240	0,00%
30A	940,33	951,66	968	2,86%	657,56	2.716	0,00%
30B	2.607,43	2.616,84	2.720	4,14%	14.400	6.722	2,35%
35A	1.219,14	1.226,77	1.263	3,47%	10.919,92	56.209	0,00%
35B	3.657,94	3.667,87	3.756	2,61%	14.400	3.744	1,44%
40A	1.429,45	1.437,02	1.459	2,03%	1.130,02	9.861	0,00%
40B	4.658,93	4.670,73	4.844	3,82%	14.400	2.620	2,46%
45A	2.091,99	2.103,28	2.149	2,65%	14.400	24.347	1,02%
45B	5.873,70	5.887,28	6.135	4,26%	14.400	564	3,21%
50A	2.490,81	2.501,02	2.548	2,24%	14.400	11.998	0,98%
50B	6.516,84	6.529,47	6.970	6,50%	14.400	490	5,54%

Tabela 7 – Resultados obtidos pela Formulação 3

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4A	4,35	26,00	26	83%	0,05	1	0,00%
4B	4,63	43,00	43	89%	0,08	1	0,00%
6A	0,00	15,60	58	100%	0,69	64	0,00%
6B	0,00	54,00	54	100%	0,33	1	0,00%
8A	0,00	8,28	77	100%	3,99	1.206	0,00%
8B	0,00	52,44	266	100%	3,52	909	0,00%
10A	0,00	8,87	102	100%	13,26	10.921	0,00%
10B	0,00	65,71	376	100%	13,27	6.384	0,00%
12A	0,00	1,51	144	100%	81,67	44.085	0,00%
12B	0,00	23,98	455	100%	951,25	791.582	0,00%
14A	0,00	24,00	204	100%	757,16	438.419	0,00%
14B	0,00	0,00	694	100%	14.400	4.311.393	7,06%
16A	0,00	12,06	252	100%	14.400	3.636.838	13,89%
16B	0,00	0,00	887	100%	14.400	835.098	36,64%
18A	0,00	0,00	278	100%	14.400	1.485.278	46,40%
18B	0,00	0,00	1.144	100%	14.400	759.087	59,35%
20A	0,00	0,00	496	100%	14.400	526.455	58,27%
20B	0,00	0,00	1.350	100%	14.400	423.235	69,85%
25A	0,00	3,96	714	100%	14.400	202.980	86,13%
25B	0,00	0,00	1.890	100%	14.400	221.415	90,16%
30A	0,00	0,00	1.136	100%	14.400	57.018	96,65%
30B	0,00	0,00	3.016	100%	14.400	66.213	98,04%
35A	0,00	0,00	1.654	100%	14.400	21.141	96,98%
35B	0,00	0,00	3.949	100%	14.400	194.508	95,67%
40A	0,00	0,00	2.570	100%	14.400	9.430	99,84%
40B	0,00	0,00	5.164	100%	14.400	88.902	98,01%
45A	0,00	0,00	3.225	100%	14.400	3.344	99,81%
45B	0,00	0,00	7.327	100%	14.400	35.550	99,26%
50A	0,00	0,00	4.370	100%	14.400	565	99,93%
50B	0,00	0,00	8.044	100%	14.400	12.186	100,00%

Tabela 8 – Resultados obtidos pela Formulação 4

A análise do desempenho das Formulações 1, 2, 3 e 4 considerará as três questões a seguir:

- Proximidade do valor da relaxação linear do problema inicial (limite dual inicial) em relação ao valor ótimo da função objetivo;
- Sucesso ou fracasso em encontrar a solução ótima dentro do limite de tempo máximo estabelecido. Para aquelas instâncias onde não foi encontrada a solução ótima, o desempenho será medido em função da distância máxima entre a melhor solução encontrada e a solução ótima; e
- Tempo empregado para encontrar a solução ótima.

No tocante a proximidade do valor da relaxação linear em relação ao valor ótimo, é possível notar nas tabelas de resultados que a Formulação 1, quando comparadas às demais formulações, obteve valores mais próximos ao valor ótimo em todas as instâncias. A única exceção foi o fato de que, nas instâncias 4B e 6B, os limites duais iniciais foram exatamente iguais ao obtido pelas Formulações 2 e 3.

A Tabela 9 compara as formulações, uma a uma, sob o aspecto dos limites iniciais. Isto foi feito subtraindo, para cada instância, o valor de suas relaxações lineares e, em seguida, dividindo pelo valor ótimo da função objetivo (ou melhor valor encontrado, considerando o emprego das quatro formulações). O valor apresentado é percentual, e significa o quanto a diferença entre as duas relaxações lineares comparadas representa no valor ótimo (ou melhor valor encontrado) da função objetivo.

Um percentual positivo indica que a formulação que vem na frente no título do campo gerou valor da relaxação linear mais próximo ao valor ótimo da função objetivo, na instância. Ao se registrarem percentuais negativos, isto traduz que a formulação que se apresenta em segundo na seqüência do título do campo é a formulação que gerou valor mais próximo ao valor ótimo.

A Tabela 9 é apresentada a seguir:

Instância	Form 1 x Form 2	Form 1 x Form 3	Form 1 x Form 4	Form 2 x Form 3	Form 2 x Form 4	Form 3 x Form 4
4A	13,31%	0,00%	83,25%	-13,31%	69,95%	83,25%
4B	0,00%	0,00%	89,23%	0,00%	89,23%	89,23%
6A	11,75%	3,45%	98,28%	-8,30%	86,53%	94,83%
6B	0,00%	0,00%	100,00%	0,00%	100,00%	100,00%
8A	6,82%	1,44%	93,18%	-5,38%	86,36%	91,74%
8B	7,38%	1,39%	98,68%	-5,99%	91,31%	97,29%
10A	11,74%	1,30%	97,62%	-10,44%	85,88%	96,32%
10B	6,86%	0,60%	96,82%	-6,26%	89,96%	96,23%
12A	12,38%	0,85%	95,15%	-11,53%	82,77%	94,31%
12B	6,06%	0,35%	95,11%	-5,71%	89,05%	94,76%
14A	6,28%	0,11%	93,41%	-6,17%	87,13%	93,30%
14B	7,16%	0,30%	97,01%	-6,86%	89,85%	96,72%
16A	5,16%	0,10%	92,40%	-5,06%	87,24%	92,30%
16B	5,86%	0,18%	97,23%	-5,68%	91,37%	97,05%
18A	6,10%	0,49%	94,81%	-5,61%	88,71%	94,32%
18B	5,84%	0,30%	96,53%	-5,54%	90,69%	96,23%
20A	7,25%	0,53%	97,01%	-6,72%	89,75%	96,47%
20B	5,23%	0,32%	96,20%	-4,92%	90,97%	95,88%
25A	6,25%	0,36%	97,96%	-5,89%	91,71%	97,59%
25B	5,88%	0,09%	97,56%	-5,79%	91,68%	97,46%
30A	5,28%	0,65%	97,79%	-4,63%	92,51%	97,14%
30B	3,91%	0,07%	95,93%	-3,84%	92,02%	95,86%
35A	3,83%	0,20%	96,73%	-3,63%	92,90%	96,53%
35B	3,50%	0,11%	97,50%	-3,39%	94,00%	97,39%
40A	2,34%	0,07%	98,04%	-2,27%	95,70%	97,97%
40B	3,30%	0,11%	96,29%	-3,19%	92,99%	96,18%
45A	1,63%	0,06%	97,40%	-1,58%	95,77%	97,35%
45B	3,11%	0,08%	95,82%	-3,04%	92,70%	95,74%
50A	1,60%	0,08%	97,83%	-1,52%	96,23%	97,76%
50B	2,65%	0,07%	93,57%	-2,57%	90,92%	93,50%

Tabela 9 – Comparação entre os limites duais iniciais

Que conclusões podem ser retiradas da Tabela 9? A primeira delas é que, como já adiantado acima, a Formulação 1 é a formulação que apresentou os limites mais apertados, quando comparada às demais. Tendo a Formulação 1 apresentado o melhor desempenho neste critério, qual das três formulações apresentou o segundo melhor desempenho? A Tabela 9

transmite que, dentre todas as formulações, a Formulação 4 obteve os piores limites iniciais. Neste sentido, coloca-se a Formulação 4 como sendo aquela com menor sucesso neste quesito e, sobram as Formulações 2 e 3 para avaliação.

A coluna “Form 2 x Form 3” esclarece a dúvida. A Formulação 3 gerou limites superiores quando comparada à Formulação 2. Notam-se valores nulos ou negativos em toda a coluna. Na avaliação sob o ponto de geração dos limites mais apertados, a Formulação 3 apresentou o segundo melhor desempenho, sendo a Formulação 2 a terceira colocada neste aspecto.

Tomado o segundo aspecto da análise: sucesso ou fracasso em encontrar a solução ótima dentro do limite de tempo máximo estabelecido (ou a distância máxima da solução ótima, para os casos onde não foi encontrada solução comprovadamente ótima dentro do limite de tempo), qual, dentre as formulações, apresentou o melhor desempenho?

O exame das tabelas de resultados das instâncias transmite que, ao utilizar a Formulação 1, nove instâncias não apresentaram solução ótima. Ao empregar a Formulação 2, dez instâncias não registraram solução ótima. Já o emprego da Formulação 3 fez com que não se chegasse à solução ótima em sete casos. Por último, ao empregar a Formulação 4, em dezenove instâncias não foram obtidos valores comprovadamente ótimos. A Tabela 10 sintetiza estes resultados:

Formulação	Nº de instâncias aonde não se chegou ao valor ótimo em 14.400s
1	9
2	10
3	7
4	19

Tabela 10 – Nº de instâncias onde não se encontrou a solução ótima

Exceto a Formulação 4, as demais formulações apresentaram números próximos de instâncias onde não foi possível encontrar a solução ótima dentro do limite de tempo, sendo a Formulação 3 um pouco superior em relação as demais. Contudo, a análise da distância máxima entre o valor ótimo e a solução encontrada ajuda a solidificar a conclusão de que, no tocante ao sucesso ou fracasso em encontrar a solução ótima dentro do limite de tempo máximo estabelecido, a Formulação 3 apresentou o melhor desempenho. Quando são

analisadas estas distâncias, nota-se que a Formulação 3 apresentou as menores diferenças. Ou seja, para aquelas formulações onde não se encontrou a solução ótima, pelo menos a solução encontrada ficou mais próxima da solução comprovadamente ótima. Julgar qual formulação apresentou o segundo melhor desempenho dentre as Formulações 1 e 2 é difícil pois, apesar de a Formulação 1 apresentar menor número de instâncias onde não se encontrou o valor ótimo, as distâncias máximas em relação ao valor ótimo verificadas foram superiores quando comparadas à Formulação 2.

Há que se notar também que, independente da formulação empregada, a dificuldade em encontrar a solução ótima concentra-se na parte inferior das tabelas de resultados. Isto é justificável na medida em que, quanto maior o número de tarefas da instância, maior é o número de variáveis e restrições empregadas.

No tocante ao último ponto de comparação, tempo empregado para encontro da solução ótima, as análises serão feitas tomando por base as instâncias da série A e da série B, separadamente. Ao se realizar esta divisão, torna-se possível elaborar oito gráficos (dois para cada formulação) contendo o número de tarefas no eixo horizontal e o tempo utilizado para encontrar a solução ótima no eixo vertical.

Os gráficos são apresentados logo a seguir. De forma a possibilitar a inclusão de amostras com diferentes ordens de grandeza no gráfico, foi utilizada a escala logarítmica. Os marcadores retangulares informam que a solução encontrada é ótima. Já os marcadores triangulares informam que a solução não é comprovadamente ótima. Exceção se faz a Formulação 1 quando utilizada a instância de 45 tarefas, da série B. Neste caso, nenhuma solução viável foi encontrada ao final dos 14.400 segundos.

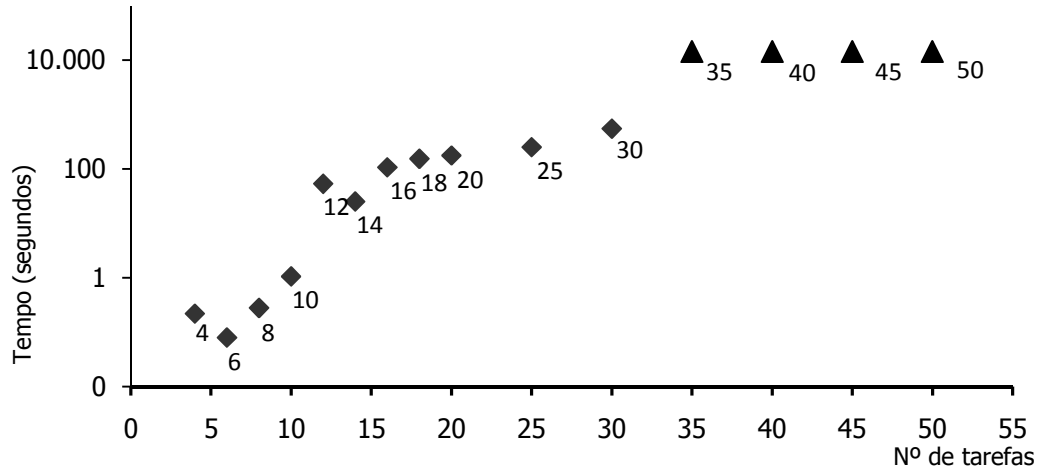

Figura 15 - Tempo x Nº de Tarefas: Formulação 1 – Série A

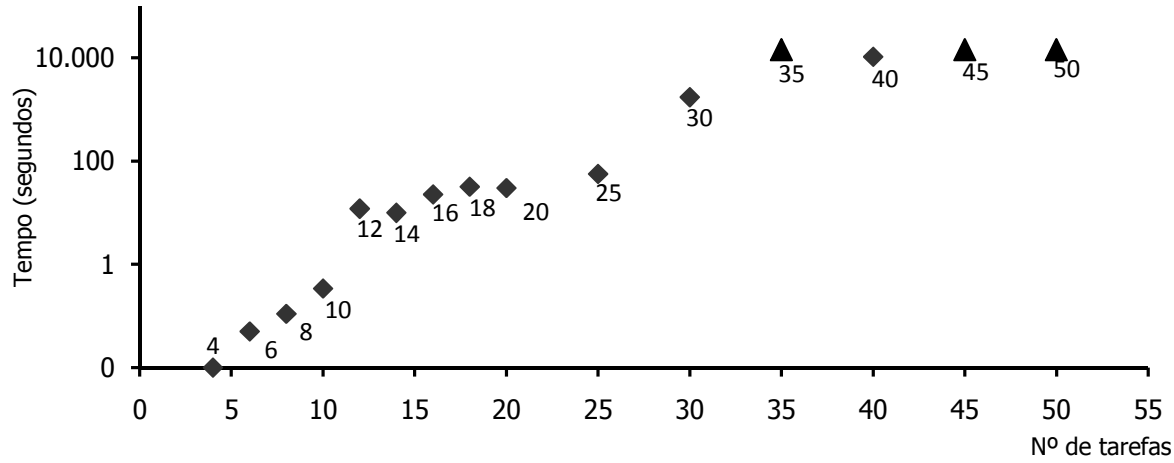

Figura 16 - Tempo x Nº de Tarefas: Formulação 2 – Série A

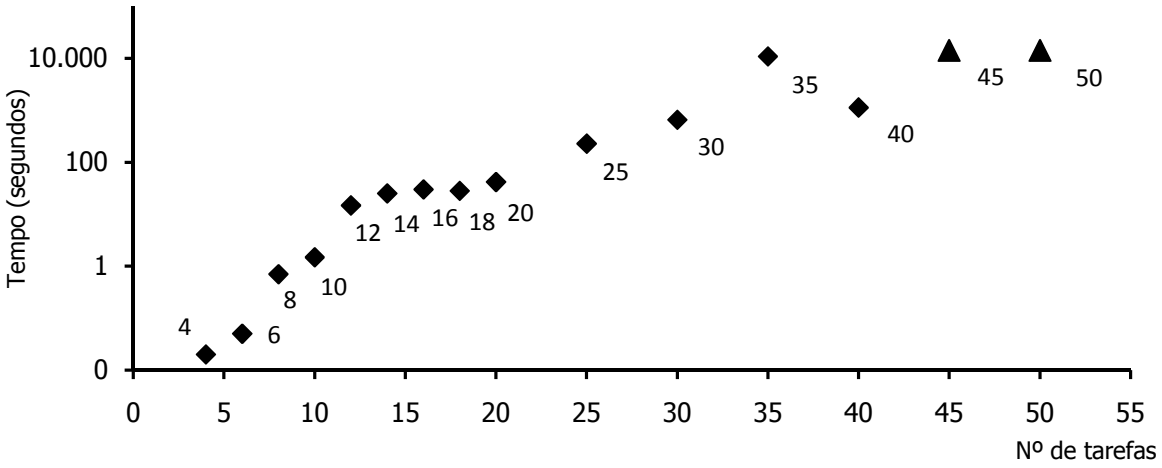

Figura 17 - Tempo x Nº de Tarefas: Formulação 3 – Série A

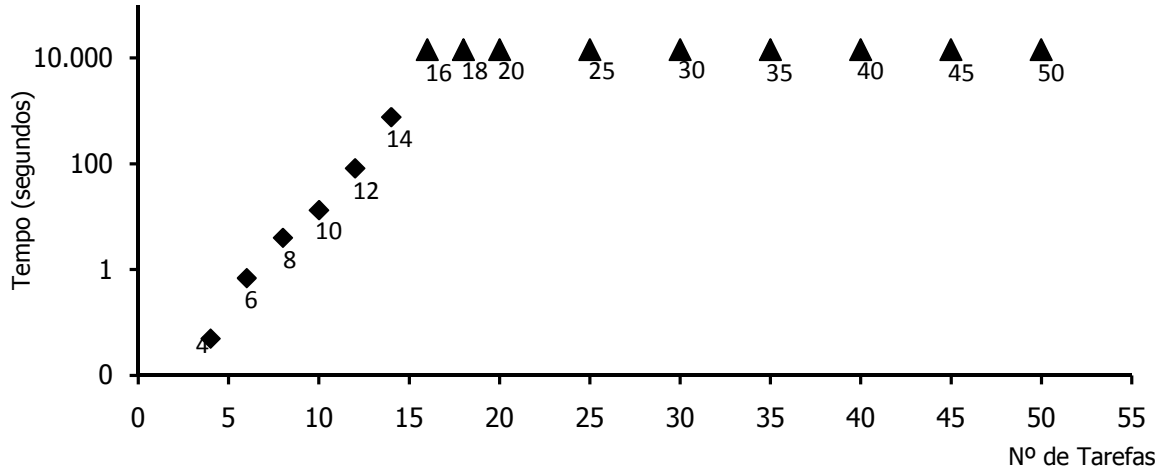

Figura 18 - Tempo x Nº de Tarefas: Formulação 4 – Série A

Analisando os gráficos acima, nota-se que as Formulações 2 e 3 apresentaram, cada uma, 6 instâncias onde o tempo de execução foi superior a 100 segundos. A Formulação 4 foi a que apresentou maior número de instâncias que necessitaram mais de 100 segundos, sendo que a Formulação 1 apresentou número um pouco superior ao das Formulações 2 e 3.

Apesar do fato de que as Formulações 2 e 3 apresentarem mesmo número de instâncias com tempo de execução superior a 100 segundos, a Formulação 3 foi superior ao lidar com as instâncias de número de tarefas igual ou superior a trinta e cinco. Empregando a Formulação 3, foi possível encontrar a solução ótima da instância de trinta e cinco tarefas, encontrar de maneira mais rápida a solução ótima da instância de quarenta tarefas (além de encontrar soluções mais próximas do valor ótimo da função objetivo nas instâncias de 45 e 50 tarefas, fato abordado anteriormente).

Serão analisadas, agora, as instâncias da série B. As figuras 19, 20, 21 e 22, ilustram os resultados das Formulações 1, 2, 3 e 4 respectivamente, nestas instâncias:

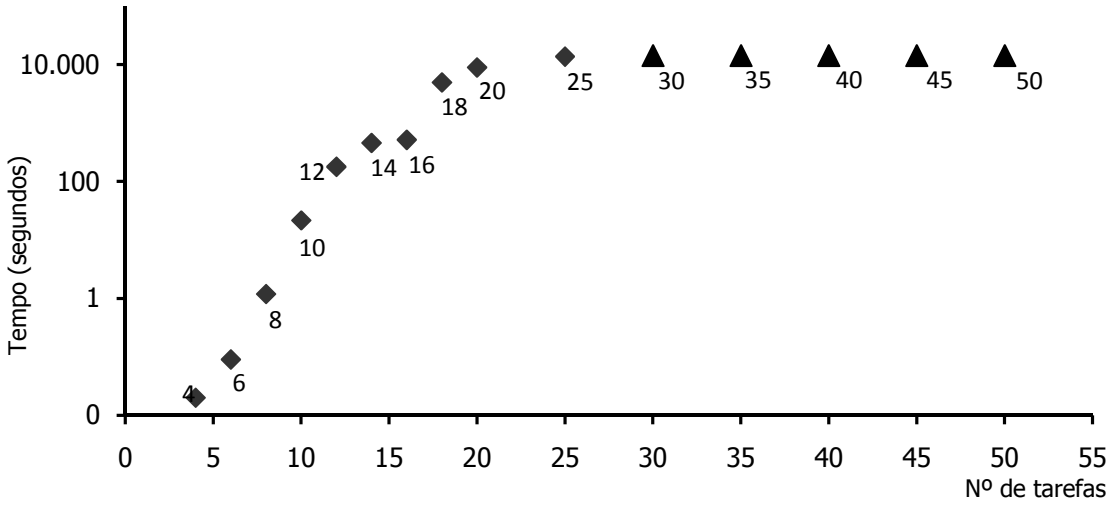

Figura 19 - Tempo x Nº de Tarefas: Formulação 1 – Série B

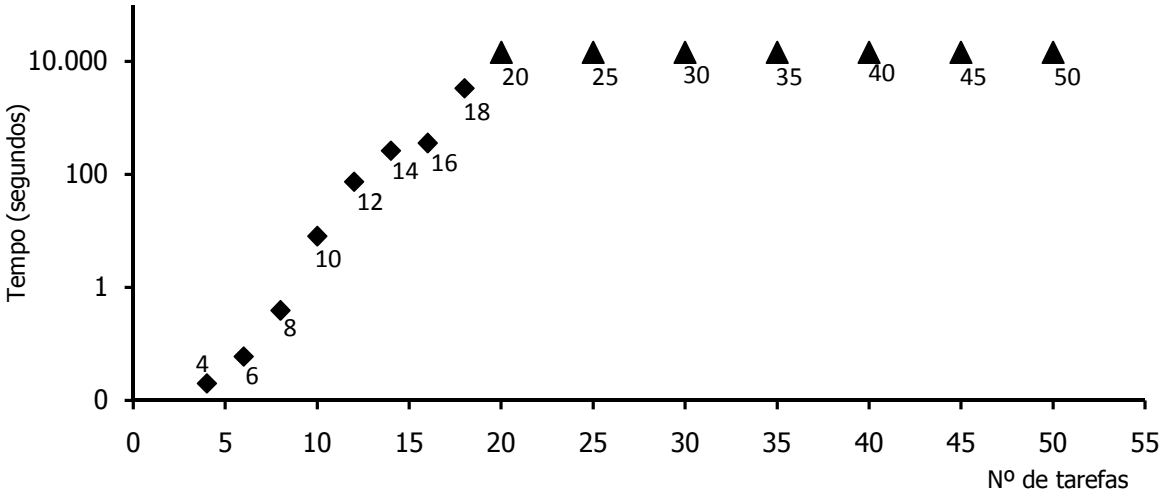

Figura 20 - Tempo x Nº de Tarefas: Formulação 2 – Série B

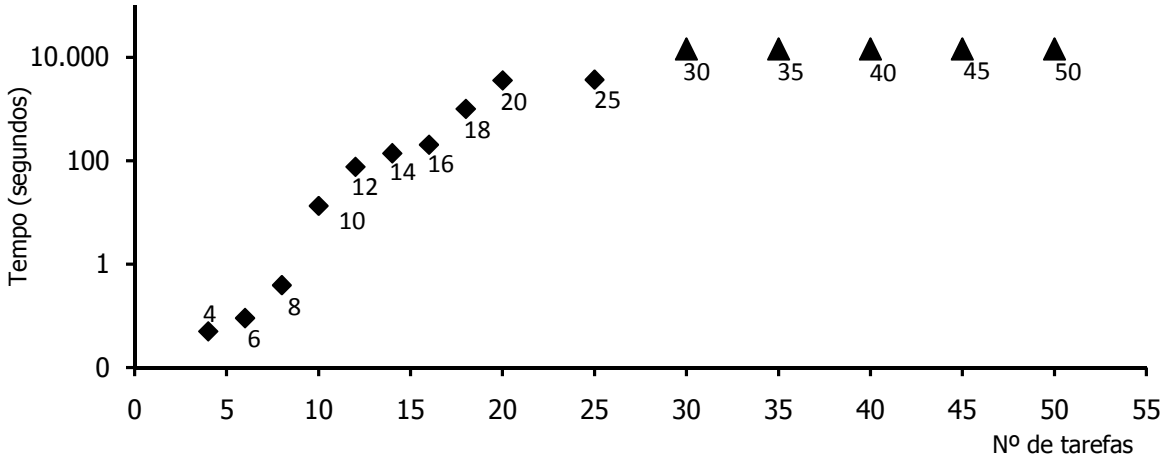

Figura 21 - Tempo x Nº de Tarefas: Formulação 3 – Série B

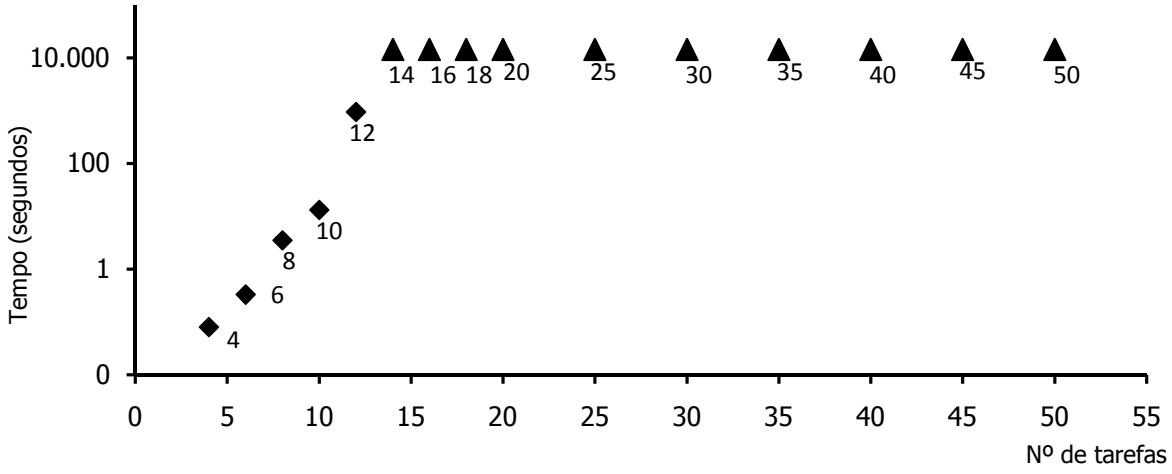

Figura 22 - Tempo x Nº de Tarefas: Formulação 4 – Série B

Analisando os gráficos acima, nota-se que a Formulação 4 apresentou o pior desempenho. A Formulação 3 foi aquela onde se conseguiu solucionar o maior número de instâncias com vinte tarefas ou mais. A Formulação 1 conseguiu resolver instâncias com maior número de tarefas do que a formulação 2 (fato que não ocorreu quando foram utilizadas as instâncias da série A).

É possível sintetizar os critérios analisados e eleger, dentre as quatro formulações, a melhor? Sem dúvida, trata-se de uma questão de qual critério a ser adotado. Se levado em consideração a questão de quão apertada é a formulação, a Formulação 1 seria eleita como melhor. Se tomado o critério de velocidade e capacidade de resolver as instâncias propostas, a formulação que mais se destacaria seria a Formulação 3.

O que pode ser inferido é o fato de que, sem dúvida, as Formulações 1 e 3 se destacam em relação às Formulações 2 e 4.

3.3.2 Análise das Formulações para um Caso Particular do Problema

Para a realização de testes, serão utilizadas instâncias iguais às empregadas anteriormente, exceto pelo fato de apresentarem $w_j = 1, \ \forall\, j = 1, \ldots, n$. Foi utilizado um limite de tempo igual a 3.600 segundos (1 hora). Os resultados obtidos pela Formulação 5 podem ser observados na tabela a seguir.

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4A	10,67	11,00	11	3,03%	0,22	1	0,00%
4B	11,00	11,00	11	0,00%	0	1	0,00%
6A	21,45	23,00	23	6,72%	0,06	1	0,00%
6B	16,86	19,00	19	11,25%	0,02	1	0,00%
8A	28,44	32,00	32	11,11%	0,17	1	0,00%
8B	79,48	81,00	81	1,88%	0,05	1	0,00%
10A	42,55	47,30	49	13,15%	0,48	12	0,00%
10B	108,26	116,19	122	11,26%	0,41	12	0,00%
12A	60,27	64,57	69	12,66%	1,31	45	0,00%
12B	131,77	138,76	152	13,31%	0,83	211	0,00%
14A	97,70	101,20	107	8,69%	1,34	72	0,00%
14B	226,23	233,05	247	8,41%	1,47	215	0,00%
16A	137,70	142,17	148	6,96%	2,81	285	0,00%
16B	277,05	283,08	320	13,42%	5,03	809	0,00%
18A	149,80	154,97	170	11,88%	14,95	2.329	0,00%
18B	335,85	344,98	393	14,54%	10,89	1.864	0,00%
20A	263,60	268,56	284	7,18%	15,88	3.367	0,00%
20B	386,12	394,27	455	15,14%	53,56	20.183	0,00%
25A	388,53	394,92	419	7,27%	90,38	31.114	0,00%
25B	581,22	592,16	678	14,27%	1045,27	381.195	0,00%
30A	570,80	577,49	605	5,65%	534,81	202.991	0,00%
30B	896,49	902,70	997	10,08%	3.600	623.491	1,62%
35A	707,33	715,47	784	9,78%	3.600	639.714	0,26%
35B	1.286,05	1.293,47	1.386	7,21%	3.600	372.135	1,80%
40A	817,51	825,85	934	12,47%	3.600	220.047	2,25%
40B	1.650,81	1.655,39	1.782	7,36%	3.600	183.505	3,42%
45A	1.198,26	1.207,55	1.315	8,88%	3.600	203.127	2,13%
45B	2.031,94	2.037,33	2.253	9,81%	3.600	147.144	3,15%
50A	1.400,57	1.412,48	1.576	11,13%	3.600	96.575	4,82%
50B	2.150,63	2.155,49	2.571	16,35%	3.600	103.775	9,18%

Tabela 11 - Resultados obtidos pela Formulação 5

O desempenho da Formulação 5 será comparado ao da Formulação 3 (formulação que apresentou o melhor desempenho em termos de velocidade e capacidade de resolver instâncias com maior número de tarefas). Neste sentido, conforme procedido para a Formulação 5, utilizando as instâncias da série A e B (com valores unitários para todas as tarefas), foram obtidos os seguintes resultados ao utilizar a Formulação 3.

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4A	11,00	11,00	11	0,00%	0	1	0,00%
4B	11,00	11,00	11	0,00%	0,03	1	0,00%
6A	23,00	23,00	23	0,00%	0,02	1	0,00%
6B	19,00	19,00	19	0,00%	0,06	1	0,00%
8A	31,70	32,00	32	0,94%	0,09	1	0,00%
8B	80,01	81,00	81	1,22%	0,73	1	0,00%
10A	48,70	49,00	49	0,62%	0,2	1	0,00%
10B	118,68	122,00	122	2,72%	10,09	1	0,00%
12A	67,50	69,00	69	2,17%	3,83	1	0,00%
12B	145,12	152,00	152	4,53%	39,63	1	0,00%
14A	105,35	107,00	107	1,55%	2,02	1	0,00%
14B	238,72	240,57	247	3,35%	89,95	130	0,00%
16A	145,55	146,61	148	1,65%	21,5	14	0,00%
16B	310,43	320,00	320	2,99%	126,2	59	0,00%
18A	164,98	166,83	170	2,96%	30,84	101	0,00%
18B	381,38	383,79	393	2,96%	1049	2.782	0,00%
20A	278,40	279,60	284	1,97%	43,25	88	0,00%
20B	440,78	443,14	455	3,13%	1674,56	10.445	0,00%
25A	411,54	412,74	419	1,78%	426,05	3.877	0,00%
25B	656,98	659,13	683	3,81%	3.600	3.335	1,95%
30A	593,60	595,77	608	2,37%	3.600	10.194	0,82%
30B	969,91	972,49	1.005	3,49%	3.600	1.048	2,09%
35A	771,88	773,96	786	1,80%	3.600	7.683	0,64%
35B	1.361,56	1.363,62	1.450	6,10%	3.600	101	5,45%
40A	922,09	924,31	969	4,84%	3.600	5.056	4,13%
40B	1.746,91	1.748,61	1.859	6,03%	3.600	22	5,81%
45A	1.297,64	1.299,59	1.343	3,38%	3.600	3.308	2,83%
45B	2.214,04	2.214,93	2.382	7,05%	3.600	9	6,97%
50A	1.453,24	1.545,82	1.581	8,08%	3.600	1.794	1,96%
50B	2.442,74	2.444,33	2.560	4,58%	3.600	13	4,41%

Tabela 12 - Resultados obtidos pela Formulação 3 quando utilizadas em instâncias do problema do somatório do *Tardiness*

Com base nas duas tabelas acima, nota-se que a formulação com variáveis que contém índices de posição consegue, de modo geral, solucionar de maneira mais rápida as instâncias propostas. Isto acontece mesmo tendo a Formulação 3 obtido limites duais inicias superiores em todas as instâncias.

Uma questão que auxilia a formulação que emprega variáveis com índice de posição é o fato de que esta formulação utiliza um número menor de variáveis, quando comparada as formulações com variáveis contendo índice de tempo.

Outro fato que pode ser notado é que a formulação que emprega variáveis com índice de tempo apresenta seu desempenho bastante associado aos tempos de processamento das tarefas. Caso as instâncias a serem resolvidas apresentem tarefas com tempos de processamento entre limites, por exemplo, variando entre 1 e 100, o número de variáveis de decisão eleva-se consideravelmente..

Ilustrando com instâncias de 4, 6, 8 e 10 tarefas, sendo os tempos de processamento gerados de forma aleatória com valores entre 1 e 100 e pesos unitários (estas instâncias constam na Tabela 18, no Apêndice), têm-se os seguintes desempenhos das Formulações 5 e 3 quando aplicadas na minimização do somatório do *Tardiness*.

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4M	111,11	128,00	128	13,20%	0	1	0,00%
6M	286,43	326,00	326	12,14%	0,08	1	0,00%
8M	459,11	522,00	522	12,05%	0,22	1	0,00%
10M	837,26	883,99	907	7,69%	0,34	17	0,00%

Tabela 13 - Resultados da Formulação 5 quando aplicado em instâncias com tempos de processamento entre 1 e 100

Instância	Valor da Relaxação Linear	Valor da Relaxação Linear + Cortes	Valor da função objetivo	Valor da função objetivo x Relaxação linear	Tempo (s)	Número de nós	Distância máxima do valor ótimo
4M	121,14	128,00	128	5,36%	5,33	1	0,00%
6M	298,57	326,00	326	8,41%	151,83	1	0,00%
8M	510,11	522,00	522	2,28%	31,69	1	0,00%
10M	894,64	907,00	907	1,36%	78,11	1	0,00%

Tabela 14 - Resultados da Formulação 3 quando aplicado em instâncias com tempos de processamento entre 1 e 100

Nota-se que a formulação que emprega variáveis com índice de tempo incorre num período bastante superior, quando comparada a formulação com variáveis com índice de posição.

3.4 HEURÍSTICAS PARA BUSCA DE LIMITES PRIMAIS INICIAIS

Por último, o que poderia ser feito para apoiar o algoritmo de *branch and cut* a encontrar uma solução, pelo menos próxima da ótima, para a instância 45B na Formulação 1? Para este caso, não se encontrou sequer uma solução viável, mesmo após os 14.400 segundos de execução.

Conforme exposto na revisão bibliográfica, uma das aplicações das heurísticas está em passar limites primais iniciais para o algoritmo de *branch and bound* ou *branch and cut*. Quanto mais próximo este limite for do valor ótimo da função objetivo, melhor será a contribuição fornecida pela heurística.

Neste sentido serão apresentadas a seguir algumas heurísticas, desenvolvidas para o problema desta pesquisa. O principal objetivo é observar e comparar seus resultados, além de utilizar uma destas na Formulação 1 com a instância 45B, no papel de um limite primal inicial.

As heurísticas abaixo estão descritas pela regra com que a seqüência de tarefas é elaborada.

Heurística 1: Ordena as tarefas segundo o maior tempo de processamento p_{2j}.

Heurística 2: Ordena as tarefas segundo o maior tempo de processamento p_{1j}.

Heurística 3: Ordena as tarefas em ordem crescentes dos d_j.

Heurística 4: Ordena as tarefas utilizando três critérios, em ordem de prioridade: 1º maior d_j, 2º maior w_j, 3º maior tempo de processamento p_{1j}.

Heurística 5: Ordena as tarefas utilizando três critérios, em ordem de prioridade: 1º maior w_j, 2º menor d_j, 3º menor $p_{1j} + p_{2j}$.

Heurística 6: Ordena as tarefas utilizando três critérios, em ordem de prioridade: 1º menor quociente: d_j / w_j, 2º menor p_{1j}, 3º menor p_{2j}.

Heurística 7: Ordena as tarefas utilizando três critérios, em ordem de prioridade: 1º menor quociente: d_j / w_j, 2º menor p_{2j}, 3º menor p_{1j}.

Após a montagem destas seqüências, foi aplicado o procedimento de 1-troca. Este procedimento consistiu em alterar o seqüenciamento elaborado pela regra inicial de forma que, uma dada tarefa mude sua posição no seqüenciamento com a tarefa seguinte. Tal procedimento, para uma dada tarefa *j*, ocorre até esta situar-se em todas as posições possíveis do seqüenciamento, sendo calculado o custo para cada posição. Isto foi feito para todas as *n* tarefas, em todas as heurísticas e instâncias possíveis.

Os resultados são expostos a seguir. Os valores correspondem ao melhor valor de cada heurística, após a aplicação do procedimento de 1-troca. Tomada a mesma instância, a heurística que forneceu o melhor limite segue grafada em negrito. É possível comparar também a heurística que forneceu o melhor valor com o valor ótimo ou melhor valor encontrado em todas as formulações.

Instância	Heurística 1	Heurística 2	Heurística 3	Heurística 4	Heurística 5	Heurística 6	Heurística 7	Valor ótimo ou melhor valor encontrado	Distância mínima %
4A	38	**26**	**26**	**26**	**26**	**26**	**26**	*26*	0,00%
4B	56	**43**	**43**	56	56	56	56	*43*	0,00%
6A	88	**58**	**58**	77	**58**	**58**	**58**	*58*	0,00%
6B	134	154	117	**76**	113	92	92	*54*	40,74%
8A	113	**83**	102	138	107	**83**	**83**	*77*	7,79%
8B	457	461	365	370	468	**352**	**352**	*266*	32,33%
10A	218	113	165	205	134	**110**	**110**	*102*	7,84%
10B	718	627	515	550	611	**471**	**471**	*376*	25,27%
12A	319	230	259	315	204	**160**	**160**	*144*	11,11%
12B	857	743	588	649	701	**551**	**551**	*455*	21,10%
14A	457	443	369	518	361	**224**	**224**	*204*	9,80%
14B	1.321	1.236	996	987	1.080	**799**	**799**	*693*	15,30%
16A	611	557	498	689	546	**283**	**283**	*252*	12,30%
16B	1.724	1.587	1.265	1.367	1.396	**1.040**	**1.040**	*886*	17,38%
18A	719	607	621	744	608	**360**	**360**	*274*	31,39%
18B	2.148	1.964	1.650	1.659	1.896	**1.437**	**1.437**	*1.134*	26,72%
20A	1.062	855	1.011	1.049	847	**625**	**625**	*493*	26,77%
20B	2.471	2.314	2.066	1.957	2.171	**1.784**	1.840	*1.346*	32,54%
25A	1.717	1.112	1.487	1.601	1.153	**837**	**837**	*688*	21,66%
25B	3.556	3.266	3.297	3.130	3.007	**2.483**	2.555	*1.764*	40,76%
30A	2.479	1.589	2.034	2.329	1.727	**1.225**	**1.225**	*968*	26,55%
30B	5.209	5.011	5.092	4.587	4.201	**4.148**	4.167	*2.720*	52,50%
35A	3.407	2.371	2.880	3.081	2.424	**1.592**	1.592	*1.263*	26,05%
35B	7.108	7.053	6.644	6.813	5.803	**5.558**	5.577	*3.756*	47,98%
40A	3.912	3.246	3.641	3.857	2.993	2.022	2.016	*1.459*	38,18%
40B	10.220	9.232	8.824	8.947	7.367	**7.231**	7.250	*4.844*	49,28%
45A	5.381	4.506	5.337	5.442	4.096	3.310	**3.303**	*2.149*	53,70%
45B	13.303	12.102	11.084	11.467	**9.074**	9.435	9.454	*6.135*	47,91%
50A	7.023	6.515	6.257	6.911	4.722	4.382	**4.375**	*2.548*	71,70%
50B	15.043	15.960	20.197	12.978	**10.348**	10.745	10.764	*6.970*	48,46%

Tabela 15 - Resultado obtido pelas heurísticas propostas

Notam-se que, em alguns casos, as heurísticas fornecem valores próximos, se não ótimos (para o caso de instâncias com reduzido número de tarefas). Contudo, na maioria das

instâncias, o resultado da heurística ficou consideravelmente distante da solução ótima, ou melhor solução.

Retornando a instância não solucionada pela Formulação 1 (instância 45B). Com a passagem de uma solução inicial para o algoritmo de *branch and cut*, com custo igual a 9.074, foi possível chegar a uma solução viável com custo de 6.474 (após 14.400 segundos), estando esta, no máximo, 8,92% distante do valor ótimo da função objetivo.

4 CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi propor e analisar formulações para o problema de escalonamento determinístico *flow shop* em duas máquinas com penalidades por atraso nas tarefas. Foram apresentadas quatro formulações gerais para o problema, além de uma quinta formulação para um caso especial do problema, quando são atribuídos pesos unitários a todas as tarefas. Após serem apresentadas, as formulações foram analisadas com base em critérios de desempenho tais como: valor do limite dual inicial (gerado pela relaxação linear do problema original), sucesso ou fracasso em encontrar a solução ótima e o tempo empregado para encontrar a solução ótima. Duas formulações destacaram-se das demais. Uma delas (Formulação 1) gerou valores de limites duais iniciais mais próximos do valor ótimo em todas as instâncias analisadas. Outra, Formulação 3, foi superior ao solucionar instâncias com maior número de tarefas, dentro do tempo limite estabelecido.

Uma questão que chamou a atenção foi a relação entre o número de variáveis empregadas na formulação e o tempo necessário para resolver determinadas instâncias. Notou-se que, para uma mesma instância, as formulações com menor número de variáveis conseguiram chegar à solução ótima num intervalo de tempo inferior. Particularmente, nos testes com instâncias onde as tarefas possuíam tempo de processamento entre 1 e 100, esta questão ficou bastante visível.

Alguns pontos podem dar origem a estudos futuros. O primeiro deles é a proposição de heurísticas que forneçam respostas mais próximas do valor ótimo, como forma de passar um limite primal inicial mais justo ao algoritmo de *branch and bound* ou *branch and cut*. Conforme visto, para uma dada formulação e instância, só foi possível conhecer uma solução, próxima da solução ótima, com a passagem de uma solução viável inicial ao algoritmo. O fornecimento de um limite inicial com qualidade superior poderia talvez fazer com que a solução ótima fosse encontrada, ou que se encontrasse uma solução ainda mais próxima à

ótima, neste e nos outros problemas onde não foi obtida solução ótima comprovada. Um segundo tema que pode ser objeto de estudos futuros é a adição de planos de cortes para este problema. Na fase de *presolve* do *software* empregado para solucionar as instâncias (CPLEX 11.0), notou-se o intenso emprego de planos de cortes. Ao estudar as particularidades do problema $F_2||\sum w_j T_j$, talvez um pesquisador consiga propor cortes mais fortes à formulação, o que de fato refletiria diretamente no tempo empregado para solução.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, E; TEIXEIRA, J. *Apresentação de trabalhos monográficos de conclusão de curso*. Niterói: Eduff, 2007.

ARENALES, M. et al. *Pesquisa Operacional*. Rio de Janeiro: Elsevier, 2007.

AUBIN, J. Scheduling Ambulances. *Interfaces* 22, 1992, pp. 1-10.

BUTCHERS, E. R. et al. Optimized Crew Scheduling at Air New Zealand. *Interfaces* 31, 2001, pp. 30-56.

BOOTH, W. C.; COLOMB, G. G.; WILLIAMS, J. M. *A arte da pesquisa*. São Paulo: Martins Fontes, 2005.

COLIN, E. *Pesquisa Operacional*. Rio de Janeiro: LTC, 2007.

CORMEN, T. et al. *Algoritmos*: Teoria e Prática. Rio de Janeiro: Campus, 2002.

DANTZIG, G. B. *Linear Programming and Extensions*. Nova Jersey: Prineton University Press, 1963.

DYER, M; WOLSEY, L. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*. 26, 1990, pp. 255-270.

GAREY, M. R; JOHNSON, D. S. *Computers and Intractability*. Nova York: Freeman, 1979.

GOLDBARG, M; LUNA, H. *Otimização Combinatória e Programação Linear*: Modelos e Algoritmos. Rio de Janeiro: Elsevier, 2000.

HILLIER, F; LIEBERMAN, J. *Introdução à Pesquisa Operacional*. São Paulo: McGraw-Hill, 2006.

JOHNSON, S. M. Optimal two- and three-stage Production Schedules with set-up times included, *Naval Research Logistics Quartely*, 1, 1954, pp. 61-68.

KALLRATH, J; WILSON, J. M. *Business Optimisation Using Mathematical Programming*. Londres: Macmillian, 1997.

LACHTERMACHER, G. *Pesquisa Operacional na Tomada de Decisões*. Rio de Janeiro: Elsevier, 2006.

LAGEWEG, B. J. Computer-aided complexity classification of deterministic scheduling problems, *Report BW* 138, Mathematisch Centrum, Amsterdam, 1981.

LAND, A. H.; DOIG, A. G. An automatic method for solving discrete programming problems. *Econometrica* 28, 1960, pp. 497-520.

LAWLER, E. L; LENSTRA, J. K; RINNOOY KAN, A. H. G. Recent Developments in Deterministic Sequencing and Scheduling: A Survey. *Deterministic and Stochastic Scheduling*, 1982, pp. 35-74.

LENSTRA, J. K; RINNOOY KAN, A. H. G; BRUCKER, P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, Vol. 1, 1977, pp. 343-362.

MARCONI, M; LAKATOS, E. *Fundamentos de Metodologia Científica*. São Paulo: Atlas, 2007.

NICHOLSON, T. A. J. *Optimization in Industry*: Optimization Techniques. London: Longman Press, 1970.

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

POTTS, C. N. An adaptive branching rule for the permutation flow-shop problem, *European Journal of Operational Research*, 5, 1980, pp. 19-25.

PUCCINI, A. L; PIZZOLATO, N. D. *Programação Linear*. 2. ed. Rio de Janeiro: LTC, 1989.

PORTOUGAL, V. Production Scheduling in the Snack-Food Industry. *Interfaces* 27, 1997, pp. 51-64.

RUIZ, R.; MAROTO, C. A comprehensive review and evaluation of permutation flowshop heuristics, *European Journal of Operational Research* 165, 2005.

TAHA, H. *Pesquisa Operacional*. São Paulo: Prentice Hall, 2008.

WAGNER, H. M. An integer linear programming model for machine scheduling. *Nav. Res. Log. Quart*., 6, 1959, pp. 131-140.

WILSON, J. M. Alternative formulations of a flow-shop scheduling problem, *Journal of the Operational Research Society*, 40, 1989, pp. 395-399.

WOLSEY, L. *Integer Programming*. New York: Wiley, 1998.

WRIGHT, M. H. The interior points revolution in optimization: history, recent developments, and last consequences. *Bulletin of the American Mathematical Society*. 42, 2004, pp. 39-56.

APÊNDICE

J	p_{1j}	p_{2j}	w_j	d_j
1	5	4	4	9
2	5	1	3	10
3	2	3	5	5
4	3	4	1	10
5	1	5	4	8
6	4	5	1	20
7	3	4	4	24
8	1	2	3	12
9	5	5	3	24
10	4	2	1	27
11	3	5	3	32
12	3	2	5	21
13	3	5	5	31
14	4	5	1	31
15	5	4	1	19
16	5	5	5	57
17	1	1	2	35
18	3	1	2	30
19	5	4	3	12
20	5	3	2	16
21	3	1	5	47
22	5	3	1	62
23	5	1	2	29
24	4	5	1	76
25	2	1	1	21

Tabela 16 - Informações das tarefas para as instâncias da série A.

j	p_{1j}	p_{2j}	w_j	d_j
26	2	5	2	24
27	2	1	2	14
28	2	5	1	43
29	5	2	1	26
30	5	4	5	103
31	2	5	5	81
32	3	5	2	14
33	5	3	4	68
34	1	4	1	31
35	1	2	5	109
36	1	4	2	14
37	3	5	3	122
38	1	4	1	79
39	2	1	3	90
40	1	4	4	77
41	2	4	4	54
42	5	2	2	54
43	4	5	3	24
44	1	4	4	69
45	5	5	5	86
46	4	3	1	137
47	5	4	2	9
48	5	3	3	117
49	2	3	3	135
50	3	2	4	90

Tabela 16 (continuação) - Informações das tarefas para as instâncias da série A.

j	p_{1j}	p_{2j}	w_j	d_j
1	6	10	4	16
2	2	6	3	15
3	1	8	4	16
4	7	1	5	11
5	7	8	1	30
6	9	7	4	37
7	7	10	5	25
8	8	7	4	17
9	3	6	1	49
10	7	6	4	32
11	2	2	1	23
12	4	1	1	21
13	10	3	2	24
14	10	10	5	50
15	9	5	3	26
16	1	8	5	84
17	3	3	5	69
18	10	7	5	54
19	6	5	4	48
20	2	3	2	76
21	2	6	1	119
22	6	4	1	76
23	7	7	1	46
24	6	8	5	101
25	7	2	4	42

Tabela 17 - Informações das tarefas para as instâncias da série B.

j	p_{1j}	p_{2j}	w_j	d_j
26	10	5	2	104
27	1	2	4	73
28	2	9	4	84
29	7	5	5	62
30	7	6	3	38
31	10	5	5	177
32	5	8	3	42
33	9	8	4	120
34	8	5	3	53
35	6	3	2	127
36	4	6	5	168
37	3	5	3	70
38	9	7	1	186
39	9	2	5	108
40	5	4	5	18
41	1	4	2	200
42	2	10	4	182
43	10	6	5	24
44	1	8	4	177
45	5	7	1	22
46	3	8	2	261
47	5	7	4	85
48	3	1	1	208
49	5	5	4	246
50	2	1	2	62

Tabela 17 (continuação) - Informações das tarefas para as instâncias da série B.

j	p_{1j}	p_{2j}	w_j	d_j
1	20	84	1	104
2	71	64	1	137
3	68	22	1	142
4	17	75	1	209
5	13	7	1	93
6	4	71	1	92
7	7	54	1	295
8	8	32	1	75
9	35	11	1	174
10	17	97	1	180

Tabela 18 - Instâncias com tarefas que possuem tempo de processamento entre 1 e 100

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo