Tópicos em Métodos de Otimização para Programação de Tarefas e Problemas Correlatos

DÉBORA PRETTI RONCONI

Texto Sistematizado apresentado ao
Departamento de Engenharia de Produção
como requisito para o concurso de Livre
Docência

Departamento de Engenharia de Produção
Escola Politécnica da Universidade de São Paulo
São Paulo 2006

FT-2283

UNIVERSIDADE DE SÃO PAULO

REITORIA

Proc. USP nº 99.1.558.3.6
Portaria nº 1447
1694/vfm

A REITORA DA UNIVERSIDADE DE SÃO PAULO, usando de suas atribuições legais, **EXPEDE** a presente Portaria para declarar que tendo DEBORA PRETTI RONCONI, Professor Doutor, ref. MS-3, do QDUSP-PG, em Regime de Dedicação Integral à Docência e à Pesquisa, lotada na Escola Politécnica (Departamento de Engenharia de Produção), obtido o título de Livre-Docente, conforme elementos constantes do Processo USP nº 99.1.558.3.6, fica, a contar de 11/08/2006, designada para exercer a função de Professor Associado, ref. MS-5, do mesmo Quadro, Parte, Regime e Lotação, nos termos do artigo 84 do Estatuto da Universidade de São Paulo.

Reitoria da Universidade de São Paulo, 31 AGO 2006

SUELY VILELA
REITORA

Por Delegação da M. Reitora
Art. 42 do Estatuto da USP
FRANCO MARIA LAJOLO
Vice-Reitor

SUMÁRIO

CAPÍTULO 1. INTRODUÇÃO 01

CAPÍTULO 2. PROGRAMAÇÃO DE TAREFAS 07

2.1. Modelos de Programação Linear Inteira Mista 08

2.1.1. Minimizando o Atraso Total 10

2.1.2. Minimizando a Soma dos Adiantamentos e Atrasos 15

2.2. Algoritmos *Branch-and-Bound* 19

2.2.1. Minimizando o Atraso Total 20

2.2.2. Minimizando o *Makespan* 26

2.3. Heurísticas e Meta-Heurísticas 31

2.3.1. Máquina Única Minimizando a Soma dos Adiantamentos e Atrasos 32

2.3.2. *Flowshop* com Buffer Zero Minimizando o *Makespan* 50

2.3.3. *Flowshop* com Buffer Zero Minimizando o Atraso Total 62

2.3.4. *Flowshop* com Buffer Ilimitado Minimizando a Soma dos Adiantamentos e Atrasos 73

CAPÍTULO 3. PROBLEMAS DE EMPACOTAMENTO 89

3.1. Cilindros Idênticos em Objetos Retangulares 89

3.2. Retângulos Ortogonais em Convexos Arbitrários 98

CAPÍTULO 4. APLICAÇÕES 106

4.1. Planejamento da Distribuição de Açúcar e Álcool 106

4.2. Localização Dinâmica de Pontos de Armazenagem de Cloreto de Potássio 111

CAPÍTULO 5. CONCLUSÕES 119

BIBLIOGRAFIA 122

RESUMO

A grande variedade de problemas de natureza combinatória encontrada em setores da indústria tem motivado a proliferação de técnicas de resolução para os mesmos. Dado o bom desempenho dos métodos de otimização em inúmeras aplicações, durante os últimos anos a candidata abordou, como tema central de sua pesquisa, a aplicação/desenvolvimento de diversos métodos de otimização para a resolução de problemas de produção relacionados à área de Gestão de Operações e Logística. Neste texto são descritos artigos selecionados que abrangem problemas abordados pela candidata. Dentre estes destaca-se a resolução de problemas de programação de tarefas em diferentes ambientes. Além disso, foram considerados problemas de empacotamento e aplicações práticas. As técnicas de otimização mais utilizadas são: resolução de problemas através de modelos de programação linear e programação inteira, propostas de novos limitantes inferiores combinados com o algoritmo *branch-and-bound*, desenvolvimento de heurísticas construtivas e de busca local e aplicação de meta-heurísticas puras e híbridas. Além disso, estratégias mistas envolvendo métodos de programação não linear também foram elaboradas. Cada técnica foi selecionada de acordo com a complexidade e propriedades de cada problema. Bons resultados foram obtidos e os trabalhos estão publicados ou submetidos em revistas internacionais reconhecidas. Este texto também descreve trabalhos inéditos recentemente desenvolvidos.

ABSTRACT

The wide variety of combinatorial problems found in industrial environments has motivated the propagation of techniques to solve them. Due to the good performance of optimization methods in countless applications, the candidate has addressed the application/development of optimization methods for the resolution of production problems associated with Operations and Logistics Management, which has been the candidate's research topic in the past years. Selected papers comprising problems addressed by the candidate are described herein. Job-scheduling problems in various environments account for the majority of them. Packing problems and practical applications are also addressed. The applied optimization techniques include: resolution of problems using linear and integer programming models, introduction of new lower bounds associated with *branch-and-bound* algorithms, development of constructive and local search heuristics and the application of pure and hybrid metaheuristics. Furthermore, mixed strategies involving non-linear optimization methods are also described. Each method was selected according to the complexity and properties of each problem. Relevant results were achieved and those papers have been published or submitted for publication in renowned, well-established international journals. This work also describes unpublished papers that have been recently developed.

CAPÍTULO 1. INTRODUÇÃO

A grande variedade de problemas de natureza combinatória encontrada em setores da indústria tem motivado a proliferação de técnicas de resolução para os mesmos. Exemplos comumente encontrados incluem o *design* de redes de telecomunicações, programação de tarefas (*scheduling*), localização de depósitos e roteamento de veículos de entrega de bens e serviços, entre muitos outros. Teoricamente é possível enumerar todas as combinações de soluções e avaliá-las com respeito a um objetivo. Aquelas soluções que produzem os resultados mais favoráveis são conhecidas como soluções ótimas. Entretanto, por uma perspectiva prática, é inviável seguir tal estratégia porque o número de combinações, na maioria dos problemas, cresce exponencialmente com o seu tamanho. Nas últimas décadas, grande esforço de pesquisa tem sido dedicado ao desenvolvimento de métodos de resolução que não requerem explicitamente o exame de cada alternativa. Esta pesquisa resultou no surgimento da área de otimização combinatória que pode ser definida como a área que lida com problemas de maximização ou minimização de uma função com muitas variáveis sujeitas a restrições de desigualdade e igualdade, ou condições que obrigam que algumas ou todas as variáveis sejam inteiras (Nemhauser & Wosley 1988).

A formulação de problemas de otimização combinatória como problemas de programação inteira abre a possibilidade de aplicação de diversos algoritmos desenvolvidos para modelos gerais e específicos. Desde o trabalho pioneiro de Ralph Gomory no final dos anos 50, a programação inteira tem sido uma das áreas da Pesquisa Operacional que mais se desenvolveu. As abordagens mais conhecidas são: planos de corte e enumerações. Dentro da segunda abordagem podemos destacar a técnica *branch-and-bound* que é basicamente uma maneira sofisticada de realizar uma enumeração. Conforme Dantzig & Thapa (1997) a técnica *branch-and-bound* se tornou uma das formas mais bem sucedidas de resolução de problemas práticos de programação inteira.

Porém existem vários problemas de otimização combinatória que são de difícil resolução através de métodos que garantidamente fornecem a solução ótima. Nestes casos métodos heurísticos são geralmente empregados para encontrar rapidamente soluções não

necessariamente ótimas, porém de boa qualidade. Isto é fundamental para decisões de prazo muito curto como aquelas associadas a problemas reais. Conforme Rayward-Smith *et al.* (1996) a seguinte definição para os métodos heurísticos pode ser dada: "*Uma técnica heurística é um método que busca boas soluções a um custo computacional razoável sem ser capaz de garantir a otimalidade, e talvez nem a viabilidade. Infelizmente, nem sempre é possível estabelecer o quão próximo do valor ótimo a solução heurística está*". Embora à luz desta definição pareça que existem graves problemas inerentes ao uso das heurísticas, deve ser ressaltado que várias heurísticas modernas fornecem soluções de alta qualidade na prática. De acordo com Zanakis *et al.* (1989) a área de produção, especialmente na programação de tarefas, é uma das áreas que mais utiliza métodos heurísticos.

Uma heurística popular com largas aplicações é a busca local ou busca em vizinhança. A idéia básica é muito simples: inicia-se de uma solução qualquer e busca-se em uma certa vizinhança a melhor solução possível. A partir desta, o processo de busca recomeça e prossegue até que nenhuma melhoria seja obtida. A solução final é chamada ótimo local. A desvantagem evidente deste método é que este ótimo local na maioria das vezes não é o ótimo global. Buscando evitar que a heurística estacione nestes ótimos locais surgem as meta-heurísticas que trabalham como guias da busca local direcionando seus movimentos para a exploração de diferentes regiões do espaço de soluções. Atualmente as meta-heurísticas mais conhecidas por seu bom desempenho são Busca Tabu, *Simulated Annealing*, Algoritmos Genéticos e *Greedy Ramdomized Adaptive Search Procedures* (Adenso-Díaz *et al.* 1996). Além disso, as meta-heurísticas híbridas também tem recebido considerável atenção no campo de otimização combinatória (veja Talbi 2002).

Dado o bom desempenho dos métodos de otimização em inúmeras aplicações, durante os últimos anos a candidata abordou como tema central de sua pesquisa a aplicação/desenvolvimento de diversos métodos de otimização para a resolução de problemas de produção relacionados à área de Gestão de Operações e Logística. Dentre os problemas abordados, destaca-se a resolução de problemas de programação de tarefas em diferentes ambientes. Além disso, foram considerados problemas de empacotamento (*packing problems*) e aplicações práticas. Os principais resultados obtidos se resumem a: 10 artigos em revistas internacionais (*Annals of Operations Research, Computers and Operations Research, European Journal of Operational Research, International Journal of*

Production Economics, International Transactions in Operational Research e *Journal of the Operational Research Society*), 4 artigos em revistas nacionais (Caderno de Engenharia de Produção/DEP, Gestão & Produção, GEPROS: Gestão da Produção e Sistemas e Pesquisa Operacional), 4 artigos submetidos a revistas internacionais e diversos trabalhos (completos, resumos estendidos e resumos) em conferências nacionais e internacionais. Esta pesquisa está relacionada à Bolsa de Produtividade em Pesquisa do CNPq, atualmente nível 1D, que a candidata possui desde março de 2002. Segue uma breve descrição dos trabalhos efetuados ao longo de sua carreira de pesquisadora.

Um tópico amplamente tratado dentro da pesquisa da candidata foi o problema de programação de tarefas. Considerando o ambiente *flowshop* com capacidade ilimitada de armazenamento entre as máquinas, Ronconi & Armentano (1995) e Armentano & Ronconi (1999) propuseram métodos heurísticos baseados na meta-heurística Busca Tabu para a minimização do atraso total das tarefas. Com o mesmo enfoque de resolução, Armentano & Ronconi (2000) abordaram este ambiente com capacidade de armazenamento zero entre as máquinas. A resolução deste problema através da meta-heurística GRASP também foi considerada no recente trabalho de Ronconi & Henriques (2005). Em Ronconi & Armentano (2001), um limitante inferior do atraso total foi proposto, possibilitando o desenvolvimento de um algoritmo *branch-and-bound* e conseqüentemente a realização de comparações com soluções heurísticas. Ronconi & Togashi (2002) realizaram uma comparação entre formulações inteiras mistas para o ambiente *flowshop* com buffer ilimitado e zero. Uma versão preliminar deste trabalho foi selecionada como um dos cinco melhores trabalhos de Iniciação Científica pela Sociedade Brasileira de Pesquisa Operacional (2000-2001).

O problema de programar tarefas no ambiente *flowshop* com buffer zero também foi abordado com outra medida de desempenho: a minimização do *makespan*. Considerando este critério de otimização, Ronconi (2004) desenvolveu uma heurística construtiva e Ronconi (2005) apresentou um limitante inferior. O limitante proposto foi avaliado através da implementação de um algoritmo *branch-and-bound* específico para esta medida de desempenho.

(Kawamura *et al.*, 2005). Outro trabalho que também lida com formulações para problemas práticos pode ser encontrado em Santoro & Ronconi (2005). Os autores apresentam um modelo de programação linear inteira mista para um problema dinâmico de localização de armazéns para a distribuição de cloreto de potássio.

Neste trabalho são descritos artigos selecionados, dentre os citados anteriormente, onde várias técnicas de otimização foram aplicadas a problemas de produção. Esta seleção foi realizada de forma a abranger os diversos tipos de problemas abordados pela candidata durante o seu trabalho de pesquisa. Nesta seleção incluem-se trabalhos publicados recentemente, submetidos ou ainda inéditos

No Capítulo 2 são considerados os problemas de programação de tarefas. Na primeira seção são descritos modelos de programação linear inteira para o ambiente *flowshop* com diferentes critérios envolvendo datas de entrega apresentados nos trabalhos de Ronconi & Togashi (2002) e Sakuraba & Ronconi (2006). Na Seção 2.2 são apresentados limitantes inferiores introduzidos por Ronconi & Armentano (2001) e por Ronconi (2005), respectivamente, para a minimização da soma dos atrasos e do *makespan* no ambiente *flowshop* com buffer zero. Todos os limitantes foram avaliados através do desenvolvimento de algoritmos *branch-and-bound* específicos para cada problema. Na última seção trabalhos envolvendo técnicas heurísticas são relatados. As diversas metodologias propostas por Hino *et al.* (2005) para o ambiente com máquina única minimizando o adiantamento e o atraso das tarefas são descritas. Em seguida apresentamos os trabalhos de Ronconi (2004) e Ronconi & Henriques (2005) que abordam o ambiente *flowshop* com buffer zero com diferentes medidas desempenho. Além disso, neste capítulo descrevemos o estudo desenvolvido em Sakuraba & Ronconi (2006) para o ambiente *flowshop* com duas máquinas para a minimização dos desvios absolutos dos instantes de término das tarefas em relação a uma data de entrega comum.

No Capítulo 3 apresentam-se trabalhos recentes que lidam com o problema de empacotar itens dentro de objetos utilizando heurísticas baseadas em subproblemas de programação não linear. Na Seção 3.1 descreve-se o trabalho de Birgin *et al.* (2005a) onde o empacotamento de cilindros idênticos em um contêiner retangular é abordado. Em

seguida descremos na Seção 3.2 a estratégia elaborada por Birgin *et al.* 2005b para o problema de empacotamento de retângulos ortogonais em regiões convexas.

Aplicações são consideradas no Capítulo 4 onde descrevemos os trabalhos de Kawamura *et al.* (2005) e Santoro & Ronconi (2005), que apresentam, respectivamente, modelos de programação linear e de programação linear inteira mista para a resolução de problemas práticos. No Capítulo 5 são apresentadas as conclusões deste trabalho e linhas futuras de pesquisa.

CAPÍTULO 2. PROGRAMAÇÃO DE TAREFAS

Neste capítulo será abordado o problema de programação de tarefas, conhecido na língua inglesa como *job scheduling*, que consiste na alocação de recursos para tarefas respeitando as restrições impostas e otimizando um ou mais critérios. Os recursos e tarefas podem ter várias formas. Os recursos podem ser máquinas numa fábrica, costureiras de uma confecção, pistas de um aeroporto, unidades de processamento de um computador, etc.. As tarefas podem ser operações num processo de produção, aterrissagens e decolagens de aviões, execuções de programas, etc.. As tarefas podem ter níveis de prioridades, disponibilidades e datas de entrega distintas. De forma geral, o problema de programação de tarefas é muito complexo devido a sua natureza combinatória e, dentre os problemas de otimização combinatória, pode ser classificado como um dos mais difíceis.

Para classificar a maioria dos problemas de programação de tarefas é necessário caracterizar a configuração dos recursos e o comportamento das tarefas. Por exemplo, um problema pode conter um tipo de recurso ou vários. Se contiver um tipo, as tarefas são provavelmente de um estágio, enquanto problemas com múltiplos recursos envolvem tarefas multiestágios. Em ambos casos os recursos podem estar disponíveis em quantidade unitária ou em paralelo. Além disso, se o conjunto de tarefas disponível para a programação não se modifica no tempo, o sistema é denominado estático, senão o sistema é conhecido como dinâmico. Tradicionalmente, os modelos estáticos são mais tratáveis que os dinâmicos e tem sido alvo de vários estudos. Segundo Baker (1974), os modelos estáticos freqüentemente capturam a essência de sistemas dinâmicos mais complexos e a análise destes modelos costuma descobrir valiosos *insights* que são úteis em situações mais gerais.

Na maioria dos trabalhos teóricos sobre *scheduling*, medidas simples de desempenho foram aplicadas, como por exemplo a minimização do *makespan* (Conway *et al.*, 1967; Pinedo, 1995). Em geral estas medidas são medidas regulares, ou seja não-decrescentes com o instante de término das tarefas. Dentre estas podemos citar medidas de desempenho relacionadas com atraso, como a minimização da soma dos atrasos, cuja dificuldade se baseia no fato de que o atraso não é uma função linear do

makespan (Baker 1974). Com o passar dos anos tais medidas passaram a ser mais exploradas, especialmente nos ambientes com múltiplas máquinas, tendo em vista a crescente importância para o mercado de fatores ligados a determinação e ao cumprimento de datas de entrega (Kim, 1995).

Além disso, diversos artigos sobre programação de tarefas têm utilizado a penalização por adiantamento em adição à penalização por atraso (veja revisões da literatura em Baker & Scudder, 1990; Gordon *et al.*, 2002). Essa tendência se deve à propagação da filosofia *Just In Time* (JIT) pelas indústrias, segundo a qual os produtos devem estar disponíveis o mais próximo possível do instante de início de sua utilização. O JIT visa assim reduzir o volume e conseqüentemente os custos de estoque, rotulando como indesejáveis peças fabricadas tanto antes quanto depois da data estipulada. Segundo Baker & Scudder (1990), os modelos com penalização por atrasos e adiantamentos refletem melhor este tipo de ambiente. Outro exemplo de utilização deste tipo de modelagem é a indústria química, citada por Sarper (1995), onde podemos encontrar diferentes produtos a serem confeccionados através de um mesmo processo e que precisam ser misturados o mais próximo possível de um determinado instante para evitar a sua deterioração. Outras aplicações na indústria química podem ser encontradas em Reklaitis (1982).

Com o intuito de explorar medidas de desempenho pouco tratadas na literatura várias pesquisas foram realizadas recentemente. Nas seções seguintes são resumidamente descritos trabalhos onde diversas técnicas de otimização foram aplicadas a problemas estáticos de programação de tarefas considerando especialmente a minimização de medidas relacionadas com o atraso e/ou o adiamento de tarefas.

2.1. Modelos de Programação Linear Inteira Mista

Uma surpreendente ampla classe de problemas práticos pode ser modelada utilizando variáveis inteiras e restrições lineares (Williams, 1999). Algumas vezes o modelo consiste somente de variáveis inteiras, porém mais freqüentemente ocorrem dois tipos de variáveis: inteiras e contínuas. Tal modelo é conhecido como um modelo de programação linear inteira mista (PLIM). Com o aprimoramento de softwares específicos

para resolução deste tipo de problema, pesquisas neste campo foram amplamente impulsionadas.

Os modelos PLIM podem ser utilizados para a programação de tarefas de forma a otimizar diversos critérios de desempenho. O principal objetivo desta seção é descrever modelos PLIM para a programação de tarefas no ambiente *flowshop* com diferentes critérios envolvendo datas de entrega apresentados nos trabalhos de Ronconi & Togashi (2002) e Sakuraba & Ronconi (2006).

O ambiente de produção *flowshop* é caracterizado pelo processamento de n tarefas por m máquinas na mesma ordem, ou seja, a j-ésima operação de todas as tarefas deve ser sempre feita na máquina j. Neste ambiente a quantidade de tarefas e máquinas, o tempo de processamento das operações e a data de entrega das tarefas são dados conhecidos e fixos. Além disso, todas as tarefas estão disponíveis para processamento no instante zero; os tempos de preparação das máquinas são independentes da seqüência e estão incluídos nos tempos de processamento; as máquinas estão continuamente disponíveis e as operações não podem ser interrompidas. Estas propriedades definem que este ambiente pode ser visualizado como um problema determinístico e estático. O problema consiste em determinar a seqüência de processamento das tarefas nas máquinas e o início do processamento de cada tarefa em cada máquina de forma a otimizar um critério de desempenho selecionado.

Dentre diversas aplicações, a manufatura de papel, o processamento e refinamento de metais e a fabricação de pães podem ser caracterizados por um processo de fluxo (Raman, 1995). Dudek *et al.* (1992) e Reklaitis (1982) indicam as indústrias químicas como uma das mais relevantes aplicações do *flowshop*. Dado que os equipamentos específicos para os processos químicos são caros, existe uma forte motivação para otimizar este ambiente de produção. Além disso, conforme Baker (1974), o atendimento de datas de entregas é uma das medidas de desempenho mais comumente encontradas na prática.

2.1.1. Minimizando o Atraso Total

Ronconi & Togashi (2002) abordam o ambiente *flowshop* com buffer zero que é caracterizado por não apresentar estoques intermediários entre as diversas máquinas. Portanto, pode ocorrer uma situação conhecida como bloqueio, na qual a máquina interrompe o seu ciclo de produção mesmo tendo finalizado a tarefa, devido à não liberação da máquina seguinte. Somente a capacidade de armazenamento intermediária zero foi considerada pois qualquer *flowshop* com capacidade de armazenamento finita não-zero pode ser modelado como um problema de *flowshop* com capacidade de armazenamento ou buffer zero. Isto ocorre porque pode-se simular a capacidade de se armazenar uma tarefa por uma máquina fictícia com tempos de processamento nulos (McCormick *et al.* 1989).

O problema de programação de tarefas com capacidade limitada de armazenamento é um ambiente comumente encontrado na produção industrial. Com o desenvolvimento da manufatura *Just In Time* e dos sistemas kanban que mantêm um estoque limitado e fixo de tarefas em processamento, este problema assume grande significância, e necessita ser intensamente estudado (Leisten 1990, Berkley 1993, Hall & Sriskandarajah 1996, Sharadapriyadarshini & Rajendran 1996).

O critério de otimização a ser adotado é a minimização da soma dos atrasos. Medidas de desempenho envolvendo datas de entrega são de grande importância nos sistemas de manufatura, pois pode existir uma série de custos quando uma tarefa é entregue com atraso, como por exemplo: penalidades contratuais, perda de credibilidade e danos na reputação da empresa que poderão afastar outros clientes (Sen & Gupta, 1984).

Modelos

Dentre os ambientes *flowshop*, um dos mais tratados na literatura é o *flowshop* com buffer ilimitado. De acordo com Stafford *et al.* (2005) o modelo que melhor representa este ambiente minimizando o critério do *makespan* é a formulação inteira mista proposta por Wagner (1959) e aprimorada por Stafford (1988). Estes autores afirmam que os modelos baseados na estrutura proposta por Wagner, que utiliza alocação de tarefas a posições, requerem um tempo computacional muito menor que outras formulações, como por

exemplo as baseadas no modelo proposto por Manne (1960) com restrições dicotômicas para controlar a ordem relativa das tarefas. Conclusões semelhantes são apresentadas em Ronconi & Togashi (2002) que realizam comparações entre formulações baseadas nos modelos de Wagner (1959) e Manne (1960) para a minimização da soma dos atrasos.

Para o ambiente *flowshop* com buffer zero minimizando a soma dos atrasos Ronconi (1997) propõe uma adaptação do modelo de Wagner porém não apresenta testes computacionais. Ronconi & Togashi (2002) apresentam um estudo comparativo onde tal formulação é avaliada e comparada com outros trabalhos da literatura.

Apresenta-se a seguir o modelo de programação inteira mista avaliado:

Parâmetros:

n: número de tarefas;
m: número de máquinas;
p_{ik}: tempo de processamento da tarefa i na máquina k;
d_i: data de entrega da tarefa i;

Variáveis:

X_i: variável de decisão: assume valor 1 caso a tarefa i esteja na posição j da seqüência, caso contrário, assume valor 0;
I_{jk}: tempo ocioso (*idle time*) na máquina k entre o processamento das tarefas nas posições j e $j+1$;
T_j: atraso da tarefa na posição j.
B_{jk}: tempo de bloqueio da máquina k ao término do processamento da tarefa na posição j;
D_{jk}: instante de liberação da máquina k após o processamento da tarefa na posição j.

Formulação:

minimizar $$\sum_{i=1}^{n} T_j \qquad (1)$$

sujeito a

$$T_j \geq D_{jm} - \sum_{i=1}^{n} X_{ij} . d_i \qquad j = 1,\ldots, n \qquad (2)$$

$$D_{1m} = \sum_{k=1}^{m} \sum_{i=1}^{n} X_{i1} \cdot p_{ik} \qquad (3)$$

$$D_{jm} = D_{j-1,m} + I_{j-1,m} + \sum_{i=1}^{n} X_{ij} \cdot p_{im} \qquad j = 2,\ldots, n \qquad (4)$$

$$\sum_{i=1}^{n} X_{ij} = 1 \qquad j = 1,\ldots, n \qquad (5)$$

$$\sum_{j=1}^{n} X_{ij} = 1 \qquad i = 1,\ldots, n \qquad (6)$$

$$I_{jk} + \sum_{i=1}^{n} X_{i,j+1} \cdot p_{ik} + B_{j+1,k} - B_{j,k+1} - \sum_{i=1}^{n} X_{ij} \cdot p_{i,k+1} - I_{j,k+1} = 0 \qquad (7)$$

$$j = 1,\ldots, n-1;\ k = 1,\ldots, m-1$$

$$T_j \geq 0 \qquad j = 1,\ldots, n \qquad (8)$$

$$I_{jk} \geq 0 \qquad j = 1,\ldots, n-1;\ k = 2,\ldots, m \qquad (9)$$

$$B_{jk} \geq 0 \qquad j = 2,\ldots, n;\ k = 1,\ldots, m-1 \qquad (10)$$

$$X_{ij} \in \{0,1\} \qquad i = 1,\ldots, n;\ j = 1,\ldots, n \qquad (11)$$

$$I_{j1} = 0 \qquad j = 1,\ldots, n-1 \qquad (12)$$

$$B_{1k} = 0 \qquad k = 1,\ldots, m-1 \qquad (13)$$

$$B_{jm} = 0 \qquad j = 1,\ldots, n \qquad (14)$$

A função objetivo (1) representa a soma dos atrasos. Os conjuntos de restrições (2) e (8) correspondem à linearização de $T_j = max(D_{jm} - \sum_{i=1}^{n} X_{ij} . d_i ; 0)$ e nos fornecem o valor do atraso individual de cada tarefa. A restrição (3) representa o fato de que o instante de liberação da primeira tarefa da seqüência na última máquina é igual à soma dos tempos de processamento em todas as máquinas. Já para as outras tarefas da seqüência, temos o conjunto de restrições (4) que representa o cálculo dos seus instantes de término na máquina *m*; observe que estas restrições envolvem também o tempo ocioso desta máquina. As restrições (5) e (6) são as restrições que garantem, respectivamente, que cada posição será ocupada por apenas uma tarefa e que cada tarefa ocupará apenas uma posição na seqüência.

O conjunto de restrições (7) relaciona as variáveis de decisão com as restrições físicas do problema. Estas restrições estabelecem as relações necessárias para manter a consistência entre o tempo ocioso das máquinas e o tempo de bloqueio das máquinas. Estas representam a diferença entre o instante $\mathbf{t_1}$ em que a tarefa na posição *j* libera a máquina *k*, e o instante $\mathbf{t_2}$ em que a tarefa na posição *j+1* inicia seu processamento na máquina *k+1*, como ilustrado na Figura 1. Espera-se que a relação descrita por $B_{j+1,k} \cdot I_{j,k+1} = 0$ seja satisfeita, porém, esta condição não é obrigatória em uma solução ótima.

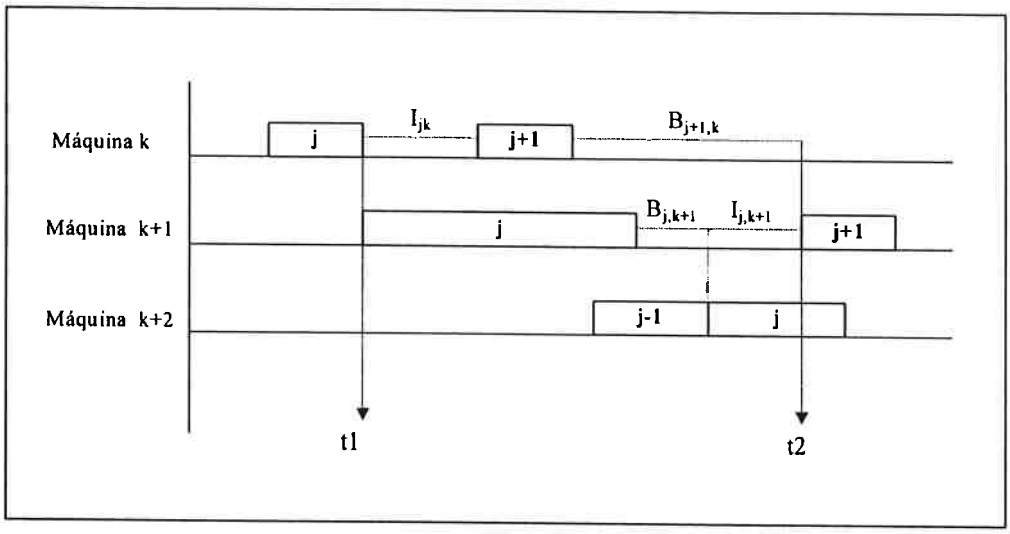

Figura 1: Representação gráfica da restrição (7).

As restrições (9) e (10) indicam que os tempos ociosos e de bloqueio não podem ser negativos e em (11) indica-se que as variáveis de decisão são binárias. Os três últimos conjuntos de restrições representam o fato de que na primeira máquina não há tempo ocioso, que a primeira tarefa nunca encontra bloqueio e que na última máquina não ocorrerá bloqueio. O número de restrições desta formulação é **3n + nm - m + 1**, o número de variáveis binárias é $\mathbf{n^2}$ e o número de variáveis contínuas não negativas é **2nm - 2m + 2**.

Um modelo alternativo para o problema de bloqueio foi apresentado no trabalho de Ronconi & Togashi (2002) a partir do trabalho de Leisten (1990). Neste artigo, o autor sugere que a situação de bloqueio pode ser modelada matematicamente através de três conjuntos de restrições, que serão apresentadas a seguir, juntamente com a formulação.

Variáveis:

S_{jk}: instante de início de processamento da tarefa de posição j na máquina k;

C_{jk}: instante de término da tarefa na posição j na máquina k.

Formulação:

minimizar $$\sum_{i=1}^{n} T_j \qquad (15)$$

Sujeito a $$T_j \geq C_{jm} - \sum_{i=1}^{n} X_{ij}.d_i \qquad j = 1,\ldots, n \qquad (16)$$

$$C_{jk} = S_{jk} + \sum_{i=1}^{n} X_{ij} \cdot p_{ik} \qquad j = 1,\ldots, n;\ k = 1,\ldots,m \qquad (17)$$

$$S_{jk} \geq C_{j,k-1} \qquad j = 1,\ldots, n;\ k = 2,\ldots,m \qquad (18)$$

$$S_{jk} \geq C_{j-1,k} \qquad j = 2,\ldots, n;\ k = 1,\ldots,m \qquad (19)$$

$$S_{jk} \geq S_{j-1,k+1} \qquad j = 2,\ldots, n;\ k = 1,\ldots, m-1 \qquad (20)$$

$$\sum_{j=1}^{n} X_{ij} = 1 \qquad i = 1,\ldots, n \qquad (21)$$

$$\sum_{i=1}^{n} X_{ij} = 1 \qquad j = 1,\ldots, n \qquad (22)$$

$$C_{jk} \geq 0 \qquad j = 1,\ldots, n;\ k = 1,\ldots,m \qquad (23)$$

$$S_{jk} \geq 0 \qquad j = 1,\ldots, n;\ k = 1,\ldots,m \qquad (24)$$

$$T_j \geq 0 \qquad j = 1,\ldots, n \qquad (25)$$

$$X_{ij} \in \{0,1\} \qquad i = 1,\ldots, n;\ j = 1,\ldots, n \qquad (26)$$

$$S_{11} = 0 \qquad (27)$$

Os conjuntos de restrições (16) e (25) fornecem os valores de atraso para as diversas tarefas, enquanto o conjunto (17) indica que o instante de término de uma tarefa é igual ao instante de início da mesma mais o seu tempo de processamento. As restrições (18), (19) e (20) envolvem o cálculo do instante de início das tarefas em cada máquina e representam a linearização de $S_{jk} = max\ (C_{j,k-1}\ ;\ C_{j-1,k};\ S_{j-1,k+1})$. Para cada variável S_{jk} apenas uma restrição estará ativa, enquanto as outras duas estarão inativas. Se houver bloqueio, a restrição (20) estará ativa. Caso contrário, se a tarefa na posição j for finalizada na máquina anterior, k-1, em um instante de tempo superior ao instante que a tarefa na posição j-1 foi finalizada na máquina k então a restrição (18) estará ativa, senão a restrição (19) estará ativa.

Os conjuntos de restrições (21) e (22) garantem, respectivamente, que cada posição será ocupada por apenas uma tarefa e que cada tarefa ocupará apenas uma posição na seqüência. As restrições (23) e (24) indicam que as variáveis são positivas enquanto as restrições (26) que as variáveis de decisão são binárias. A restrição (27) indica que a primeira máquina está preparada para processamento no instante de tempo zero. O número de restrições desta formulação é de **4mn + n - 2m + 1**, o número de variáveis binárias é $\mathbf{n^2}$ e o número de variáveis contínuas não negativas é **2nm + n -1**. Observe que esta formulação,

embora apresente o mesmo número de variáveis inteiras que a formulação anterior, possui um maior número de variáveis contínuas e restrições.

Um estudo comparativo entre as duas formulações apresentadas foi realizado em Ronconi & Togashi (2002). Noventa e seis problemas com oito diferentes dimensões (n = 5, 10 e 20 e m = 5, 15 e 20) foram resolvidos utilizando o software AMPL/CPLEX 7.0, versão profissional. Os resultados indicaram que a formulação proposta por Ronconi (1997) foi a mais rápida, possuindo as melhores médias em 7 das 8 dimensões analisadas. Além disso, os tempos computacionais até a primeira solução inteira também são melhores que os da formulação adaptada de Leisten (1990). Além disso, a formulação proposta por Ronconi (1997) apresentou desempenho bastante uniforme, inclusive alterando-se as características dos problemas, ou seja, a variação dos tempos gastos para resolução em diferentes cenários de datas de entrega foi pequena. Uma descrição detalhada deste trabalho pode ser encontrada em Ronconi & Togashi (2002).

2.1.2. Minimizando a Soma dos Adiantamentos e Atrasos

O problema tratado em Sakuraba & Ronconi (2006) corresponde à programação de tarefas em um ambiente *flowshop* com duas máquinas. O objetivo considerado é a minimização da soma dos desvios das datas de conclusão de todas as tarefas em relação a uma data de entrega comum (*SAD – Sum of Absolute Deviations*). Esse critério é uma medida de desempenho não-regular, por abranger tanto penalizações por atraso quanto por adiantamento. Diz-se que uma medida de desempenho é regular quando ao adiantarmos a programação de qualquer uma das tarefas obtém-se um ganho ou manutenção do valor da função objetivo (Baker & Scudder, 1990). Caso desejássemos minimizar o *makespan* ou o atraso total, bastaria programar as tarefas para começarem seus processamentos o quanto antes em ambas as máquinas. Como em nosso problema tratamos de uma medida de desempenho não-regular, além da seqüência de tarefas, precisamos determinar sua programação.

Além disso, considera-se uma mesma seqüência de tarefas nas máquinas, caso conhecido como **programação permutacional** (*permutational scheduling*). Dada essa restrição, o número de seqüências possíveis se reduz de $(n!)^m$ para $(n!)$. As programações

permutacionais nem sempre incluem a programação ótima para um dado problema, no entanto, sua importância não deve ser subestimada, dado que muitas vezes apenas programações permutacionais são possíveis em situações reais. Com o intuito de analisar o impacto desta restrição, Kim (1993) apresenta comparações entre programas de permutação e não-permutação obtidos com regras de prioridades para o ambiente *flowshop* minimizando o atraso total. O desempenho das regras que consideram os programas de não-permutação se mostra levemente superior ao desempenho das regras que consideram somente os programas de permutação. Porém, quando adicionou-se uma busca local baseada em trocas de pares de tarefas adjacentes na solução das regras que só consideram permutação, obteve-se um ganho considerável sobre as regras de despacho que consideram programas de não-permutação. O autor conclui que a facilidade com a qual os programas permutacionais podem ser melhorados é a sua maior vantagem.

A Figura 2 apresenta o Gráfico de Gantt de um exemplo da programação de cinco tarefas neste ambiente com duas máquinas.

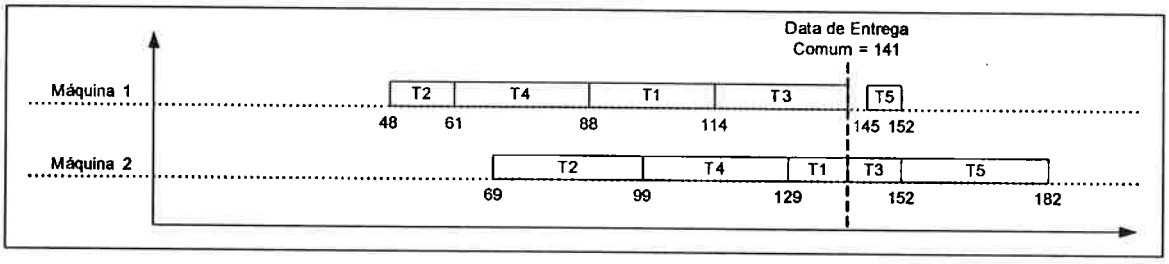

Figura 2 - Exemplo da programação de cinco tarefas no ambiente flowshop.

Modelos

Em Sakuraba & Ronconi (2006) os autores apresentam uma formulação inteira mista para o problema de minimizar a soma dos desvios no ambiente *flowshop* com duas máquinas. Esse modelo foi adaptado daquele exposto por Wagner (1959) para a minimização do *makespan* no ambiente *flowshop*. Apesar de menos intuitivo do que outros modelos, segundo Stafford *et al.* (2005) e Ronconi & Togashi (2002), os modelos baseados no trabalho de Wagner apresentam um melhor desempenho quanto ao critério tempo de processamento em pacotes comerciais de otimização.

Sarper (1995) afirma que no problema considerado não é necessário adicionar qualquer tempo ocioso entre tarefas na primeira máquina, sendo que as tarefas podem ser programadas nesta máquina para iniciar seu processamento tão cedo quanto possível sem prejudicar o valor de função objetivo. Entretanto, tal afirmação não é válida para a programação na segunda máquina. Tal fato é considerado na formulação proposta de forma que inícios tardios da programação na segunda máquina são admitidos. Apresenta-se a seguir o modelo de programação inteira mista proposto.

Parâmetros:

n : número de tarefas a serem programadas;
d : data de entrega comum;
p_{ik} : tempo de processamento da tarefa i na máquina k.

Variáveis:

x_{ij} : 1 se a tarefa i ocupa a j-ésima posição da seqüência, 0 caso contrário;
E_j : adiantamento da tarefa na posição j;
T_j : atraso da tarefa na posição j;
c_{jk} : instante de término da tarefa na posição j na máquina k;
I_{jk} : tempo ocioso na máquina k entre o processamento das tarefas na posição j e j+1;
W_{jk} : tempo de espera (*waiting time*) da tarefa na posição j entre as máquinas k e $k+1$.

Formulação:

minimizar $$\sum_{j=1}^{n}(E_j + T_j) \tag{28}$$

sujeito a $$c_{j2} = d - E_j + T_j \qquad j = 1, 2, \ldots, n \tag{29}$$

$$c_{j2} = c_{j-1,2} + I_{j-1,2} + \sum_{i=1}^{n} p_{i2} x_{ij} \qquad j = 1, 2, \ldots, n \tag{30}$$

$$I_{j1} + \sum_{i=1}^{n} p_{i1} x_{i,j+1} + W_{j+1,1} = W_{j1} + \sum_{i=1}^{n} p_{i2} x_{ij} + I_{j2} \qquad j = 1, 2, \ldots, n-1 \tag{31}$$

$$I_{01} + \sum_{i=1}^{n} p_{i1} x_{i1} + W_{11} = I_{02} \tag{32}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \ldots, n \tag{33}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \ldots, n \tag{34}$$

$$I_{jk} \geq 0 \qquad j = 0, 1, \ldots, n;\ k = 1, 2 \qquad (35)$$

$$W_{j1} \geq 0 \qquad j = 1, 2, \ldots, n \qquad (36)$$

$$x_{ij} \in \{0,1\} \qquad i = 1, 2, \ldots, n; j = 1, 2, \ldots, n \qquad (37)$$

$$c_{jk} \geq 0 \qquad j = 0, 1, \ldots, n;\ k = 1, 2 \qquad (38)$$

$$c_{02} = 0 \qquad (39)$$

$$E_j, T_j \geq 0 \qquad j = 1, 2, \ldots, n \qquad (40)$$

A função objetivo (28) representa a soma dos desvios em relação a uma data de entrega comum. O conjunto de restrições (29) calcula os valores de adiantamento e atraso das tarefas. O conjunto (30) calcula os instantes de conclusão das tarefas na segunda máquina. O conjunto (31) relaciona as variáveis de decisão com as restrições físicas do problema. A restrição (32) é uma particularização do conjunto (31) na qual acrescentamos uma tarefa fictícia "0" para que a tarefa na primeira posição não inicie seu processamento na segunda máquina obrigatoriamente no instante de término de seu processamento na primeira máquina como exemplificado na Figura 2. Os conjuntos (33) e (34) indicam, respectivamente, que cada posição é ocupada por apenas uma tarefa e que cada tarefa ocupa apenas uma posição. As outras restrições são de não-negatividade ou ditam que as variáveis x_{ij} devem ser binárias. O número de restrições desta formulação é **5n**, o número de variáveis binárias é $\mathbf{n^2}$ e o número de variáveis contínuas não negativas é **7n + 2**.

Poucos autores consideram o ambiente *flowshop* com o critério de minimização da soma dos desvios em relação a uma data comum e, especificamente com duas máquinas, o único trabalho encontrado é o de Sarper (1995). Este autor apresenta uma formulação linear inteira mista para este problema utilizando restrições dicotômicas para controlar a ordem relativa das tarefas. O modelo proposto permite a obtenção de diferentes seqüências nas máquinas (não-permutacional).

Com intuito de avaliar o desempenho da formulação exposta anteriormente, 30 problemas de dimensão reduzida sugeridos por Sarper (1995) foram resolvidos através do

pacote de otimização ILOG/CPLEX 7.0. Os problemas foram modelados utilizando-se a formulação proposta por Sarper (1995), com uma adaptação para a geração de programas permutacionais, e a formulação de Sakuraba & Ronconi (2006).

Analisando-se o resultado desta avaliação (Tabela 1) observou-se que a formulação proposta em Sakuraba & Ronconi (2006) possui melhor desempenho em termos de tempo computacional. Isto se deve ao fato de que a formulação proposta explora características do *flowshop* permutacional através das variáveis W_{ij} e I_{ij} que são conectadas pelas restrições (31) e (32) desta formulação.

Tabela 1. Tempo em segundos para as formulações avaliadas.

Dimensão $n \times m$	Sakuraba & Ronconi	Sarper
5 x 2	0,025	0,092
6 x 2	0,030	0,426
10 x 2	0,037	205,69

Além disso, esta análise confirma conclusões anteriores de Stafford *et al.* (2005) e Ronconi & Togashi (2002) que afirmam que para o ambiente *flowshop* modelos baseados na estrutura proposta por Wagner, que utiliza alocação de tarefas a posições, requerem um tempo computacional muito menor que outras formulações. Maiores detalhes sobre este estudo podem ser encontrados em Sakuraba & Ronconi (2006).

2.2. Algoritmos *Branch-and-Bound*

A formulação de problemas de otimização combinatória como problemas de programação inteira abre a possibilidade da aplicação de diversos algoritmos desenvolvidos para modelos gerais e específicos. Dentre as abordagens mais conhecidas estão planos de corte e enumerações; e dentro da segunda abordagem podemos destacar a técnica *branch-and-bound* que é basicamente uma maneira sofisticada de realizar uma enumeração. Conforme Dantzig & Thapa (1997) a técnica *branch-and-bound* se tornou uma das formas mais bem sucedidas de resolução de problemas práticos de programação inteira. O termo *Branch* ou ramificação refere-se ao particionamento do espaço de soluções. Cada parte deste espaço é considerada separadamente. O termo *Bound* ou limitação refere-se ao

desenvolvimento de limitantes para tais partes. A qualidade dos limitantes é crucial para o sucesso do algoritmo *branch-and-bound*; estes devem ser bons o suficiente para reduzir o número de subproblemas considerados e conseqüentemente o tempo computacional necessário para a sua execução.

Nesta seção apresentaremos limitantes inferiores introduzidos por Ronconi & Armentano (2001) e por Ronconi (2005), respectivamente, para a minimização da soma dos atrasos e do *makespan* no ambiente *flowshop* com buffer zero.

2.2.1. Minimizando o Atraso Total

O trabalho de Ronconi & Armentano (2001) considera o ambiente *flowshop* com buffer zero, ou seja, neste ambiente não existem estoques intermediários entre as diversas máquinas (veja Seção 2.1.1). A medida de desempenho considerada é a minimização da soma dos atrasos das tarefas em relação às suas respectivas datas de entrega.

A pesquisa sobre *flowshop* com buffer zero e limitado com o critério do atraso é limitada e a maioria dos estudos considera o critério de minimização do *makespan*. Por tal motivo, apresenta-se a seguir uma pequena revisão considerando este critério e métodos exatos. Uma boa revisão deste ambiente pode ser encontrada em Hall & Sriskandarajah (1996). Estes autores provam que o problema *flowshop* com três máquinas e buffer zero é *NP-hard in the strong sense*. Reddi & Ramamoorthy (1972) mostraram que para duas máquinas este problema pode ser reduzido a um caso especial do caixeiro viajante, que pode ser resolvido em tempo polinomial pelo algoritmo de Gilmore & Gomory (1964). Dutta & Cunningham (1975) e Reddi (1976) utilizam programação dinâmica para obter soluções ótimas para este problema com buffer limitado.

Levner (1969) e Suhami & Mah (1981) apresentam algoritmos *branch-and-bound* que tratam especificamente do ambiente *flowshop* com buffer zero. Levner mostra que calcular o *makespan* de um *flowshop* é equivalente a encontrar o caminho mais longo em um grafo direcionado. Resultados computacionais não são apresentados, mas o autor afirma ter encontrado soluções ótimas para problemas com $n \leq 15$. Suhami & Mah propõem um algoritmo *branch-and-bound* heurístico que utiliza um parâmetro empírico que ajusta o

valor do limitante inferior. Para problemas com $m = 3$ e $6 \leq n \leq 10$, este algoritmo usualmente encontra soluções a 1% do valor ótimo.

O algoritmo *branch-and-bound* proposto por Ronconi & Armentano (2001) aborda o ambiente *flowshop* com buffer zero para minimizar o atraso total e é baseado no algoritmo proposto por Kim (1995) para o ambiente *flowshop* com buffer ilimitado. Cada nó da árvore *branch-and-bound* (b&b) corresponde a um subproblema, que é definido pela subseqüência de tarefas que são fixadas no final da seqüência completa. Esta subseqüência é denominada seqüência parcial. Quando um nó é ramificado, um ou mais nós são gerados pela adição de mais uma tarefa na frente da seqüência parcial associada ao nó ramificado (veja Figura 3). A seleção do nó para ramificar segue a regra de profundidade, onde o nó com mais tarefas na seqüência parcial é selecionado. No caso de empate, o algoritmo seleciona o nó com o menor limitante inferior. Um nó é sondado se seu limitante inferior é maior que a atual melhor solução encontrada (*incumbent solution*).

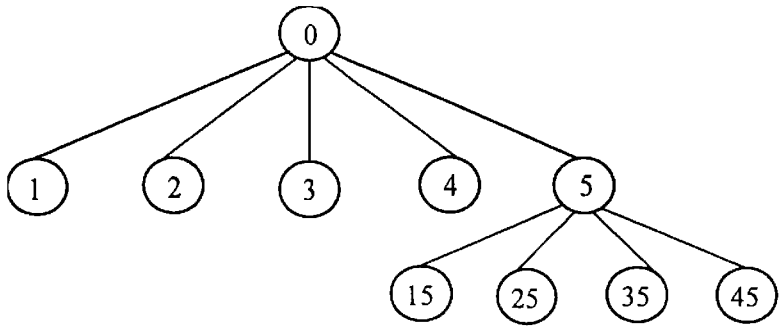

Figura 3. Exemplo da árvore parcial gerada pelo b&b para um problema com 5 tarefas.

Para cada nó gerado um limitante inferior da soma dos atrasos de todas as tarefas é calculado como a soma de dois limitantes inferiores. Um limitante inferior corresponde ao atraso das tarefas na seqüência parcial (PS) e o outro está associado com as tarefas não incluídas na seqüência parcial (NPS). A solução inicial é fornecida pela heurística baseada em Busca Tabu proposta por Armentano & Ronconi (2000).

Limitantes Inferiores

O limitante inferior do atraso total para as tarefas não incluídas na seqüência parcial (NPS) é obtido através do cálculo do limitante inferior do instante de término de cada tarefa na última máquina. Uma das formas de calcular o instante de término das tarefas é através

de um grafo direcionado (Pinedo, 1995) onde os nós (i,k) significam o instante de partida da tarefa i de cada máquina k, os valores nos arcos horizontais representam o tempo de processamento de cada tarefa em cada máquina, e os arcos inclinados representam o possível bloqueio em uma máquina causado por uma tarefa na máquina posterior. O sentido dos arcos indica a precedência necessária para cada instante de liberação. A Figura 4 apresenta este grafo para m máquinas e a seqüência de tarefas $1, 2,...,i,...,n$.

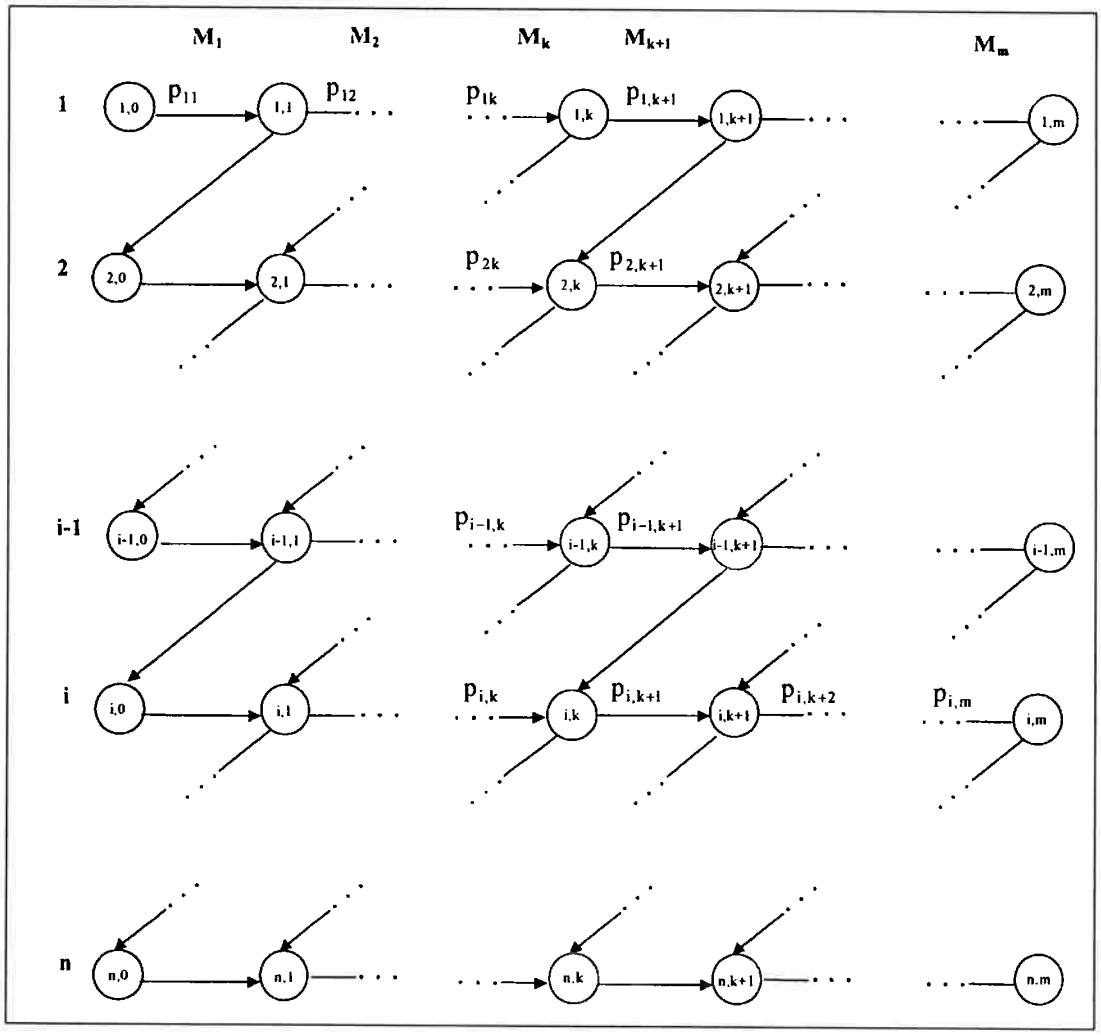

Figura 4. Representação do ambiente flowshop com bloqueio através de um grafo direcionado.

O limitante inferior proposto para as tarefas em NPS foi inspirado neste grafo e é apresentado na proposição a seguir.

Proposição 1. Considere uma seqüência arbitrária de tarefas em NPS. O instante de término da i-ésima tarefa na máquina m, C_{im}, é maior ou igual a $LC2_i$, onde:

$$LC\,2_1 = \sum_{q=1}^{m} p_q^1 ,$$

$$LC\,2_i\,(k) = \sum_{q=1}^{m} p_q^1 + \sum_{g=2}^{i} max(\ p_k^g , p_{k+1}^g),\ i \neq 1,$$

$$LC\,2_i = \max_{1 \le k \le m} \{\ LC\,2_i(k)\} = \max_{1 \le k \le m} \left\{ \sum_{q=1}^{m} p_q^1 + \sum_{g=2}^{i} max(\ p_k^g , p_{k+1}^g) \right\}\ i \neq 1,$$

onde, p_k^g representa o g-ésimo menor tempo de processamento na máquina k dentre todas as tarefas em NPS, e $p_{m+1}^g = 0$ para $g = 1,\ldots, i$.

A prova desta proposição está detalhada em Ronconi & Armentano (2001), assim como a prova das Proposições 2 e 3 que sustentam as definições dos limitantes inferiores da soma dos atrasos das tarefas em NPS e PS, respectivamente.

Proposição 2: No ambiente *flowshop* com bloqueio, a soma dos atrasos de todas as tarefas em NPS não é menor que:

$$\sum_{i=1}^{|NPS|} (\ LC\,2_i - d_{s_1(i)}\ )^+ ,$$

onde S_1 é a seqüência na qual as tarefas $\in$ NPS estão ordenadas de acordo com a regra de prioridades *earliest due date* (EDD), $d_{s_1(i)}$ é a data de entrega da i-ésima tarefa da seqüência S_1, |NPS| é a cardinalidade de NPS e $x^+ = \max(x,0)$.

O limitante inferior para o atraso total das tarefas na seqüência parcial fixada no final da seqüência completa (PS) é obtido calculando-se o limitante inferior do instante de partida da última tarefa em NPS de cada máquina (Proposição 3).

Proposição 3: O limitante inferior do instante de partida da última tarefa em NPS de uma máquina k é dado por:

$$LD_k = max\ (LD_{q_1},\ LD_{q_2}),\ \text{onde:}$$

$$LD_{q_1} = \sum_{q=1}^{k} p_q^1 + \max_{1 \le q \le k-1} \sum_{g=2}^{|NSP|} max(\ p_q^g , p_{q+1}^g),$$

$$LD_{q_2} = \sum_{q=1}^{k} p_q^1 + \sum_{g=1}^{|NSP|-1} max(\ p_k^{g+1} , p_{k+1}^g),$$

$$p_{m+1}^g = 0\ para\ g = 1,\ldots, |NPS| - 1.$$

O limitante inferior LD_k é utilizado para calcular o limitante inferior do instante de partida de cada tarefa em PS como segue:

$$D_{i0} = LD_1 \qquad i = |NPS| + 1,$$

$$D_{ik} = max(\ D_{i,k-1} + p_{ik},\ LD_{k+1}) \qquad i = |NPS| + 1,\ k = 1,\ldots,m-1,$$

$$D_{i0} = D_{i-1,1} \qquad i = |NPS| + 2,\ \ldots,n,$$

$$D_{ik} = max(D_{i,k-1} + p_{ik},\ D_{i-1,k+1}) \qquad i = |NPS| + 2,\ \ldots,n\ ;\ k = 1,\ldots,m-1,$$

$$D_{im} = D_{i,m-1} + p_{im} \qquad i = |NPS| + 1,\ \ldots,n\ .$$

O limitante inferior de partida de cada tarefa em PS da máquina m e sua respectiva data de entrega fornecem um limitante inferior de seu atraso.

Experimentos Computacionais

O algoritmo b&b foi aplicado em 540 problemas. Para cada uma das vinte e sete dimensões do problema (Tabela 2) cinco diferentes matrizes de tempos de processamento foram geradas e, para cada uma destas foram selecionados quatro cenários. O tempo de processamento é uniformemente distribuído entre 1 e 99. Cada cenário representa uma diferente geração de datas de entrega, que são uniformemente distribuídas entre $P(1 - T - R/2)$ e $P(1 - T + R/2)$ (Potts & Van Wassenhove, 1982), onde T e R representam o fator de atraso e a faixa de dispersão, respectivamente, e P é um limitante inferior do *makespan* no ambiente *flowshop* com buffer ilimitado (Taillard, 1993) definido por:

$$P = max\left\{ max_{1 \le k \le m} \left\{ \sum_{i=1}^{n} p_{ik} + min_i \sum_{q=1}^{k-1} p_{iq} + min_i \sum_{q=k+1}^{m} p_{iq} \right\}, max_i \sum_{k=1}^{m} p_{ik} \right\}.$$

Através da variação de T e R foram obtidos os seguintes cenários:

- ***cenário 1:*** baixo fator de atraso (T=0,2) e pequena faixa de datas de entrega(R=0,6)
- ***cenário 2***: baixo fator de atraso (T=0,2) e ampla faixa de datas de entrega (R=1,2)
- ***cenário 3***: alto fator de atraso (T=0,4) e pequena faixa de datas de entrega (R=0,6)
- ***cenário 4***: alto fator de atraso (T=0,4) e ampla faixa de datas de entrega (R=1,2)

Os programas computacionais foram escritos em linguagem C e os testes realizados em uma estação de trabalho SUN ULTRA com 360 MHz de *clock*, 512 Mb RAM. A tolerância utilizada foi de 0,5%. A Tabela 2 apresenta o tempo médio de execução do

algoritmo para cada cenário e a porcentagem de nós analisados. Esta porcentagem é a razão entre o número de nós gerados pelo algoritmo b&b e o número de nós da árvore gerada por uma enumeração completa. O tempo de execução para cada problema foi limitado em 3600 segundos.

A Tabela 2 mostra que os problemas pertencentes aos cenários 1 e 2 possuem um tempo de execução menor que os tempos apresentados para os problemas dos cenários 3 e 4. Uma análise, através de uma enumeração completa, de todas as possíveis soluções foi realizada em problemas com 9 tarefas e 10 máquinas. Observou-se que a maioria das soluções dos cenários 1 e 2 era de baixa qualidade. Com base neste estudo, acredita-se que o algoritmo b&b sonde mais rapidamente os nós para estes cenários e conseqüentemente apresente um tempo de execução menor. Nota-se também que o número de tarefas possui um impacto maior no tempo computacional quando comparado com o número de máquinas. Este resultado era previsível dado que o número de tarefas está relacionado com o número de nós e com o tempo de cálculo em cada nó. O número de máquinas afeta somente o tempo consumido em cada nó.

Analisando a Tabela 2 podemos observar que, na maioria dos casos, a porcentagem de nós examinados pelo b&b decresce quando a dimensão do problema aumenta. Este fato indica que os limitantes tiveram um bom desempenho oferecendo uma redução significativa no número de nós do problema. Tal fato era esperado dado que os limitantes propostos exploram características específicas do problema abordado, como a capacidade nula de armazenagem intermediária. Ronconi & Armentano (2001) apresentam maiores detalhes sobre este estudo.

Tabela 2. Resultados do algoritmo b&b para o critério do atraso total.

Dimensão		Nº de problemas não resolvidos	Tempo médio de CPU (segundos)				Média por dimensão	% de nós analisados
n	*m*		Cenário 1	Cenário 2	Cenário 3	Cenário 4		
10	2	0	0,00	0,08	0,01	0,05	0,04	6,50E-02
10	3	0	0,05	0,54	0,35	0,78	0,43	6,36E-01
10	4	0	0,13	0,40	0,43	0,81	0,44	5,00E-01
10	5	0	0,30	0,59	1,93	1,59	1,10	1,08E+00
10	7	0	1,02	1,71	7,15	6,26	4,04	3,20E+00
10	10	0	5,53	6,22	20,72	19,47	12,99	7,79E+00
12	2	0	0,02	0,09	0,08	6,59	1,70	2,12E-02
12	3	0	0,63	3,94	0,99	1,75	1,83	1,69E-02
12	4	0	1,01	5,44	19,04	33,25	14,69	1,15E-01
12	5	0	4,97	11,72	55,66	85,23	39,40	2,59E-01
12	7	0	32,26	60,50	134,22	135,19	90,54	4,69E-01
12	10	0	134,03	188,84	487,71	1444,49	563,77	2,26E+00
14	2	0	0,02	0,28	0,06	18,70	4,77	2,89E-04
14	3	0	0,89	1,52	13,53	124,46	35,10	1,63E-03
14	4	0	13,02	6,47	121,34	671,85	203,17	7,45E-03
14	5	2	62,44	165,18	1273,53	1345,46	711,65	2,30E-02
14	7	7	1036,56	1095,04	2114,77	3600,00	1961,59	4,97E-02
14	10	16	2686,27	2641,08	3600,00	3600,00	3131,84	5,97E-02
16	2	0	0,03	0,00	0,57	8,87	2,37	5,05E-07
16	3	1	3,79	742,30	47,71	802,29	399,02	6,94E-05
16	4	1	243,45	203,34	606,14	2244,49	824,36	1,09E-04
18	2	1	0,27	42,98	5,95	819,60	217,20	1,71E-07
18	3	4	49,38	737,38	483,24	2488,58	939,65	4,71E-07
18	4	6	224,46	1251,65	1226,67	3456,76	1539,89	6,09E-07
20	2	2	0,04	761,74	2,29	1563,08	581,79	9,31E-10
20	3	5	290,57	720,01	758,75	3245,78	1253,78	1,41E-09
20	4	9	1210,42	83,75	2880,77	3600,00	1943,74	1,86E-09
Média por cenário			222,28	323,44	513,47	1086,13		

2.2.2. Minimizando o *Makespan*

Devido à escassez de trabalhos no ambiente *flowshop* com estoque intermediário zero, Ronconi (2005) aborda este ambiente considerando a minimização do *makespan*. Neste trabalho um novo limitante específico para este critério de desempenho é proposto.

No algoritmo b&b desenvolvido cada nó da árvore corresponde a um subproblema, que é definido pela subseqüência de tarefas que são fixadas no início da seqüência completa. Esta subseqüência é denominada seqüência parcial, PS, e o conjunto de tarefas não incluídas em PS é denominado NPS. Quando um nó é ramificado, um ou mais nós são gerados pela adição de mais uma tarefa na frente da seqüência parcial associada ao nó ramificado. A seleção do nó para ramificar segue a regra de profundidade, no caso de empate, o algoritmo seleciona o nó com o menor limitante inferior. Um nó é sondado se seu limitante inferior é maior que a atual melhor solução encontrada.

Limitante Inferior

O limitante inferior do *makespan* é obtido através do cálculo do instante de término da última tarefa em NPS na última máquina. Este limitante requer a avaliação do instante de partida da última tarefa em PS de cada máquina. Estes instantes podem ser obtidos utilizando as equações recursivas descritas a seguir.

Seja $\pi_1, \pi_2, \ldots, \pi_i, \ldots, \pi_n$ a seqüência de tarefas, onde π_i representa a i-ésima tarefa na seqüência considerada, $M_1, M_2, \ldots, M_k, \ldots, M_m$ a seqüência ordenada de máquinas, p_{ik} o tempo de processamento da tarefa π_i na máquina k. Seja D_{ik} o instante de partida da tarefa π_i na máquina k. Claramente, $D_{ik} \geq C_{ik}$ onde C_{ik} é o instante de término da tarefa π_i na máquina k. O instante de tempo no qual a tarefa π_i inicia o seu processamento na primeira máquina é representado por D_{i0}. Os instantes de partida são dados pelas seguintes equações:

$$D_{10} = 0$$

$$D_{1k} = \sum_{q=1}^{k} p_{1q} \qquad k = 1,\ldots,m\text{-}1,$$

$$D_{i0} = D_{i-1,1} \qquad i = 2,\ldots\ldots,n,$$

$$D_{ik} = max(D_{i,k-1} + p_{ik}, D_{i-1,k+1}) \qquad i = 2,\ldots\ldots,n; k = 1,\ldots,m\text{-}1,$$

$$D_{im} = D_{i,m-1} + p_{im} \qquad i = 1,\ldots\ldots,n.$$

Na recursão acima, primeiro são calculados os instantes de partida da primeira tarefa, em seguida os da segunda, até a última tarefa. Dado que $C_{im} = D_{im}$, o *makespan* (C_{max}) é dado por D_{nm}.

O conceito do limitante inferior para as tarefas em NPS é mais facilmente compreendido utilizando o grafo descrito na Seção 2.2.1. O instante de término da última tarefa na máquina m (C_{max}) é dado pelo caminho mais longo entre os nós (1,0) e (n, m). Então, o tamanho de um caminho qualquer que una estes dois nós fornece um limitante inferior deste instante de tempo.

Para uma dada máquina k, um conjunto de tarefas em PS e utilizando uma seqüência conhecida de tarefas em NPS, considere o caminho cujo comprimento é dado pela seguinte expressão:

$$L1(k) = max(\ D_{|PS|,k} + p_{|PS|+1,k}\ ;\ D_{|PS|,k+1}\ ) + \sum_{r=|PS|+1}^{n-1} max(\ p_{r+1,k}\ ,\ p_{r,k+1}\ ) + \sum_{q=k+1}^{m} p_{nq} \qquad (41)$$

onde |PS| é a cardinalidade de PS.

O primeiro termo da equação (41) corresponde ao cálculo do instante de partida da tarefa na posição |PS|+1, $\pi_{|PS|+1}$, da máquina k, *i.e.*, $D_{|PS|+1,k}$. Este termo adicionado ao primeiro máximo da somatória seguinte representa um limitante inferior do instante de partida da tarefa na posição $|PS|+2$, $\pi_{|PS|+2}$, da máquina k, este valor mais o segundo máximo representa um limitante inferior do instante de partida da tarefa na posição $|PS|+3$, $\pi_{|PS|+3}$, da máquina k e assim por diante. O último termo é o tempo despendido pela tarefa n nas máquinas $q>k$. Logo, um limitante inferior do instante de término da tarefa π_n na máquina m pode ser descrito como:

$$LB1 = \max_{1 \le k \le m} \{ L1(k) \}$$

onde $p_{r,m+1} = 0$ e $D_{r,m+1} = 0$ para todo r.

O limitante inferior LB1 é válido para uma dada seqüência de tarefas. A proposição a seguir apresenta um limitante inferior para as tarefas em NPS quando não se conhece a seqüência.

Proposição 4. Considere uma seqüência arbitrária em NPS com |NPS| tarefas. O instante de término da última tarefa na máquina m é maior ou igual a LB2, onde:

$$L2(k) = D_{|PS|,k} + \sum_{g=1}^{|NPS|} \max(\ a_k^g\ ,\ b_{k+1}^g\ ) + \sum_{q=k+1}^{m} p_{\min_q}$$

$$LB2 = \max_{1 \le k \le m} \{ L2(k) \}$$

$p_{\min_k}$: menor tempo de processamento na máquina k dentre todas as tarefas em NPS ,

a_k^g : g-ésimo menor tempo de processamento na máquina k dentre todas as tarefas em NPS,

b_{k+1}^g : g-ésimo menor valor dentre $D_{|PS|,k+1} - D_{|PS|,k}$ e os tempos de processamento das tarefas em NPS na máquina k+1 sem $p_{\min_{k+1}}$ e $b_{m+1}^g = 0$ para todo g.

Partindo do fato de que LB1 é um limitante inferior do instante de término das tarefas em NPS, Ronconi (2005) comprova que LB2 é sempre menor ou igual a LB1, ou seja, que a Proposição 4 é válida.

Experimentos Computacionais

O algoritmo b&b foi aplicado em 540 problemas. Para cada uma das vinte e sete dimensões do problema (Tabela 3) vinte diferentes matrizes de tempos de processamento foram geradas. O tempo de processamento foi uniformemente distribuído entre 1 e 99. A tolerância utilizada foi de 0,5%. Os programas computacionais foram escritos em linguagem C e os testes realizados em um Pentium IV com 1.4 GHz de clock, 512 Mb RAM. A solução inicial foi fornecida pelo algoritmo NEH proposto por Nawaz *et al.* (1983). O tempo de execução para cada problema foi limitado em 3600 segundos.

Para avaliar o algoritmo b&b proposto ($b\&b_{prop}$) o mesmo conjunto de problemas foi resolvido através do algoritmo b&b proposto por Ronconi & Armentano (2001) ($b\&b_{RA}$). Estes autores além de apresentarem um limitante inferior para o atraso total, propõem uma adaptação deste limitante que fornece um limitante para o *makespan*. A Tabela 3 resume os resultados obtidos pelos algoritmos b&b.

Observa-se que, na maioria dos casos, o número de nós analisados pelo algoritmo $b\&b_{prop}$ é menor que o número de nós examinados pelo algoritmo $b\&b_{RA}$. Além disso, o método proposto é mais rápido que o método $b\&b_{RA}$. O algoritmo proposto forneceu melhores resultados em 85,2% dos casos, além de resolver um maior número de problemas, 526, enquanto o algoritmo $b\&b_{RA}$ resolveu 513. Este desempenho era esperado dado que o limitante inferior proposto é especifico para o critério do *makespan* enquanto o limitante utilizado pelo algoritmo $b\&b_{RA}$ é uma adaptação do limitante inferior proposto para o critério do atraso total.

Tabela 3. Desempenho dos algoritmos b&b para o critério do makespan.

Dimensão		$b\&b_{prop}$			$b\&b_{RA}$		
n	*m*	Nº de problemas não resolvidos	Tempo médio de CPU (segundos)	Nº médio de nós analisados	Nº de problemas não resolvidos	Tempo médio de CPU (segundos)	Nº médio de nós analisados
10	2	0	0,00	74,60	0	0,00	1884,15
10	3	0	0,00	2354,20	0	0,01	6717,60
10	4	0	0,01	8009,05	0	0,03	10937,50
10	5	0	0,02	15006,30	0	0,11	37954,60
10	7	0	0,06	22061,45	0	0,10	24835,90
10	10	0	0,08	24152,10	0	0,26	48852,55
12	2	0	0,00	205,95	0	0,11	94292,45
12	3	0	0,09	68257,50	0	0,10	62157,25
12	4	0	0,38	237378,09	0	0,81	374809,50
12	5	0	0,85	405272,69	0	0,77	263241,25
12	7	0	2,00	668053,06	0	1,99	496700,94
12	10	0	7,24	1747112,75	0	8,42	1275805,88
14	2	0	0,00	7155,40	0	0,43	258602,00
14	3	0	0,36	266459,81	0	29,40	8754992,00
14	4	0	7,71	4195833,50	0	8,45	3106239,50
14	5	0	19,60	8563077,00	0	27,36	6990951,00
14	7	0	19,38	6010332,50	0	129,14	29172214,00
14	10	0	57,75	12050290,00	0	136,20	22245622,00
16	2	0	0,00	418,40	0	3,91	2550440,00
16	3	0	6,36	4349962,50	0	104,13	51616312,00
16	4	1	315,62[a]	144089152,00	0	188,04	67812464,00
18	2	0	1,24	1069994,75	0	20,47	13071317,00
18	3	0	108,24	68076352,00	0	390,46	173675936,00
18	4	3	946,84[a]	443388768,00	6	1332,63[a]	450607040,00
20	2	0	0,01	3812,50	1	286,73[a]	173655712,00
20	3	1	446,44[a]	260417072,00	9	1873,51[a]	460729600,00
20	4	9	2518,48[a]	851694720,00	11	2608,53[a]	772345216,00

[a] Estes são limitantes inferiores dos valores médios pois existem problemas não resolvidos nestes conjuntos.

2.3. Heurísticas e Meta-Heurísticas

Existem vários problemas de otimização combinatória que são de difícil resolução através de métodos que garantidamente fornecem a solução ótima, nestes casos, métodos heurísticos são geralmente empregados para encontrar rapidamente soluções não necessariamente ótimas, porém de boa qualidade.

De acordo com Zanakis *et al.* (1989) a área de produção, especialmente na programação de tarefas, é uma das áreas que mais utiliza métodos heurísticos. Existem diferentes tipos de heurísticas, segundo o modo como buscam e constroem suas soluções. Uma heurística com largas aplicações é a heurística construtiva. Este método constrói uma solução passo a passo de forma tal que a cada passo se adicionam componentes individuais. Os algoritmos mais populares dentro desta categoria são os "gulosos" (*greedy*) que se caracterizam por buscar o máximo benefício em cada passo. O pequeno esforço computacional desta estratégia, que é um fator valioso em aplicações práticas, é uma das razões que motiva o seu estudo. Vale destacar que, diferentemente de outras técnicas mais elaboradas como as meta-heurísticas, heurísticas construtivas são geralmente mais simples e fáceis de implementar e não requerem testes computacionais exaustivos para o seu ajuste.

Outra heurística popular com largas aplicações é a busca local ou busca em vizinhança. A idéia básica é muito simples: inicia-se de uma solução qualquer e busca-se em uma certa vizinhança a melhor solução possível. A partir desta, o processo de busca recomeça e prossegue até que nenhuma melhoria seja obtida. A solução final é chamada ótimo local, dado que esta é a melhor solução dentre toda uma vizinhança. A desvantagem evidente deste método é que este ótimo local na maioria das vezes não é o ótimo global. Buscando evitar que a heurística estacione nestes ótimos locais surgem as meta-heurísticas, que trabalham como guias da busca local direcionando seus movimentos para a exploração de diferentes regiões do espaço de soluções. Atualmente as meta-heurísticas mais conhecidas por seu bom desempenho são Busca Tabu, *Simulated Annealing*, Algoritmos Genéticos e *Greedy Ramdomized Adaptive Search Procedures* (Adenso-Díaz *et al.*, 1996). Além disso, as meta-heurísticas híbridas também tem recebido considerável atenção no

campo de otimização combinatória (veja Talbi, 2002). A estratégia híbrida objetiva criar uma técnica de busca alternativa incorporando as melhores características de cada método que a compõe. O intuito é melhorar a eficiência dos métodos ou reduzindo o tempo computacional e/ou melhorando a qualidade das soluções encontradas.

Neste tópico descreveremos trabalhos que introduzem heurísticas e meta-heurísticas para diferentes ambientes de programação de tarefas. Hino *et al.* (2005) consideram máquina única minimizando a soma ponderada de atrasos e adiantamentos. Para o ambiente *flowshop* Ronconi (2004) e Ronconi & Henriques (2005) exploram a ausência de estoque intermediário entre as máquinas (buffer zero). Ronconi & Henriques (2005) tratam deste ambiente minimizando o atraso total, enquanto Ronconi (2004) apresenta heurísticas construtivas para a minimização do *makespan*. Sakuraba & Ronconi (2006) também consideram este ambiente, porém minimizando a soma dos atrasos e adiantamentos e, além de heurísticas, propõem um algoritmo que, dada uma seqüência, encontra a programação ótima da mesma.

2.3.1. Máquina Única Minimizando o Adiantamento e Atraso das Tarefas

Hino *et al.* (2005) consideram o problema de programação de n tarefas em uma única máquina. Cada tarefa j está disponível no instante inicial zero e possui um tempo de processamento p_j. As tarefas têm uma data de entrega comum e conhecida d. A medida de desempenho considerada é a minimização da soma ponderada dos adiantamentos e atrasos das tarefas. O estudo de modelos contendo penalidades por antecipação e atraso tem tido uma importância significativa em função do conceito *Just In Time*, onde o cliente não quer receber o seu produto antes da data desejada, de tal forma a evitar a formação de estoques, e nem após a data de entrega, para não penalizar a utilização deste produto nos elos subseqüentes da cadeia produtiva.

Este tipo de critério de otimização é conhecido por ser não regular, ou seja, não é uma função crescente do instante de término das tarefas. O atraso de uma tarefa é dado por $T_j = max\ (0\ ,\ C_j - d)$ e a antecipação é definida por $E_i = max\ (0\ ,\ d - C_j)$, onde C_j é o instante

de término da tarefa. O objetivo é achar uma seqüência (S) que minimize a seguinte expressão:

$$f(S) = \sum_{j=1}^{n} \left(\alpha_j E_j + \beta_j T_j \right),$$

onde α_j e β_j são as penalidades positivas por concluir cada tarefa i antes e depois da data de entrega, respectivamente. A Figura 5 representa esquematicamente o problema:

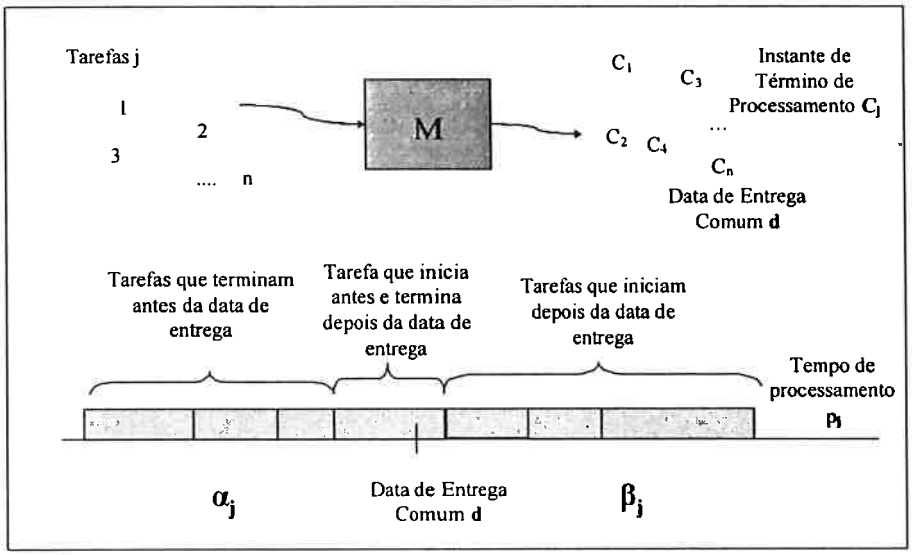

Figura 5. Representação esquemática do problema.

Existem duas abordagens principais para a data de entrega comum. No caso irrestrito, a data de entrega comum é uma variável de decisão ou, se o seu valor é conhecido, ele não possui influência na seqüência ótima. Tal fato ocorre quando a data de entrega é maior ou igual que a soma de todos os tempos de processamento. Porém, se a data é conhecida e influencia a seqüência ótima das tarefas, a mesma é considerada restrita. Este problema, com data de entrega restrita, possui propriedades que auxiliam na geração de soluções de alta qualidade, especificamente, pode ser provado que sempre existe uma solução ótima com as seguintes propriedades:

(1) Não existe inserção de ociosidade dentro do programa gerado (Cheng & Kahlllbacher 1991).

(2) O programa tem a forma de V (V-Shaped), isto é, as tarefas finalizadas antes ou na data de entrega são ordenadas em uma ordem não-crescente da razão p_i/α_i e as tarefas que iniciam o seu processamento após ou na data de entrega estão ordenadas em uma ordem não-decrescente da razão p_i/β_i. Note que um "straddling job" pode existir, i.e., uma tarefa cujo processamento é iniciado antes e terminado depois da data de entrega (Biskup & Feldman 2001).

(3) Uma tarefa é completada na data de entrega ou a primeira tarefa inicia o seu processamento no instante zero (Biskup & Feldman 2001).

Diversos autores abordaram problemas de programação de tarefas envolvendo o adiantamento e o atraso de tarefas (E/T - *Earliness/Tardiness*). Uma ampla revisão bibliográfica sobre a determinação da data de entrega e a programação das tarefas pode ser encontrada em Gordon *et al.* (2002). Baker & Scuder (1990) além de revisarem a literatura, destacaram que o problema de E/T com uma máquina, penalidades dependentes das tarefas e data comum restrita não havia sido tratado na literatura até o momento. Na mesma época, Hall *et al.*(1991) comprovaram a alta complexidade do problema demonstrando que o mesmo é *NP-hard*.

Devido à sua complexidade, diversos autores abordaram este problema com técnicas heurísticas, principalmente utilizando meta-heurísticas. Lee & Kim (1995) apresentaram um algoritmo genético paralelo, enquanto James (1997) desenvolveu um algoritmo baseado na meta-heurística Busca Tabu. Estes autores, embora utilizem técnicas distintas, utilizam uma prerrogativa comum: os programas propostos começam no instante zero. Observe que a Propriedade (3) estabelece que, no programa ótimo, não obrigatoriamente o processamento das tarefas deve se iniciar no instante zero. Logo não considerar a inserção de ociosidade no início do programa pode acarretar a exclusão do programa ótimo.

Biskup & Feldman (2001) estudaram este problema considerando a possibilidade de adiar o início do programa. Estes autores apresentaram duas heurísticas para o problema, com resultados satisfatórios para problemas de pequeno porte. Como contribuição significativa desse trabalho, ressalta-se a geração de 280 problemas, variando entre 10 e 1000 tarefas, e dos resultados obtidos com as heurísticas elaboradas, que se tornaram

referência para estudos posteriores. Posteriormente, Feldman & Biskup (2003) apresentaram metodologias baseadas nas seguintes meta-heurísticas: *Evolutionary Search* (ES), *Simulated Anneling* (SA) e *Threshold Accepting* (TA). Em um estudo comparativo com o método proposto por James observou-se que o algoritmo baseado em TA obteve os melhores resultados na maioria dos problemas avaliados, tanto em tempo computacional quanto em qualidade da solução.

Hino *et al.* (2005) com o intuito de contribuir com novos métodos para a resolução do problema de E/T com data restrita propuseram uma heurística construtiva, heurísticas baseadas em Busca Tabu e Algoritmos Genéticos e métodos híbridos. Os procedimentos propostos, que estão detalhados a seguir, exploram as três propriedades descritas na Seção 2.2.3, inclusive a possível inserção de tempo de espera no início da seqüência.

Heurística Construtiva

Dado que no máximo uma tarefa será finalizada na data de entrega, existirá um conjunto de tarefas (S^E) finalizadas antes da data de entrega e um conjunto de tarefas (S^T) finalizadas após a data de entrega. A estratégia heurística proposta, doravante denominada HRM, consiste em (*i*) determinar estes dois conjuntos, (*ii*) construir uma seqüência para cada um deles e (*iii*) estabelecer a programação final das tarefas (S) como a concatenação das duas seqüências. Para garantir que S satisfaça as Propriedades (1) e (2), não existirá tempo ocioso entre tarefas consecutivas e as seqüências S^E e S^T estarão no formato "\-shaped" e "/-shaped", respectivamente. A construção dos conjuntos S^E e S^T seguirá a seguinte estratégia. Em cada iteração, as tarefas ainda não programadas com as maiores taxas p_j/α_j e p_j/β_j serão avaliadas para inclusão em cada um dos dois conjuntos. De acordo com as distâncias entre o término das tarefas candidatas e a data de entrega, somente uma será incluída. Ajustes no tempo ocioso no começo do programa serão considerados. Finalmente, quando todas as tarefas estiverem programadas e, se conveniente, o tempo ocioso inicial será ajustado de acordo com a Propriedade (3). Descrevem-se a seguir os principais passos da heurística:

<u>Notação:</u>

P: *conjunto de tarefas não alocadas;*

g: *tempo ocioso no início do programa;*

S^E: *conjunto de tarefas finalizadas antes ou na data de entrega;*

S^T: *conjunto de tarefas finalizadas depois da data de entrega;*

S: *programação das tarefas,* $S = (g, S^E, S^T)$;

e: *tarefa candidata à inclusão em* S^E;

t: *tarefa candidata à inclusão em* S^T;

E^e: *distância entre o possível término da tarefa e e a data de entrega;*

T^t: *distância entre o possível término da tarefa t e a data de entrega;*

d^T: *janela de tempo disponível para a inserção de uma tarefa no conjunto* S^T;

d^E: *janela de tempo disponível para a inserção de uma tarefa no conjunto* S^E;

H: *tempo total de processamento, i.e.,* $H = \sum_{j=1}^{n} p_j$.

Passo 1: *Seja* $P = \{1, 2,..., n\}$, $S^E = S^T = \{\varnothing\}$, $g = max\ \{0, d - 0.5H\}$, $d^E = d - g$ *e* $d^T = g + H - d$.

Passo2: *Seja* $e = \underset{j \in P}{argmax}\{p_j / \alpha_j\}$ *e* $t = \underset{j \in P}{arg\ max}\{p_j / \beta_j\}$ *(em caso de empate selecione a tarefa com o maior tempo de processamento).*

Passo 3: *Seja* $E^e = d^E - p_e$ *e* $T^t = d^T$. *Se* $E^e < 0$, *vá para o Passo 5. Se* $T^t \leq 0$, *vá para o Passo 6.*

Passo 4: *Selecione a tarefa a ser inserida:*

Se $E^e > T^t$ *então* $S^E = S^E + \{e\}$, $d^E = d^E - p_e$ *e* $P = P - \{e\}$.

Se $E^e < T^t$ *então* $S^T = S^T + \{t\}$, $d^T = d^T - p_t$ *e* $P = P - \{t\}$.

Se $E^e = T^t$ *então Se* $\alpha_e > \beta_t$ *então* $S^T = S^T + \{t\}$, $d^T = d^T - p_t$ *e* $P = P - \{t\}$.

senão $S^E = S^E + \{e\}$, $d^E = d^E - p_e$ *e* $P = P - \{e\}$.

Vá para o Passo 7.

Passo 5: *Ajuste do tempo ocioso (fim do espaço antes da data de entrega):*

Se $(g + E_e) < 0$ *então:* $S^T = S^T + \{t\}$, $d^T = d^T - p_t$ *e* $P = P - \{t\}$.

senão: $S^{E\prime} = S^E$, $S^{T\prime} = S^T \cup P$, $g' = d - \sum_{i \in S^{E\prime}} p_i$, $S' = (g', S^{E\prime}, S^{T\prime})$;

$S^{E\prime\prime} = S^E + \{e\}$, $S^{T\prime\prime} = (S^T \cup P) - \{e\}$, $g'' = d - \sum_{i \in S^{E\prime\prime}} p_i$ *e* $S'' = (g'', S^{E\prime\prime}, S^{T\prime\prime})$;

Se $f(S') \leq f(S'')$ *então* $S^T = S^T + \{t\}$, $g = g'$, $d^T = d^T - p_t$ *e* $P = P - \{t\}$.

senão $S^E = S^E + \{e\}$, $d^E = 0$, $d^T = d^T + E_e$, $g = g''$ *e* $P = P - \{e\}$.

Vá para o Passo 7.

Passo 6: *Ajuste do tempo ocioso (fim do espaço depois da data de entrega):*

$S^{E\prime} = S^E \cup P$, $S^{T\prime} = S^T$, $g' = d - \sum_{i \in S^{E\prime}} p_i$, $S' = (g', S^{E\prime}, S^{T\prime})$;

$S^{E}{}'' = (S^{E} \cup P) - \{t\}$, $S^{T}{}'' = S^{T} + \{t\}$, $g'' = d - \sum_{i \in S^{E}{}''} p_i$ e $S'' = (g'', S^{E}{}'', S^{T}{}'')$;

Se $f(S') \leq f(S'')$ *então* $S^{E} = S^{E} + \{e\}$, $g = g'$, $d^{E} = d^{E} - p_e$ e $P = P - \{e\}$.

senão $d^{E} = d^{E} + T_t - p_t$, $d^{T} = 0$, $g = g''$, $S^{T} = S^{T} + \{t\}$ e $P = P - \{t\}$.

Passo 7: *Se* $P \neq \{\emptyset\}$ *vá para Passo 2.*

Passo 8: *Se existe uma tarefa k que possui parte do seu processamento* ($p_b > 0$) *antes da data de entrega e parte depois da data de entrega* ($p_a > 0$) *então aplique o* ***teste_g***: $g' = g + p_b$, $S' = (g', S^{E}, S^{T})$; $g'' = max\{0, g - p_a\}$, $S'' = (g'', S^{E} + \{k\}, S^{T} - \{k\})$; $S = argmin\{f(S), f(S'), f(S'')\}$.

Note que a inserção de uma tarefa em S^{E} é sempre realizada no final do conjunto. Por exemplo, se a tarefa $\{2\}$ é inserida no conjunto $S^{E} = \{1, 4, 3\}$, o novo conjunto gerado será $S^{E} = \{1, 4, 3, 2\}$. Por outro lado, quando a inserção é realizada no conjunto S^{T}, a nova tarefa será alocada no início do conjunto. Além disso, todo cálculo de $f(S)$ considera a Propriedade (2). Para realizar tal avaliação é importante observar que o conjunto S^{T} pode conter um *straddling job*. Se o *straddling job* não existir todos os elementos do conjunto S^{E} são ordenados em ordem não-crescente de p_j/α_j e os elementos de S^{T} são ordenados em ordem não-decrescente de p_j/β_j. Quando existir um *straddling job* o seguinte procedimento será executado:

Passo a: *Ordene o conjunto* S^{E} *em ordem não-crescente da taxa* p_j/α_j.

Passo b: *Ordene o conjunto* S^{T} *em ordem não-decrescente da taxa* p_j/β_j. *Note que agora o "straddling job" pode ser outra tarefa ou não existir mais nenhuma tarefa nesta situação.*

Passo c: *Se existirem tarefas no conjunto* S^{T} *que agora são finalizadas antes ou na data de entrega, aloque estas tarefas para o conjunto* S^{E} *de acordo com o formato "\-shape".*

O algoritmo começa com duas cópias de P, ordenadas pelas taxas p_j/α_j e p_j/β_j, e mantém a seqüência S^{E} (S^{T}) *em ordem não-crescente* (*não-decrescente*) de p_j/α_j (p_j/β_j). Então, podemos verificar que cada iteração pode ser implementada em $O(n)$ operações e o algoritmo completo tem complexidade $O(n^2)$. Note que ao final do algoritmo uma única solução será representada pela variável g e as seqüências S^{E} e S^{T}.

Algoritmo Genético

O Algoritmo Genético (AG) foi introduzido por Holland (1975). Cada indivíduo na população representa uma solução viável para o problema tratado. A qualidade de cada solução é medida pela função de aptidão (*fitness*). A busca segue por um certo número de gerações onde a contribuição de um indivíduo para a próxima geração é proporcional a sua aptidão (Dowsland, 1996).

No trabalho de Hino *et al.* (2005), o indivíduo possui dois componentes principais: a seqüência e o tempo ocioso inserido no início do programa. Por exemplo: [1, 4, 3, 2] [6] indica que a ordem de processamento é: tarefas 1, 4, 3 e 2, com a primeira tarefa começando no instante de tempo 6. A população inicial é composta da solução fornecida pela heurística HRM e por indivíduos gerados aleatoriamente. Os operadores genéticos utilizados foram:

- *Mutação* – cada indivíduo possui uma probabilidade P_{MP} de sofrer esta mutação. O novo indivíduo possuirá a seqüência e o tempo ocioso igual ao indivíduo original, exceto por um par de tarefas trocadas de forma aleatória. Uma destas tarefas necessariamente pertence ao conjunto de tarefas que iniciam o seu processamento antes da data de entrega e a outra pertence ao conjunto de tarefas que são iniciadas após esta data.

- *Mutação do Tempo Ocioso* – cada indivíduo possui uma probabilidade P_{MI} de sofrer esta mutação. A nova solução possuirá a seqüência igual ao indivíduo original, porém seu tempo ocioso inicial será modificado segundo a seguinte expressão $g_{new} = g_{current} + 4\lfloor p_{av}(3rnd - 1)\rfloor$, onde $\lfloor x \rfloor$ é o maior inteiro menor ou igual a x, $rnd \in [0,1]$ é um valor aleatório gerado através de uma distribuição uniforme e p_{av} é a media aritmética do tempo de processamento de todas as tarefas.

- *Crossover* – cada indivíduo (denominado "pai 1") possui uma probabilidade P_{CR} de sofrer esta operação. Caso um indivíduo seja selecionado para esta operação, outro indivíduo ("pai 2") será escolhido aleatoriamente dentro da população. O novo indivíduo gerado é composto por elementos de cada pai e tempo ocioso igual ao pai 1. Por exemplo, sejam os indivíduos a seguir e os pontos de *crossover* 4 e 7:

[1, 7, 3, 4, 5, 6, 2, 8, 9, 10][12] – pai 1

[1, 2, 3, 6, 8, 5, 7, 4, 9, 10][14] – pai 2

Este procedimento obterá:

[1, 7, 3, 6, 8, 5, 7, 8, 9, 10][12]

Dado que existem tarefas que aparecem duplicadas, é necessário substituí-las por outras tarefas não consideradas. O procedimento de correção busca na primeira e na terceira subseqüências por tarefas que também compõem a segunda subseqüência, neste exemplo, as tarefas 7 e 8. Tais tarefas são substituídas por outras que nunca apareceram seguindo a ordem na qual aparecem no pai 1. Neste exemplo o método proposto geraria o seguinte indivíduo:

[1, 4, 3, 6, 8, 5, 7, 2, 9, 10][12]

Um procedimento similar para gerar um indivíduo válido é apresentado por Kozan & Preston (1999).

Após a aplicação dos operadores genéticos, as tarefas são reordenadas de acordo com a Propriedade *V-Shaped* como segue: identificamos as tarefas que são finalizadas na data de entrega ou antes (S^E) e as tarefas que são finalizadas depois da data (S^T) e aplicamos a rotina composta pelos Passos a, b e c descritos anteriormente. Outros parâmetros considerados no AG proposto são:

- *Tamanho da População* (*GenPop*) – cada geração começa com *GenPop* indivíduos. Após a aplicação dos operadores genéticos, a população terá um aumento esperado de *GenPop* (P_{MP} + P_{MI} + P_{CR}). Para restringir o seu tamanho a função de aptidão é utilizada para selecionar os *GenPop* indivíduos de maior qualidade.

- *Número de Gerações* (*Ng*) – Este parâmetro afeta a extensão do processo evolucionário. O número esperado de soluções geradas ao longo da busca pode ser estimado em *GenPop* (P_{MP} + P_{MI} + P_{CR})*Ng*.

Ao final do algoritmo aplica-se o ***teste_g*** na melhor solução encontrada para ajustar o tempo ocioso inicial de forma a obedecer a Propriedade (3).

Busca Tabu

A metodologia proposta consiste em uma busca em vizinhança orientada pela Busca Tabu (BT). Esta técnica é descrita em Glover & Laguna (1997) e objetiva guiar a busca na exploração do espaço de soluções de um problema além da otimalidade local com vistas à obtenção de soluções de qualidade superior. Uma das principais características da meta-heurística BT é memorizar atributos de soluções recentemente examinadas. Regras de proibição são associadas com estes atributos para proibir a ocorrência de alguns movimentos por um certo numero de iterações. Esta característica tem o intuito de evitar ciclos e visitar diferentes regiões do espaço de soluções.

Esta estratégia requer a definição dos seus principais componentes para a resolução do problema tratado, tais como a solução inicial, o tipo de movimento que gera a vizinhança, o método de seleção da próxima solução, o atributo a ser armazenado de cada solução, a regra de proibição, a duração tabu e os critérios de aspiração e de parada. Para determiná-los, diversos testes foram realizados. A combinação de componentes que apresentou os melhores resultados esta descrita a seguir:

- *Solução Inicial*: heurística HRM.
- *Estratégia de geração de vizinhança*: a vizinhança é gerada pela troca aleatória da tarefa j_1 finalizada antes da data de entrega com a tarefa j_2 finalizada depois desta data. Após este movimento a seqüência é reordenada de acordo com a Propriedade *V-Shaped* como segue: identificamos as tarefas que são finalizadas na data de entrega ou antes (S^E) e as tarefas que são finalizadas depois da data (S^T) e aplicamos a rotina composta pelos Passos a, b e c descritos anteriormente. O tamanho da vizinhança é definido por *SN*.
- *Atributo e regra de proibição*: as tarefas j_1 e j_2 não podem ser trocadas pelo período da duração tabu.
- *Duração Tabu:* $\lfloor \sqrt{n} \rfloor$ iterações.
- *Critério de aspiração*: um movimento tabu pode ser liberado de sua condição tabu caso gere uma solução melhor que a melhor solução encontrada até o momento;

- *Número de iterações (Nit):* este parâmetro determina o número de soluções examinadas durante toda a busca e pode ser calculado através da seguinte expressão: *Nit x SN.*

Outro componente importante da busca tabu implementada é a inserção do tempo de espera no início do programa.

- *Modificação do Tempo Ocioso:* O valor do tempo ocioso inicial é ajustado se o número de iterações sem melhoria na função objetivo (*NIWI*) é maior ou igual a *MaxIts*, como mostrado na Figura 6. Este critério busca corrigir o tempo ocioso de forma a aumentar ou diminuir o seu valor com base no benefício obtido com a última alteração deste tempo. Uma variável, denominada *IncIdle* indica que tipo de ajuste acarretou tal benefício. Note que se *IncIdle* é "verdadeiro" o tempo ocioso provavelmente será aumentado e se é "falso" o tempo ocioso provavelmente será diminuído.

Vale destacar que ao final deste algoritmo aplica-se o **teste-g** descrito na heurística HRM com o intuito de ajustar o tempo de espera do sistema de forma a considerar a Propriedade (3).

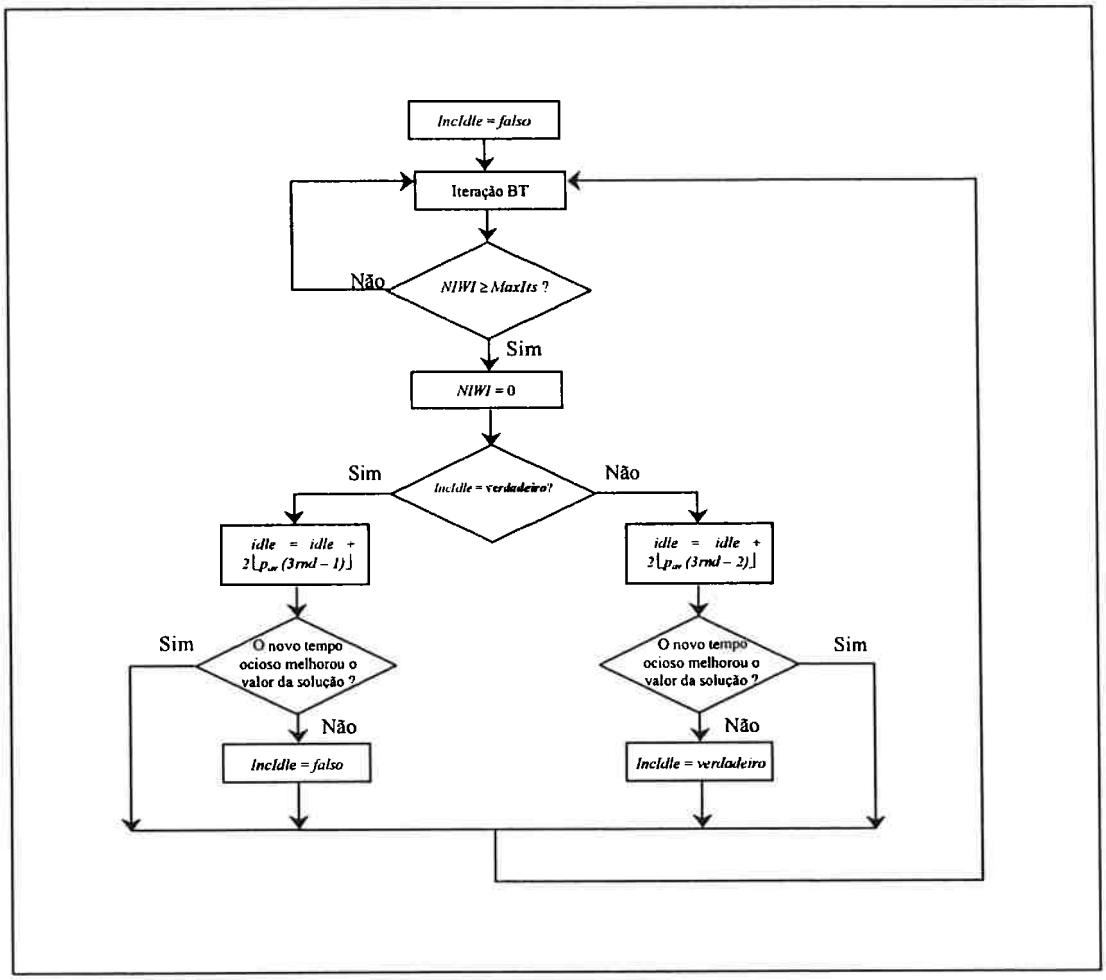

Figura 6: Critério de atualização do tempo ocioso na BT.

Métodos Híbridos

As meta-heurísticas híbridas tem recebido considerável atenção no campo da otimização combinatória (Talbi, 2002). Hino *et al.* (2005) apresentam estratégias híbridas que mesclam as meta-heurísticas BT e AG. O objetivo é gerar novos métodos que preservem as boas características de cada metodologia de forma a obter resultados superiores, em termos de tempo computacional e/ou qualidade da solução, aos obtidos por estas técnicas quando utilizadas isoladamente.

Outros autores também utilizaram abordagens híbridas, em geral com sucesso, na resolução de problemas de programação de tarefas. Por exemplo, Gonçalves *et al.* (2005)

propõem um AG híbrido para a programação de tarefas no ambiente *job shop*. Após a obtenção de um programa utilizando prioridades definidas pelo AG, uma busca local é aplicada para aprimorar a solução. Adenso-Díaz (1996) utiliza *Simulated Annealing* combinada BT para a programação de tarefas no ambiente *flowshop*, enquanto Buzzo & Moccellin (2000) apresentam, para este mesmo ambiente, uma versão híbrida utilizando *Simulated Annealing* e AG. No ambiente com máquinas paralelas idênticas Laguna & Velarde (1991) mesclam a meta-heurística GRASP com BT. Para o ambiente abordado neste tópico não foram encontradas aplicações envolvendo algoritmos híbridos.

A estrutura híbrida utilizada segue os conceitos propostos em Adenso-Díaz (1996) onde as meta-heurísticas são aplicadas de forma seqüencial, como descrito a seguir:

- *HTG*: Busca Tabu + Algoritmo Genético
  - *Solução inicial*: *HRM*;
  - *Número de iterações da Busca Tabu*: *NitH*;
  - *Número de gerações do Algoritmo Genético*: *NgH*, utilizando como população inicial a melhor solução obtida na Busca Tabu + 20 indivíduos gerados por procedimento aleatório.
- *HGT*: Algoritmo Genético + Busca Tabu
  - *População inicial*: HRM + 20 indivíduos gerados por procedimento aleatório;
  - *Número de gerações do Algoritmo Genético*: *NgH*;
  - *Número de iterações da Busca Tabu*: *NitH*, utilizando como solução inicial a melhor solução encontrada pelo Algoritmo Genético.

Experimentos Computacionais

O desempenho dos métodos propostos foi avaliado em um conjunto de problemas teste disponível no endereço eletrônico *http://people.brunel.ac.uk/~mastjjb/jeb/info.html* e publicado por Biskup & Feldmann (2001). O conjunto teste possui sete diferentes dimensões (n=10, 20, 50, 100, 200, 500, 1000) e quatro fatores restritivos (h= 0,2; 0,4; 0,6; 0,8) que definem a data de entrega comum segundo a seguinte expressão: $d = \left\lfloor h \sum_{j=1}^{n} p_j \right\rfloor$.

Existem 10 problemas para cada combinação de dimensão e fator restritivo, totalizando 7 x 4 x 10 = 280 problemas. A avaliação de cada método foi realizada considerando a diferença percentual obtida em relação ao *benchmark* publicado por Biskup & Feldmann (2001):

$$\%Dif = 100\frac{F_{ap} - F_{BF}}{F_{BF}}$$

onde F_{ap} e F_{BF} são os valores da função objetivo obtidos, respectivamente, pelo algoritmo proposto e o apresentado por Biskup & Feldmann.

Os programas computacionais foram codificados utilizando Borland Delphi 5, em um Pentium IV com 1.7 GHz de *clock* e 512Mb RAM. Após testes preliminares com um número reduzido de problemas com diversas combinações de parâmetros, os seguintes valores foram selecionados para o AG, $P_{MP} = P_{MI} = 0{,}8$, $P_{CR} = 0{,}4$, $GenPop = 10+\lfloor\sqrt{n}\rfloor$ e $Ng = 10000$; para a BT, $MaxIts = 4$, $Nit = 1000$ e $SN = (P_{MP} + P_{MI} + P_{CR})\ GenPop\ Ng\ /\ Nit$; e para os métodos híbridos, $NitH = 200$ e $NgH = 2000$. Ressalta-se que nestes testes pequenas variações dos parâmetros não geraram modificações significativas nos resultados. O parâmetro *SN* foi fixado de forma que o número total de soluções geradas pelo AG e pela BT fosse o mesmo facilitando a comparação entre os métodos.

Os resultados obtidos pela heurística construtiva proposta estão apresentados na Tabela 4. Cada célula da tabela representa a média de *%Dif* para os 10 problemas correspondentes a cada dimensão *n* e fator restritivo *h*.

Tabela 4. Desempenho da heurística proposta em relação ao benchmark.

		n							Média
		10	**20**	**50**	**100**	**200**	**500**	**1000**	
h	**0,2**	1,53	-3,07	-5,33	-6,02	-5,63	-6,32	-6,68	-4,50
	0,4	8,68	0,46	-3,87	-4,42	-3,51	-3,46	-4,26	-1,48
	0,6	19,27	9,78	7,59	4,69	3,71	2,53	3,23	7,26
	0,8	22,97	13,52	8,10	4,70	3,71	2,53	3,23	8,39
Média		13,11	5,17	1,62	-0,26	-0,43	-1,18	-1,12	2,42

Analisando o valor médio de *%Dif* por dimensão observa-se que a heurística proposta apresenta melhores resultados para problemas maiores. Para $h = 0{,}2$ e $h = 0{,}4$, respectivamente, as diferenças percentuais médias são de –4,50% e –1,48%, mostrando

uma melhoria em relação ao *benchmark* para os cenários mais restritos. Além disso, para estes cenários a heurística obteve resultados melhores ou iguais às soluções apresentadas no *benchmark* em 84% e 80% dos problemas.

Para $h = 0{,}6$ e $h = 0{,}8$ as medidas *%Dif* são 7,26% e 8,39%, respectivamente, e em somente 6 dos 140 problemas a heurística obteve resultados melhores ou iguais que os apresentados no *benchmark*, justificando, para tais cenários, a aplicação de meta-heurísticas. Contudo, a diferença percentual geral foi de 2,42% e o tempo de execução médio para cada problema foi de 0,01 segundo, reforçando a idéia de que heurísticas construtivas que incorporam características relevantes do problema tratado podem apresentar bons desempenhos médios com um curto tempo de execução.

A Tabela 5 apresenta os resultados obtidos pelos métodos BT e AG. Cada célula da tabela representa a média de *%Dif* para os 10 problemas correspondentes a cada combinação da dimensão n e do fator restritivo h. Pode-se observar, como esperado, que os valores obtidos por estas estratégias superam os valores obtidos pela heurística HRM. Além disso, nota-se que os resultados obtidos para os cenários com $h = 0{,}2$ e $h = 0{,}4$ são muito similares.

Para $h = 0{,}6$ e $h = 0{,}8$ o procedimento AG apresentou resultados levemente superiores com diferenças percentuais médias de -0,20 e -0,13, respectivamente, enquanto a meta-heurística BT apresentou médias de 0,04 e 0,07. Podemos observar que a melhoria média geral das duas estratégias foi similar, porém as técnicas AG e BT obtiveram soluções com valores que atingiram ou superaram os encontrados no *benchmark* em 258 e 216 problemas, respectivamente. A principal diferença foi observada nos problemas com $n \geq 100$ e $h = 0{,}6$ e $0{,}8$, um total de 80 problemas apresentam estas características. O AG alcançou ou superou os resultados do *benchmark* em 70 problemas, enquanto a BT alcançou ou superou tais resultados em 24 problemas. Ressalta-se ainda que o AG apresentou o melhor desempenho médio em todas as dimensões, inclusive superando os resultados do *benchmark* em todos os casos com $n \geq 20$. Vale comentar que os resultados provenientes do *benchmarck* para problemas com 10 tarefas provêm de algoritmos exatos.

Tabela 5. Dif% para as estratégias BT e AG.

n	*h*	BT	AG
10	0,2	0,25	0,12
	0,4	0,24	0,19
	0,6	0,10	0,03
	0,8	0,00	0,00
20	0,2	-3,84	-3,84
	0,4	-1,62	-1,62
	0,6	-0,71	-0,68
	0,8	-0,41	-0,28
50	0,2	-5,70	-5,68
	0,4	-4,66	-4,60
	0,6	-0,32	-0,31
	0,8	-0,24	-0,19
100	0,2	-6,19	-6,17
	0,4	-4,93	-4,91
	0,6	-0,01	-0,12
	0,8	-0,15	-0,12
200	0,2	-5,76	-5,74
	0,4	-3,74	-3,75
	0,6	-0,01	-0,13
	0,8	-0,04	-0,14
500	0,2	-6,41	-6,41
	0,4	-3,57	-3,58
	0,6	0,25	-0,11
	0,8	0,21	-0,11
1000	0,2	-6,73	-6,75
	0,4	-4,39	-4,40
	0,6	1,01	-0,05
	0,8	1,13	-0,05
Média		-2,01	-2,12

A Tabela 6 apresenta os resultados obtidos com os métodos híbridos HTG e HGT. Pode ser observado que os resultados médios são semelhantes aos encontrados pelo Algoritmo Genético. As estratégias HTG e HGT obtiveram resultados melhores ou iguais do que os do *benchmark* em 205 e 214 problemas, respectivamente.

Tabela 6. Dif% para as estratégias híbridas HTG e HGT.

n	*h*	HTG	HGT
10	0,2	0,12	0,12
	0,4	0,19	0,19
	0,6	0,03	0,01
	0,8	0,00	0,00
20	0,2	-3,84	-3,84
	0,4	-1,62	-1,62
	0,6	-0,71	-0,71
	0,8	-0,41	-0,41
50	0,2	-5,70	-5,70
	0,4	-4,66	-4,66
	0,6	-0,27	-0,31
	0,8	-0,23	-0,23
100	0,2	-6,19	-6,19
	0,4	-4,93	-4,93
	0,6	0,08	0,04
	0,8	-0,08	-0,11
200	0,2	-5,76	-5,76
	0,4	-3,75	-3,75
	0,6	0,37	0,07
	0,8	0,26	0,07
500	0,2	-6,41	-6,41
	0,4	-3,58	-3,58
	0,6	0,73	0,15
	0,8	0,73	0,13
1000	0,2	-6,74	-6,74
	0,4	-4,39	-4,39
	0,6	1,28	0,42
	0,8	1,28	0,40
Média		-1,93	-2,06

A Figura 7 apresenta os tempos de processamento do procedimento AG e das estratégias híbridas. Note que os métodos híbridos apresentam resultados similares com um tempo de processamento menor.

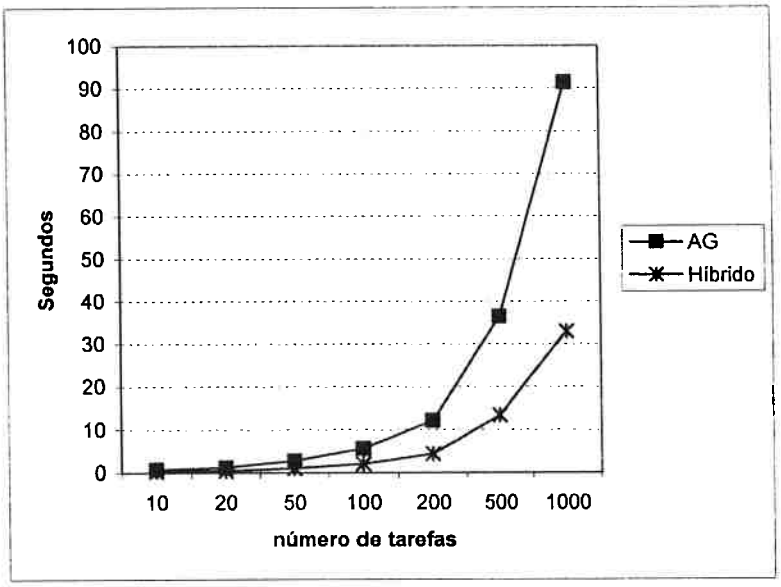

Figura 7: Comparação entre tempos de execução.

Com o intuito de aprimorar a avaliação da estratégia híbrida proposta, uma comparação foi realizada com os resultados apresentados por Feldmann & Biskup (2003). Neste trabalho, o mesmo conjunto de problemas foi estudado, porém somente considerando o fator restritivo h com valores 0,2 e 0,4. Estes autores utilizaram cinco meta-heurísticas diferentes: *Evolutionary Search* (ES), *Evolutionary Search with a Destabilization phase* (ESD), *Simulated Annealing* (SA), *Threshold Accepting* (TA) e *Threshold Accepting with a 'back step'* (TAR). Dentre as cinco meta-heurísticas, o algoritmo TAR apresentou o melhor desempenho. A sua versão original, o algoritmo TA, é uma versão determinística do *simulated annealing* e foi desenvolvida por Dueck & Scheuer (1990). Sempre que a busca parar de melhorar a função objetivo, o passo "back step" do algoritmo TAR pode retornar a melhor solução encontrada até o momento.

A Tabela 7 mostra as diferenças percentuais obtida pela meta-heurística TAR e pela estratégia HGT em 140 problemas. Note que as estratégias apresentam um desempenho similar em termos de melhoria média, embora deva ser destacado que HGT apresenta melhores médias nos problemas com $n \geq 50$.

Tabela 7. Dif% para as estratégias HGT e TAR.

n	*h*	HGT	TAR
10	0,2	0,12	0,00*
	0,4	0,19	0,00*
20	0,2	-3,84*	-3,84*
	0,4	-1,62	-1,63*
50	0,2	-5,70*	-5,64
	0,4	-4,66*	-4,62
100	0,2	-6,19*	-6,16
	0,4	-4,93*	-4,86
200	0,2	-5,76*	-5,72
	0,4	-3,75*	-3,63
500	0,2	-6,41*	-6,39
	0,4	-3,58*	-3,49
1000	0,2	-6,74*	-6,72
	0,4	-4,39*	-4,29
Média		-4,09	-4,07

* Maior melhoria para a dimensão e h analisados.

Uma análise adicional dos resultados obtidos pelas estratégias comparadas é apresentada na Tabela 8. O número de sucessos é definido como o número total de vezes que a melhor solução foi obtida somente pela estratégia analisada. A última coluna, denominada Empates, denota o número de problemas nos quais a melhor solução de ambas estratégias possui o mesmo valor da função objetivo.

Tabela 8. Comparação entre as estratégias TAR e HGT.

n	*h*	Número de sucessos de HGT	Número de sucessos de TAR	Empates
10	0,2	0	2	8
	0,4	0	2	8
20	0,2	0	0	10
	0,4	0	1	9
50	0,2	3	0	7
	0,4	4	1	5
100	0,2	7	1	2
	0,4	8	1	1
200	0,2	7	3	0
	0,4	10	0	0
500	0,2	5	5	0
	0,4	10	0	0
1000	0,2	6	4	0
	0,4	10	0	0
Total		70	20	50

Analisando a Tabela 8 podemos notar que, quando comparada com o algoritmo TAR, HGT obtém melhores resultados em 50%, resultados iguais em 35,7% e piores resultados em 14,3% dos problemas. Além disso, para as maiores dimensões ($n \geq 100$) o método proposto apresenta os melhores resultados em 63 dos 80 problemas considerados.

Tais resultados indicam que a aplicação das meta-heurísticas propostas gerou benefícios na qualidade da solução. Esta melhoria pode ser explicada pelo fato das meta-heurísticas apresentadas explorarem de maneira mais exaustiva a vizinhança, além de considerarem propriedades específicas do problema permitindo, inclusive, a inserção de ociosidade. As meta-heurísticas híbridas obtiveram praticamente o mesmo desempenho que o Algoritmo Genético, porém com um custo computacional muito menor. O sucesso desta metodologia indica que a combinação de meta-heurísticas fornece uma rica linha de pesquisa para a resolução de problemas de otimização combinatória. Maiores detalhes sobre este trabalho podem ser encontrados em Hino *et al.* (2005).

2.3.2. *Flowshop* com Buffer Zero Minimizando o *Makespan*

Ronconi (2004) considera o problema de programação de tarefas no ambiente *flowshop* com buffer zero entre as máquinas. Este ambiente se caracteriza pelo processamento de n tarefas por m máquinas na mesma ordem. Os tempos de processamento de cada tarefa em cada máquina são conhecidos. Como não existem espaços de armazenamento entre as máquinas, filas de tarefas esperando no sistema pelas próximas operações não são permitidas. Uma tarefa finalizada em uma máquina bloqueia esta máquina até que a próxima máquina esteja disponível para o seu processamento.

Conforme Hall & Sriskandarajah (1996) uma das razões para a ocorrência de bloqueio se deve ao próprio processo de produção. Por exemplo, em alguns processos a temperatura ou outras características (tal como a viscosidade) do material requerem que cada operação seja efetuada imediatamente após a anterior. Tais situações ocorrem na indústria química onde tarefas parcialmente processadas (transformações físicas e químicas de material) são, algumas vezes, mantidas nas máquinas por causa da falta de armazenagem intermediária. Reklaitis (1982), em uma revisão sobre a programação de tarefas, afirma que a produção de produtos químicos requer que alguns lotes sejam produzidos no ambiente

flowshop com bloqueio no processo. Outro exemplo pode ser encontrado na produção de blocos de concreto que, em alguns estágios de sua manufatura, não permite estoques (Grabowski & Pempera, 2000).

A pesquisa sobre *flowshop* com buffer zero com o objetivo de minimizar o *makespan* não é muito extensa. Uma boa revisão sobre este ambiente pode ser encontrada em Hall & Sriskandarajah (1996). Estes autores provam que o problema de *flowshop* com três máquinas e buffer zero é *NP-hard in the strong sense*.

Devido à sua complexidade, alguns autores abordaram este ambiente com métodos heurísticos. McCormick *et al.* (1989) desenvolveram um algoritmo, conhecido por *Profile Fitting*, onde se procura seqüenciar primeiro as tarefas que acarretam uma menor soma dos tempos de ociosidade e de bloqueio nas máquinas. Uma abordagem mais ampla é apresentada por Leisten (1990) que propôs duas heurísticas que tratam o problema de *flowshop* de permutação e não-permutação com buffer finito e ilimitado. O principal objetivo destas heurísticas é maximizar o uso dos buffers e minimizar o bloqueio das máquinas. Leisten compara estas heurísticas e outras conhecidas na literatura e conclui que o algoritmo NEH (Nawaz *et al.*, 1983), inicialmente proposto para buffer ilimitado, possui um desempenho superior.

Ronconi (2004) apresenta heurísticas construtivas que exploram as características inerentes à ocorrência de bloqueio para a minimizar o *makespan*. Primeiro, propõe-se uma heurística construtiva baseada em propriedades do *makespan*. Este método é comparado com a heurística *Profile Fitting*. Posteriormente estes dois métodos são avaliados como geradores da solução inicial do procedimento de enumeração do algoritmo NEH. O desempenho das heurísticas é examinado através de um estudo comparativo com o algoritmo NEH original, conhecido como a melhor heurística construtiva para este problema. Os principais resultados deste trabalho estão detalhados a seguir.

Heurística Construtiva MM

A heurística proposta, denominada MinMax (MM), baseia-se em algumas das propriedades do *makespan* apresentadas por Ronconi & Armentano (2001) (veja Seção

2.2.1). Com o intuito de esclarecer a idéia central da heurística MM algumas destas propriedades serão apresentadas a seguir.

Sejam 1, 2,...., i ,...., n uma seqüência de tarefas, onde i representa a i-ésima tarefa, M_1, M_2,...., M_k,...., M_m a seqüência ordenada de máquinas e p_{ik} o tempo de processamento da tarefa i na máquina k. Seja D_{ik} o instante de partida da tarefa i da máquina k. Claramente, $D_{ik} \geq C_{ik}$ onde C_{ik} é o instante de término da tarefa i na máquina k. Dado que $C_{im} = D_{im}$ o *makespan* é dado pelo cálculo de D_{nm}. Uma forma de se calcular o *makespan* de uma dada seqüência é utilizando o grafo descrito na Seção 2.2.1. O instante de término de uma tarefa i na máquina m é dado pelo maior caminho entre os nós (1,0) e (i, m). O *makespan* é dado pelo maior caminho entre os nós (1,0) e (n,m), conhecido como caminho crítico. Logo, o tamanho de um caminho qualquer que une tais nós é um limitante inferior desta medida. Dada uma máquina k considere o caminho cujo tamanho é dado pela seguinte expressão:

$$LC(k) = \sum_{q=1}^{k} p_{1q} + \sum_{r=2}^{n} max(p_{rk} , p_{r-1,k+1}) + \sum_{q=k+1}^{m} p_{nq}$$

onde $p_{i,\,m+1} = 0$ para todo i = 1,..., n.

A Figura 8 ilustra o grafo direcionado com 4 máquinas e a seqüência de tarefas 1,2,..,5. O limitante inferior do *makespan* relativo à máquina 2, LC(2), é dado pelo caminho entre os nós (1,0) e (5,4). Os tempos de processamento p_{11} e p_{12} compõem a primeira parcela do limitante. A segunda parcela do limitante é composta pela soma dos máximos de tarefas consecutivas em máquinas consecutivas. Por exemplo, para atingir o nó (2,2), partindo do nó (1,2), seleciona-se o maior tempo de processamento entre p_{13} e p_{22}. Esta parcela representa o caminho até o nó (5,2). A partir desde nó alcança-se o nó (5,4) através dos tempos de processamento p_{53} e p_{54} que aparecem na terceira parcela do limitante.

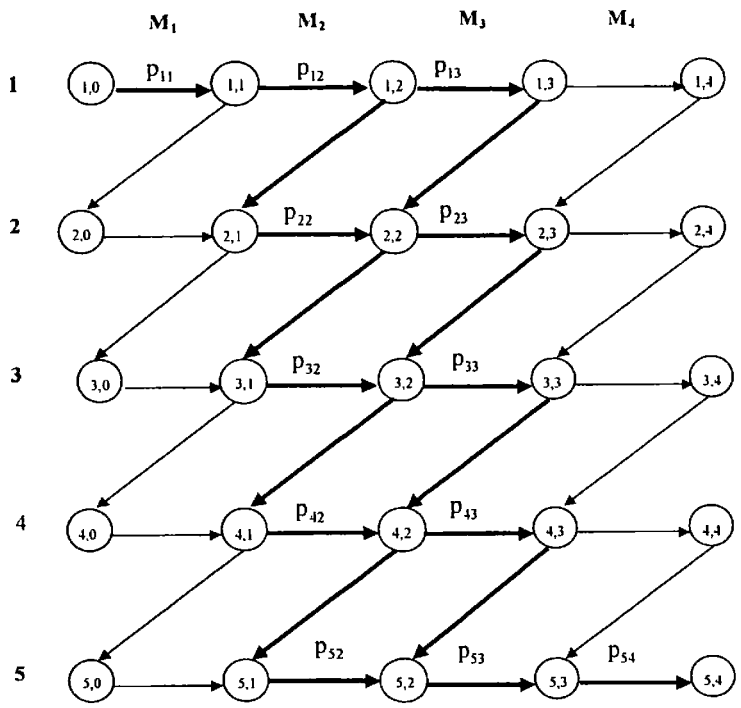

Figura 8: Grafo direcionado para o cálculo do limitante inferior do makespan - LC(2).

Deve-se ressaltar que o verdadeiro caminho crítico não obrigatoriamente passará por todos os nós relacionados a uma única máquina. Através do grafo pode-se observar que qualquer caminho que ligue o nó (1,0) ao nó (*n*,*m*) obrigatoriamente passará pelo arco associado ao tempo de processamento da tarefa na primeira posição na primeira máquina e pelo arco associado ao tempo de processamento da tarefa na última posição na última máquina. Observa-se também que podemos gerar diversos limitantes inferiores formados por caminhos que passam por diferentes máquinas. A grande maioria destes caminhos envolve a somatória de máximos entre tempos de processamento de tarefas consecutivas em máquinas consecutivas, ou seja, expressões do seguinte formato: $\sum_{r} \max(p_{rk}, p_{r-1,k+1})$. Além disso, estes limitantes possuem uma parcela relativa a soma de *m* tempos de processamento.

A heurística MM considera estas características na construção da solução. Inicialmente, o algoritmo fixa na primeira e na última posição as tarefas cujos tempos de processamento são os menores na primeira e na última máquina, respectivamente. A seguir, a heurística adiciona a tarefa da próxima posição da seqüência, a partir da segunda posição,

considerando que o caminho crítico será provavelmente composto por somatórias de máximos entre processamentos de máquinas consecutivas e pela soma de *m* tempos de processamento. Com o objetivo de diminuir tais somas, seleciona-se como tarefa consecutiva àquela já fixada (*i*) a tarefa candidata (*c*) que acarreta o menor valor da seguinte expressão:

$$\alpha \sum_{\ell=1}^{m-1} | P_{c\ell} - P_{i,\ell+1} | + (1-\alpha) \sum_{k=1}^{m} P_{ck} \qquad (42)$$

onde P_{xy} representa o processamento da tarefa x na máquina y. Esta estratégia de seleção da próxima tarefa baseia-se na seguinte afirmação:

Lema *(Ronconi & Armentano 2001): Sejam* $A = a_1, a_2, \ldots, a_t$ *e* $B = b_1, b_2, \ldots, b_t$ *seqüências de números inteiros positivos e defina* $S(A,B) = \sum_{\ell=1}^{t} max(a_\ell, b_\ell)$. *Então S(A,B) é minimizado se as seqüências A e B são ordenadas de maneira que para cada par* (a_ℓ, b_ℓ) *existe* $k \in \{1,2,\ldots,t\}$ *tal que* a_ℓ *e* b_ℓ *correspondam ao k-ésimo menor elemento das seqüências A e B, respectivamente.*

Dado que não podemos ordenar os tempos de processamento das tarefas envolvidas segundo o Lema 1, a estratégia de seleção busca construir uma solução onde a diferença entre estas seja reduzida. É para alcançar tal objetivo que a primeira parcela da expressão (42) é o módulo das diferenças entre tempos de processamento de tarefas consecutivas em máquinas consecutivas. A segunda parcela compõe esta expressão para priorizar as tarefas de menor tempo médio. Dado que o caminho crítico também será composto pela soma de *m* tempos de processamento, é relevante aumentar a probabilidade de tarefas com menores tempos comporem este caminho, posicionando-as no início da seqüência. O parâmetro α é utilizado para ponderar tais parcelas. Uma descrição sucinta dos passos da heurística MM é apresentada a seguir.

Passo 1: Fixe na primeira e na última posição as tarefas com os menores tempos de processamento, na primeira e na última máquina, respectivamente. Faça *prox_pos*= 2.

Passo 2: Selecione entre as tarefas não fixadas a que acarreta o menor valor da expressão (42). Fixe esta tarefa na posição *prox_pos*.

Passo 3: Faça *prox_pos* = *prox_pos* + 1. Se *prox_pos* for igual a *n*, pare. Senão volte para o Passo 2.

Experimentos Computacionais

Os problemas teste foram gerados conforme sugerido por Taillard (1993). Para cada uma das 12 dimensões analisadas foram geradas dez diferentes matrizes de tempos de processamento uniformemente distribuídos entre 1 e 99. Os programas computacionais foram escritos em linguagem C e os testes realizados em um Pentium III - 800MHz com 128 Mb RAM.

Primeiramente foram realizados diversos testes para a seleção do parâmetro α que pondera as duas parcelas da expressão (42). Devido à dificuldade de obter soluções ótimas para estes problemas, utilizou-se como medida de desempenho das diferentes versões da heurística o índice de desvio relativo (Kim 1993). Este índice (IDR) é calculado da seguinte forma:

$$\text{IDR} = \frac{M_a - M_{menor}}{M_{maior} - M_{menor}},$$

onde M_a é a solução do método que está sendo avaliado, e M_{menor} e M_{maior} são respectivamente a melhor e a pior solução encontrada. Note que, quanto menor o valor do índice IDR, melhor o desempenho do método avaliado. A Figura 9 apresenta o desempenho da heurística MM para diferentes valores de α.

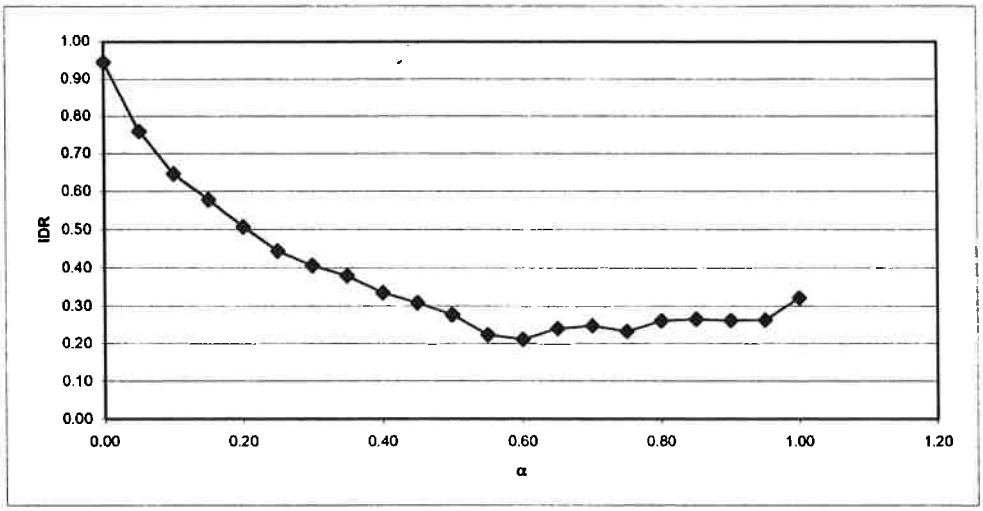

Figura 9. Comportamento do índice de desvio relativo (IDR) em função de α.

Analisando o gráfico observa-se que a heurística apresenta o seu melhor desempenho com α = 0,60. É importante notar que resultados similares são obtidos para valores de $\alpha \geq 0{,}50$, ou seja, quando se prioriza a primeira parcela da expressão (42). Este comportamento mostra uma certa robustez da metodologia com respeito à escolha deste parâmetro.

A seguir a heurística MM é comparada com outros métodos de resolução conhecidos da literatura. Os algoritmos selecionados foram: a regra de despacho LPT (*longest processing time*) e o algoritmo *Profile Fitting* (PF) (McCormick *et al.*, 1989). A regra LPT foi selecionada por sua simplicidade de implementação e por fornecer a solução inicial para o algoritmo NEH adaptado que, de acordo com os testes realizados por Leisten (1990), obtém os melhores resultados para o problema em questão. Este algoritmo gera uma lista de prioridades entre as tarefas ordenando-as em ordem não crescente da soma dos tempos de processamento nas m máquinas.

O algoritmo PF foi considerado pois, conforme Pinedo (1995), apresenta bons resultados para este problema. Porém não foram encontradas comparações deste método com outros algoritmos da literatura. Inicialmente o algoritmo PF seleciona para a primeira posição a tarefa que possui a menor soma dos tempos de processamento. Para determinar a tarefa que ocupará a próxima posição da seqüência, a partir da segunda posição, cada tarefa não seqüenciada é examinada. Para cada tarefa candidata j_c são calculados os instantes de

liberação $D_{j_c k}$ de cada máquina k por esta tarefa caso esta estivesse na posição a qual se candidata. Com estes valores é possível calcular a seguinte expressão:

$$D_{j_c k} - D_{j_i k} - P_{j_c k},$$

que representa o tempo que cada máquina k fica ociosa ou bloqueada quando a tarefa candidata, j_c é posicionada logo após uma dada tarefa, j_i. Esta medida é calculada para todas as máquinas. A tarefa candidata que possuir a menor soma destas medidas é selecionada como a próxima tarefa. Esta forma de seleção das próximas tarefas é repetida até que todas as tarefas estejam seqüenciadas. Maiores detalhes sobre esta heurística podem ser encontrados no trabalho de Pinedo (1995).

A comparação dos três métodos utilizará como medida de desempenho o índice IDR. A Tabela 9 apresenta o índice médio para cada dimensão.

Tabela 9. Comparação entre a heurística MM e os algoritmos PF e LPT.

Dimensão (n x m)	**Heurísticas**		
	LPT	**MM**	**PF**
20x5	1,00	0,25	0,04
20x10	0,98	0,19	0,15
20x20	0,93	0,33	0,20
50x5	1,00	0,24	0,00
50x10	1,00	0,31	0,00
50x20	0,90	0,29	0,13
100x5	1,00	0,29	0,00
100x10	1,00	0,40	0,00
100x20	1,00	0,47	0,00
200x10	1,00	0,41	0,00
200x20	1,00	0,48	0,00
500x20	1,00	0,56	0,00
Média	0,98	0,35	0,04

Neste teste a heurística PF obteve os melhores resultados em todas as dimensões, seguida pela heurística MM. Deve ser destacado que o tempo computacional da regra de despacho LPT pode ser considerado desprezível. As heurísticas MM e PF também possuem tempos pequenos porém maiores que a regra LPT. No maior caso, 500 tarefas e 20 máquinas, as heurísticas MM e PF obtiveram tempos médios de 0,5 e 0,7 segundos, respectivamente.

De acordo com Leisten (1990) e Nowick (1999) a melhor heurística construtiva para minimizar o *makespan* no ambiente *flowshop* com buffer zero é o algoritmo NEH (Navaz *et al.*, 1983), proposto primeiramente para este ambiente com buffer ilimitado. Inicialmente, este algoritmo gera uma lista de prioridades entre as tarefas utilizando a regra de despacho LPT. As duas tarefas com a maior prioridade são selecionadas da lista, e as duas seqüências possíveis para estas tarefas são geradas. As seqüências são avaliadas de acordo com o instante de término da última tarefa e a melhor seqüência parcial é escolhida. As posições relativas destas duas tarefas não se modificam durante todo o algoritmo. A seguir, a tarefa com a terceira maior prioridade é selecionada e três seqüências são geradas ao se posicionar a tarefa no início, no meio, e no fim da seqüência gerada no passo anterior. A melhor seqüência parcial encontrada fixará as posições relativas destas três tarefas no resto do algoritmo. Este processo é repetido até que todas as tarefas sejam fixadas e a seqüência final encontrada.

Dado o bom desempenho das heurísticas PF e MM quando comparadas à regra de despacho LPT, as mesmas serão utilizadas, no lugar desta regra, como ponto de partida para o procedimento de enumeração empregado pelo algoritmo NEH. Estudos similares, com heurísticas de duas fases para o *flowshop* com buffer ilimitado, podem ser encontrados em Suliman (2000), Nagano & Moccellin (2002) e Armentano & Ronconi (1999).

As heurísticas propostas serão doravante denominadas por MME e PFE. Dado que a heurística MM depende do parâmetro α, os testes foram refeitos com este parâmetro para selecionar a melhor versão da heurística MME. O melhor desempenho relativo à qualidade da solução foi apresentado com $\alpha = 0{,}75$. A Tabela 10 apresenta a comparação entre o algoritmo NEH e os algoritmos propostos através do IDR.

Tabela 10. Comparação das heurísticas MME e PFE com o algoritmo NEH através do IDR.

Dimensão (n x m)	Heurísticas		
	NEH	**MME**	**PFE**
20x5	0,53	0,47*	0,59
20x10	0,42*	0,58	0,54
20x20	0,30*	0,36	0,80
50x5	0,90	0,21*	0,26
50x10	0,88	0,45	0,14*
50x20	0,69	0,20*	0,69
100x5	0,99	0,52	0,00*
100x10	0,99	0,23	0,18*
100x20	0,99	0,19*	0,30
200x10	1,00	0,64	0,00*
200x20	1,00	0,13*	0,16
500x20	1,00	0,64	0,00*
Média	**0,81**	**0,39**	**0,30**

* melhor índice para a dimensão analisada

Cada célula da tabela apresenta o IDR médio para os 10 problemas da dimensão correspondente. O melhor desempenho em termos do IDR médio foi obtido pelo algoritmo PFE, que também apresentou excelentes índices nos problemas com a maior dimensão. A heurística MME obteve bons resultados, superando o algoritmo NEH em 10 das 12 dimensões analisadas.

Tais resultados eram esperados dado que as soluções iniciais utilizadas pelos algoritmos PFE e MME exploram características específicas do ambiente *flowshop* com buffer zero, características estas não exploradas pela regra LPT. Além disso, conforme Nagano & Moccellin (2002), o algoritmo NEH é fortemente influenciado pela ordenação inicial das tarefas.

A melhoria média dos algoritmos MME e PFE em relação ao algoritmo NEH é apresentada na Tabela 11. Tal melhoria é calculada utilizando a seguinte expressão:

$$Melhoria = \frac{M_{NEH} - M_a}{M_{NEH}} \; x \; 100,$$

onde M_{neh} e M_a são os valores do *makespan* obtidos através dos algoritmos NEH e do algoritmo avaliado. A Tabela 11 também apresenta o número de vezes que cada algoritmo superou o algoritmo NEH em 120 problemas.

Tabela 11. Melhoria percentual dos algoritmos MME e PFE em relação ao algoritmo NEH.

Dimensão (n x m)	Heurísticas	
	MME	PFE
20x5	0,13	-0,48
20x10	0,02	-0,35
20x20	0,07	-1,43
50x5	2,01	2,02
50x10	1,02	1,67
50x20	1,65	0,09
100x5	1,43	3,38
100x10	1,85	2,12
100x20	1,40	1,03
200x10	1,41	4,22
200x20	1,33	1,42
500x20	1,66	4,59
Média	**1,17**	**1,52**
Nº de Sucessos em relação ao NEH	**97**	**94**
N º de empates em relação ao NEH	**0**	**2**

A heurística PFE apresenta uma média de melhoria levemente superior a do algoritmo MME, porém perde do algoritmo NEH nos problemas menores. O MME ganha em todas as dimensões e apresenta um comportamento mais uniforme nos problemas com o número de tarefas superior a 20. O melhor desempenho do PFE é ressaltado nos problemas de maior dimensão, 500 tarefas e 20 máquinas, nos quais atingiu uma melhoria de 4,59%. Quanto ao número de sucessos e empates, o PFE e o MME ganham ou empatam com o algoritmo NEH em 80% e 81% dos problemas avaliados, respectivamente.

A Tabela 12 resume as informações da Tabela 10 e 11 referentes à heurística PFE. Os valores de melhoria e IDR obtidos por este método aparecem ordenados pela relação *n/m*. Observa-se que o algoritmo PFE beneficia-se do aumento desta relação, obtendo os melhores resultados quando a mesma possui valores maiores ou iguais a 5. O mesmo pode ser observado em relação ao algoritmo PF analisando a Tabela 9. Vale destacar que o algoritmo PFE apresenta o IDR zero para os valores de *n/m* iguais a 20 e 25, ou seja, para estes problemas o algoritmo PFE apresentou as melhores soluções em todos os problemas.

Tabela 12. Melhoria percentual e IDR do algoritmo PFE.

Relação *n/m*	Dimensão (*n x m*)	Média	IDR	Tempo médio de CPU (segundos)
1	20x20	-1,43	0,80	0,005
2	20x10	-0,35	0,54	0,005
2,5	50x20	0,09	0,69	0,066
4	20x5	-0,48	0,59	0,005
5	50x10	1,67	0,14	0,033
	100x20	1,03	0,30	0,456
10	50x5	2,02	0,26	0,016
	100x10	2,12	0,18	0,231
	200x20	1,42	0,16	3,440
20	100x5	3,38	0,00	0,110
	200x10	4,22	0,00	1,731
25	500x20	4,59	0,00	52,540

Este comportamento provavelmente pode ser atribuído ao fato das heurísticas PF e PFE procurarem minimizar a soma dos tempos de ociosidade e de bloqueio nas máquinas. Estes tempos possuem maior impacto no cálculo do *makespan* quando existe um número de tarefas muito maior que o número de máquinas, ou seja *n/m* grande, pois o bloqueio de uma das máquinas pode atrasar a entrada de várias tarefas na linha de produção. Quando ocorre o caso contrário, o número de tarefas é próximo do número de máquinas, a ocorrência do bloqueio não impede que ao menos grande parte das tarefas inicie o seu processamento e sofra com menor impacto o atraso no seu processamento causado por este fenômeno. Os tempos computacionais da heurística PFE também são apresentados na Tabela 12. Os tempos das outras heurísticas analisadas não apresentaram diferenças relevantes.

Podemos resumir os principais resultados deste trabalho como segue. Foram propostas três heurísticas construtivas MM, MME e PFE para o problema de programação de tarefas no ambiente *flowshop* com buffer zero. A heurística MM apresentou bons resultados, porém foi superada pela heurística PF. As heurísticas MME e PFE foram comparadas com o algoritmo NEH, considerado o melhor algoritmo para a resolução deste problema. Os novos métodos apresentaram um desempenho superior em relação ao número de melhores soluções, ao IDR e a porcentagem de melhoria. Os bons resultados obtidos podem ser atribuídos ao fato das soluções iniciais utilizadas pelos algoritmos PFE e MME explorarem características específicas do ambiente *flowshop* com buffer zero,

características estas não exploradas pela regra LPT. Ressalta-se que o algoritmo PFE apresenta uma melhoria significativa em problemas com 500 tarefas. Tal fato indica o potencial deste método para problemas de maior dimensão. Além disso, deve-se ressaltar que as heurísticas construtivas apresentadas possuem um ou nenhum parâmetro para ser calibrado, o que as torna muito simples de serem implementadas e utilizadas na prática.

2.3.3. *Flowshop* com Buffer Zero Minimizando o Atraso Total

Desde o trabalho de Johnson em 1954, muitas pesquisas foram feitas a respeito de problemas de programação de tarefas em ambientes *flowshop*, porém a grande maioria está voltada para o objetivo de minimização do *makespan*. Uma possível razão para isto pode ser o fato dos problemas ficarem mais difíceis quando objetivos envolvendo datas de entrega são considerados (Kim, 1995). Assim, a literatura a respeito de problemas de minimização de atraso total é restrita se comparada ao *makespan*. Um outro fator de destaque na literatura de programação de tarefas é o fato da maioria dos artigos não tratar de problemas com limitações de estoques intermediários (Hall & Sriskandarajah, 1996).

Ronconi & Henriques (2005) abordam o problema de programação de tarefas no ambiente *flowshop* com buffer zero entre as máquinas minimizando o atraso total das tarefas. O nível de dificuldade deste problema pode ser estimado por meio de um caso particular cuja resolução é comprovadamente complexa: o problema com uma máquina é *NP-Hard* para tal critério (Du & Leung, 1990). Critérios de otimização envolvendo datas de entrega vêm assumindo uma importância cada vez maior, pois o não cumprimento da data pode gerar uma série de problemas como: penalidades contratuais, perda de credibilidade – o que pode resultar na perda do cliente – e danos à imagem da empresa – podendo afastar novos clientes (Sen & Gupta, 1984).

Considerando o critério do de minimização do *makespan* podemos citar autores que abordaram este problema com métodos heurísticos, como por exemplo McCormick *et al.* (1989), Leisten (1990), Abadi *et al.* (2000), Caraffa *et al.* (2001) e Ronconi (2004). Especificamente para o ambiente abordado, minimização do atraso total considerando buffer zero, poucos trabalhos foram encontrados. Armentano & Ronconi (2000) propõem a

utilização da meta-heurística Busca Tabu utilizando como solução inicial o algoritmo LBNEH (Armentano & Ronconi 1999) que busca explorar características relacionadas com o critério do atraso. Em um outro trabalho, Ronconi & Armentano (2001) apresentam um algoritmo *branch-and-bound* para minimizar o atraso total onde as tarefas são fixadas no final da seqüência completa. Os autores propõem um limitante inferior para este critério e uma adaptação do mesmo para gerar limitantes inferiores para o *makespan* e para a soma dos instantes de término das tarefas.

A proposta do trabalho de Ronconi & Henriques (2005) é contribuir com o estudo deste problema através da elaboração de métodos heurísticos. Inicialmente os autores propõem uma heurística construtiva que explora, além de características relacionadas às datas de entrega, a ausência de estoques intermediários entre as máquinas. Os resultados da heurística LBNEH foram utilizados como referência para os resultados obtidos neste trabalho. Uma implementação da meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*) também é proposta. Além disso, a estratégia *path relinking* é combinada com esta meta-heurística na busca por soluções de melhor qualidade. Os principais resultados deste trabalho estão detalhados a seguir.

Heurística Construtiva FPD

A heurística proposta encontra uma solução em duas fases. Na primeira fase, a regra de despacho (FPD - *Fitting Processing Times e Due Dates*) gera uma lista de prioridades entre as tarefas. A seguir, guiado por esta lista, o procedimento de inserção do algoritmo NEH (Nawaz *et al.*, 1983) é aplicado para gerar a programação final. Esta heurística será doravante denominada FPDNEH.

O algoritmo FPD trabalha de forma dinâmica, *i.e.*, após a seleção de uma tarefa para uma posição na seqüência já fixada (σ), a lista de prioridades é reorganizada considerando todas as tarefas ainda não fixadas. A escolha da primeira tarefa desta lista é de grande importância para o desempenho do método. Enquanto a primeira tarefa está sendo processada na primeira máquina, as tarefas restantes estão esperando fora do ambiente considerado, o que acarreta um baixo desempenho em todo o sistema. Diversas alternativas foram avaliadas para selecionar esta tarefa. A escolha da tarefa com o menor valor da soma

da sua data de entrega com seu tempo de processamento na primeira máquina foi a medida que apresentou os melhores resultados.

Para a escolha das tarefas seguintes, uma medida de prioridade composta por dois termos é utilizada. Esta medida considera o perfil do tempo de processamento da última tarefa fixada (t_i) na seqüência σ, assim como as datas de entrega das tarefas candidatas. O primeiro termo favorece as tarefas que possuem tempos de processamento próximos a "janelas" geradas pela tarefa t_i em cada máquina. A idéia principal é selecionar uma tarefa que cause o mínimo bloqueio ou tempo ocioso nas máquinas. O segundo termo busca aumentar a prioridade das tarefas que possuem pouco tempo para serem processadas antes de suas respectivas datas de entrega. Uma descrição do algoritmo FPD é apresentada a seguir.

Passo 1. Seja $\sigma = \varnothing$.

Passo 2. Calcule I_k para todas as tarefas k $\notin \sigma$:

$$I_k = d_k + p_{k1}$$

Escolha a tarefa com o menor I_k para a primeira posição na seqüência σ. Faça $next_pos = 2$.

Passo 3. Calcule LB_k (Armentano & Ronconi, 1999) para todas as tarefas k $\notin \sigma$:

$$LB_k = d_k - \sum_{j=1}^{m} p_{kj}$$

Passo 4. Calcule o instante de partida da última tarefa em σ, t_i, de cada máquina $D_{ii,j}$, j=1, 2,...., m.

Passo 5. Calcule o período de tempo disponível (b_j) em cada máquina:

$$b_j = D_{ii,j+1} - D_{ii,j}\,, \quad j = 1, 2, ..., m-1$$

Passo 6. Calcule para todas as tarefas $k \notin \sigma$:

$$F_k = \rho . fit'_k + (1-\rho) . dynslack'_k\,, \tag{43}$$

onde:

$$fit'_k = \frac{fit_k - \min\limits_{k \notin \sigma}(fit_k)}{\max\limits_{k \notin \sigma}(fit_k) - \min\limits_{k \notin \sigma}(fit_k)} \qquad dynslack'_k = \frac{dynslack_k - \min\limits_{k \notin \sigma}(dynslack_k)}{\max\limits_{k \notin \sigma}(dynslack_k) - \min\limits_{k \notin \sigma}(dynslack_k)}$$

$$fit_k = \sum_{j=1}^{m-1} \left| b_j - p_{kj} \right| \qquad \text{e} \qquad dynslack_k = LB_k - D_{ii,1}$$

Passo 7. Posicione a tarefa com o menor valor de F_k na posição $next_pos$ de σ.

Passo 8. Se $next_pos = n$, pare. Senão, faça $next_pos = next_pos + 1$ e volte ao Passo 4.

Note que ambos termos de F_k, *fit'* (*perfect fit*) e *dynslack'* (*dynamic slack*) tomam valores entre 0 e 1. O objetivo desta normalização é evitar a possível distorção causada pela magnitude dos valores de cada termo. O parâmetro ρ é utilizado para ponderar os termos.

A seqüência gerada pelo algoritmo FPD é utilizada como lista de prioridades para o procedimento de inserção do algoritmo NEH. As duas tarefas com maior prioridade são selecionadas desta lista e as duas possíveis seqüências são geradas. Estas seqüências são avaliadas de acordo com a soma dos atrasos destas tarefas e a melhor seqüência parcial é selecionada. A posição relativa destas tarefas não se modifica durante toda a execução do algoritmo. A seguir, a tarefa com a terceira maior prioridade é escolhida e três seqüências são geradas posicionando esta tarefa no início, no meio e no final da seqüência selecionada no passo anterior. A melhor seqüência parcial definirá a posição relativa destas três tarefas. Em caso de empate, a ordem da lista de prioridades fornecida pelo algoritmo FPD é mantida. Este procedimento segue até que todas as tarefas estejam programadas.

GRASP

Ronconi & Henriques (2005) propõem uma heurística baseada na meta-heurística GRASP como método de exploração do espaço de soluções. Conforme os autores, esta meta-heurística, desenvolvida por Feo & Resende (1989, 1995), foi escolhida para este estudo, pois a mesma foi aplicada com sucesso a diversos problemas de programação de tarefas (veja, por exemplo, Rios-Mercado & Bard, 1998; Akturk & Ozdemir, 2001 e Aiex *et al.*, 2003). Além disso, GRASP apresenta características especiais, tais como a simplicidade de implementação e um pequeno número de parâmetros.

A meta-heurística GRASP, segundo Resende & Ribeiro (2003), é um processo iterativo ou de múltiplos reinícios, no qual as iterações consistem em duas fases: construção e busca local. A fase de construção gera uma solução viável cuja vizinhança é investigada até que um ótimo local seja encontrado durante a fase de busca local. A melhor solução encontrada nas iterações é mantida como a programação final.

Inicialmente, a meta-heurística GRASP foi utilizada somente com seus componentes básicos. Chamamos esta versão de versão básica. Os detalhes de como as duas fases do GRASP foram adaptadas para o problema tratado serão apresentados a seguir.

Construção

GRASP requer uma boa heurística construtiva para gerar múltiplas soluções. Uma solução viável é construída adicionando um elemento de cada vez. Os elementos são aleatoriamente selecionados dentre os componentes de uma lista restrita de candidatos (*restricted candidate list* - RCL). Cada vez que um elemento é adicionado à solução parcial, a RCL é atualizada. Este procedimento é repetido até que uma solução viável seja encontrada. Nesta fase, GRASP combina uma construção "gulosa" com uma aleatória. A existência da RCL fornece o aspecto "guloso" enquanto o aspecto probabilístico é dado pela aleatoriedade na seleção de um elemento desta lista. Este processo permite ao método gerar diferentes soluções cada vez que é executado.

Para a construção da RCL, um conjunto de elementos candidatos C é construído com os elementos que mantém a viabilidade da solução parcial. O custo *c(e)* é calculado para cada elemento $e \in C$, porém somente os melhores elementos são selecionados para compor a RCL. A função *c(e)* mede o benefício local de incluir um elemento na solução parcial. O tamanho da RCL pode ser fixo ou depender da quantidade de elementos de boa qualidade.

Sejam $c^{min} = \min_{e \in C} \{c(e)\}$ e $c^{max} = \max_{e \in C} \{c(e)\}$. Três estratégias descritas por Pitsoulis & Resende (2002) foram analisadas para a seleção dos elementos: *i) cardinality_based_scheme*: $\lfloor \alpha . n \rfloor$ candidatos de elite são colocados em RCL, onde α é um parâmetro dentro do intervalo [0,1]; *ii) value_based_scheme*[1]: $RCL = \{e \in C \mid c(e) \leq c^{min} \cdot (1+p)\}$, onde p é um parâmetro dentro do intervalo [0,∞) e *iii) value_based_scheme*[2]: $RCL = \{e \in C \mid c(e) \leq c^{min} + \alpha \cdot (c^{max} - c^{min})\}$, onde α é um parâmetro dentro do intervalo [0, 1]. No primeiro caso, o tamanho da RCL é previamente definido enquanto nos outros casos ele varia a cada iteração.

Busca local

A eficiência da busca local depende de diversos fatores tais como a definição da vizinhança, a estratégia de busca e a solução inicial. Neste trabalho, a vizinhança é definida pelo movimento de inserção que consiste na remoção de uma tarefa de sua posição original

e sua inserção nas n-1 posições restantes. Este movimento gera uma vizinhança de tamanho $(n-1)^2$.

Duas técnicas de busca usualmente utilizadas são consideradas: *best improving* e *first improving*. Na estratégia *best improving*, a vizinhança completa é analisada e a solução atual é substituída pela melhor solução encontrada. Na estratégia *first improving*, a solução atual é substituída pela primeira solução que melhora o seu valor. Resende & Ribeiro (2003) afirmam que ambas estratégias levam a um mesmo resultado final em muitas aplicações, porém a estratégia *first improving* é mais rápida. Além disso, de acordo com estes autores, uma convergência prematura para um ótimo local não global é mais provável de ocorrer com o uso da estratégia *best improving*. Por tais razões, para esta implementação selecionou-se a técnica *first improving*.

GRASP com Path Relinking

A técnica *path relinking* foi proposta por Glover (1996) para explorar caminhos que conectavam soluções de alta qualidade (soluções de elite) obtidas por Busca Tabu. O primeiro trabalho conhecido onde a associação desta técnica com GRASP foi utilizada como estratégia de intensificação foi apresentado por Laguna & Martí (1999). Outras aplicações mais recentes e bem sucedidas desta associação podem ser encontradas em Canuto *et al.* (2001), Aiex *et al.* (2003) e Festa *et al.* (2005).

Na aplicação de *path relinking* proposta por Ronconi & Henriques (2005) umas poucas soluções de elite são armazenadas durante a aplicação do GRASP e posteriormente utilizadas como soluções guia para o procedimento *path relinking*. Todo ótimo local é considerado para compor tal conjunto e é incluído caso o valor de sua função objetivo seja melhor que algum elemento do conjunto. De acordo com Resende & Ribeiro (2003) existem indícios de que a aplicação da estratégia *path relinking* como procedimento de intensificação em cada ótimo local é mais efetiva que sua aplicação somente no final da busca. Por tal motivo, na implementação apresentada em cada ótimo local um caminho é traçado utilizando uma solução de elite como ponto inicial e o ótimo local como solução destino. Esta estratégia é conhecida como *backward relinking*.

Com o intuito de construir um caminho conectando a solução inicial x_i e solução destino x_g, a diferença simétrica entre estas duas soluções, dada por $\Delta(x_i, x_g)$, deve ser calculada. Para o problema tratado, esta diferença é definida como o número de tarefas que não estão na mesma posição na solução inicial e na solução destino. Primeiro, o procedimento examina todos os possíveis movimentos de inserção, $m \in \Delta(x_i, x_g)$, partindo da solução inicial e seleciona a melhor solução gerada. Este movimento é realizado e uma nova solução intermediária (x) é encontrada. Se necessário, a melhor solução encontrada até o momento, x^*, é atualizada. O conjunto de movimentos disponíveis $\Delta(x, x_g)$ é atualizado e uma nova iteração do procedimento começa com a solução atual x. Este procedimento termina quando a solução destino x_g é alcançada, *i.e.*, $x = x_g$. Resende & Ribeiro (2005) descrevem uma estratégia de busca similar a recém descrita.

Experimentos Computacionais

Os problemas teste foram gerados conforme sugerido por Taillard (1993). Para cada uma das 12 dimensões analisadas foram geradas dez diferentes matrizes de tempos de processamento uniformemente distribuídos entre 1 e 99. Para cada uma destas matrizes quatro diferentes cenários de datas de entrega foram gerados (veja Seção 2.2.1). Os programas computacionais foram escritos em linguagem C++ e os testes realizados em um Pentium IV - 2.3 GHz com 512 Mb RAM.

A heurística FPDNEH foi avaliada considerando a porcentagem de melhoria em relação ao algoritmo LBNEH proposto por Armentano & Ronconi (1999, 2000). De acordo com os testes computacionais realizados por estes autores, LBNEH é o melhor heurística construtiva para minimizar o atraso total no ambiente *flowshop* com buffer zero e ilimitado. A melhoria é obtida utilizando a seguinte expressão:

$$Melhoria\% = \frac{T_{LBNEH} - T_{FPDNEH}}{T_{LBNEH}} . 100 ,$$

onde T_{LBNEH} e T_{FPDNEH} são os valores do atraso total obtidos pelos algoritmos LBNEH e FPDNEH, respectivamente.

Inicialmente diversos testes foram realizados para selecionar o parâmetro ρ da expressão (43). O desempenho de FPDNEH foi analisado em termos da melhoria média e do número de melhores soluções para os valores {0; 0,1; 0,2;...1}. A heurística FPDNEH apresentou seu melhor desempenho em ambos critérios com o parâmetro ρ = 0,3, *i.e.*, quando o termo *dynslack* possui o maior peso. A Tabela 13 apresenta a porcentagem de melhoria média de cada classe com 10 problemas com a aplicação do algoritmo FPDNEH.

Tabela 13. Melhoria percentual de FPDNEH comparado com LBNEH.

Dimensão	**Cenário**				**Média**
(*n* x *m*)	**1**	**2**	**3**	**4**	
20x5	18,12	-13,96	0,35	7,64	3,04
20x10	3,26	3,03	0,38	1,69	2,09
20x20	-1,38	-3,50	-0,44	-0,52	-1,46
50x5	24,25	62,70	10,58	2,23	24,94
50x10	7,63	11,78	-0,61	1,86	5,17
50x20	2,76	1,62	-0,80	-2,38	0,30
100x5	39,56	83,94	15,86	-3,48	33,97
100x10	17,49	34,35	4,93	1,19	14,49
100x20	9,56	6,84	1,92	0,64	4,74
200x10	25,85	56,07	10,29	-1,31	22,73
200x20	13,88	18,39	4,04	-0,94	8,84
500x20	19,04	31,86	7,91	-0,79	14,51
Média	15,00	24,43	4,53	0,49	11,11

Analisando a Tabela 13, observa-se que o algoritmo FPDNEH apresenta os melhores resultados médios em todos os cenários, especialmente nos cenários 1 e 2, com baixo fator de atraso. O algoritmo proposto supera o algoritmo LBNEH em 75% das classes e apresenta uma melhoria média superior a 10% em 35,4% das classes. Considerando o número de resultados superiores, o algoritmo FPDNEH superou o algoritmo LBNEH em 336 de 480 problemas teste e apresentou soluções com a mesma soma dos atrasos em 8 problemas. Este comportamento, provavelmente, se deve ao fato de que, no primeiro estágio, o algoritmo FPDNEH considera um número maior de características que são inerentes ao problema, tal como a situação de bloqueio. Na maior dimensão, 500 tarefas e 20 máquinas, a heurística FPDNEH apresentou um tempo de execução de 15,1 segundos. O tempo de CPU foi menor que 1 segundo para os outros problemas.

Diversos testes foram realizados para selecionar os componentes da versão básica da meta-heurística GRASP e os seguintes parâmetros foram escolhidos. A solução inicial é gerada pela regra de despacho FPD com a utilização da RCL no Passo 2 e 7. O tamanho da RCL é definido pelo critério *value_based_scheme*[1] (p = 0,35). Antes da utilização da busca local, a solução é aprimorada pelo procedimento de inserção do algoritmo NEH. A busca local utiliza o movimento de inserção e a estratégia *first improving*. O algoritmo é executado até que n buscas locais sejam efetuadas ou até que o tempo máximo de CPU de 1800 segundos seja atingido.

A meta-heurística GRASP foi avaliada considerando a sua porcentagem de melhoria em relação ao algoritmo FPDNEH. A Tabela 14 mostra que uma melhoria média de 30,98% foi obtida pelo GRASP.

Tabela 14. Melhoria percentual da versão básica de GRASP em relação a heurística FPDNEH.

Dimensão (n x m)	**Melhoria Percentual**				**Melhoria por dimensão**	**Tempo de CPU (s)**
	Cenário 1	**Cenário 2**	**Cenário 3**	**Cenário 4**		
20x5	42,24	27,52	18,29	11,97	25,00	0,10
20x10	16,78	19,21	9,76	7,07	13,20	0,19
20x20	9,68	9,66	5,91	5,07	7,58	0,31
50x5	57,32	97,67	25,87	28,93	52,45	9,48
50x10	39,85	61,91	21,22	15,75	34,68	23,71
50x20	29,94	27,17	11,67	10,78	19,89	38,11
100x5	59,56	68,81	26,21	37,76	48,09	308,84
100x10	45,94	87,50	21,84	26,61	45,47	730,46
100x20	33,90	69,22	15,51	11,91	32,64	1517,56
200x10	40,44	99,28	20,09	21,51	45,33	1705,82
200x20	28,76	69,32	14,23	13,57	31,47	1800,02
500x20	11,38	36,84	6,09	9,51	15,95	1800,02
Média	34,65	56,18	16,39	16,70	30,98	661,22

Através da coluna **Melhoria por dimensão** pode-se observar que, na maioria dos casos, o aumento do número de tarefas com o mesmo número de máquinas favorece o desempenho do algoritmo GRASP. A principal exceção é o caso com 500 tarefas e 20 máquinas, provavelmente devido ao reduzido período de tempo permitido para a execução do algoritmo. A última coluna apresenta o tempo computacional médio do GRASP. Note que a heurística FPDNEH é mais rápida, porém melhores resultados podem ser encontrados através da utilização da meta-heurística GRASP.

Testes preliminares foram realizados para selecionar a dimensão de S, o número de soluções de elite, na versão da meta-heurística GRASP com *path relinking*. Os melhores resultados foram obtidos com |S| = 1. Esta estratégia foi executada com o mesmo critério de parada que a versão básica. A Tabela 15 apresenta a porcentagem de melhoria obtida por esta associação em relação à heurística FPDNEH. Os números em negrito indicam as classes de problemas onde GRASP com *path relinking* superou a melhoria média obtida pela versão básica.

Observe que a aplicação desta estratégia acarretou em melhores soluções em 58,3% das classes e foi moderadamente superior, em termos de melhoria média, nos cenários 1, 3 e 4. Nota-se também que os melhores resultados foram atingidos em problemas de tamanho médio com $n \leq 100$, onde o tempo de execução não foi um fator crítico. Para os problemas de maiores dimensões, dentro do período de tempo máximo permitido, esta estratégia apresenta um desempenho levemente inferior quando comparada com a versão básica. Embora um pequeno ganho geral tenha sido obtido, ressalta-se que esta estratégia é fácil de implementar e utiliza para a sua execução praticamente o mesmo tempo que a versão básica.

Tabela 15. Melhoria percentual de GRASP com path relinkink em relação a heurística FPDNEH.

Dimensão (n x m)	**Melhoria Percentual**				**Melhoria por dimensão**	**Tempo de CPU (s)**
	Cenário 1	**Cenário 2**	**Cenário 3**	**Cenário 4**		
20x5	**44,42**	27,00	**18,80**	**12,08**	**25,57**	0,11
20x10	**17,48**	**19,71**	**9,94**	6,76	**13,47**	0,22
20x20	9,09	**9,78**	5,81	5,06	7,44	0,34
50x5	**60,14**	**98,00**	**25,90**	**29,62**	**53,42**	9,72
50x10	**40,15**	**63,19**	**21,38**	**16,79**	**35,37**	24,51
50x20	**30,70**	27,16	**12,33**	**10,90**	**20,27**	39,55
100x5	**59,95**	68,73	**28,07**	**37,77**	**48,63**	317,87
100x10	**46,66**	**88,02**	**22,40**	26,21	**45,82**	755,73
100x20	**34,24**	**71,04**	15,16	**12,25**	**33,17**	1576,46
200x10	39,75	98,73	19,67	**21,53**	44,92	1676,97
200x20	27,81	69,12	14,18	12,97	31,02	1803,66
500x20	9,57	32,92	5,45	8,68	14,15	1800,03
Média	**35,00**	56,11	**16,59**	**16,72**	**31,11**	667,10

A Tabela 16 apresenta uma análise adicional. O número de sucessos é utilizado para avaliar o desempenho da versão básica do GRASP em relação à sua associação com *path relinking* em 480 problemas. O número de sucessos é definido como o número de vezes que a estratégia analisada obteve a melhor solução. "Max", nas colunas 2 e 4, representa o maior ganho que uma versão obteve comparada com a outra. "Empates", na última coluna, denota o número de problemas nos quais as melhores soluções de ambas as versões possuem o mesmo valor do atrasos total. Pode ser observado que a aplicação de GRASP com *path relinking* resultou em um maior número de sucessos.

Tabela 16. Comparação entre GRASP com path relinkink e na versão básica.

Número de sucessos com *path relinking*	**Max %**	**Número de sucessos com a versão básica**	**Max %**	**Empates**
213	100,0	180	95,5	87

Para aprimorar a avaliação dos métodos propostos, foi realizado um estudo comparativo com os resultados ótimos provenientes da aplicação do algoritmo *branch-and-bound* proposto por Ronconi & Armentano (2001). Esta técnica foi aplicada a uma amostra de 5 matrizes de tempos de processamento para cada dimensão ($n \leq 20$ e $m \leq 10$) e 4 cenários de datas de entregas para cada matriz, isto é, em 27 x 5 x 4 = 540 problemas. A tolerância utilizada foi de 0,5%. Respeitando o tempo máximo de CPU permitido, o algoritmo b&b obteve a solução ótima em 486 problemas. Os métodos propostos – algoritmo FPDNEH, a versão básica do GRASP e sua associação com *path relinking* – foram aplicados aos problemas com soluções ótimas conhecidas. As duas versões da meta-heurística GRASP foram executadas até que n buscas locais fossem finalizadas. O tempo de execução para cada problema foi menor que 0,14 segundo. Nestes problemas de dimensão reduzida, a versão básica do GRASP obteve soluções ótimas em 287 problemas, enquanto a associação do GRASP com *path relinking* obteve valores ótimos em 297 problemas. A heurística FPDNEH encontrou soluções ótimas em 61 problemas.

Podemos resumir os principais resultados deste trabalho como segue. Foram propostas três heurísticas FPDNEH, GRASP versão básica e sua associação com *path relinking* para o problema de programação de tarefas no ambiente *flowshop* com buffer zero minimizando o atraso total. A heurística FPDNEH superou o algoritmo LBNEH, conhecido como a melhor heurística construtiva para este problema. A versão básica da

meta-heurística apresentou uma melhoria média de 30,98% em relação à FPDNEH e melhores resultados em 475 dos 480 problemas teste. A associação de GRASP com a técnica *path relinking* como estratégia de intensificação acarretou melhores soluções com praticamente o mesmo tempo de execução.

2.3.4. *Flowshop* com Buffer Ilimitado Minimizando a Soma dos Adiantamentos e Atrasos

Sakuraba & Ronconi (2006) tratam do problema de programação de tarefas em um ambiente *flowshop* com duas máquinas e buffer ilimitado onde se quer minimizar a soma dos desvios das datas de conclusão das tarefas em relação a uma data de entrega comum. Além disso, considera-se uma mesma seqüência de tarefas nas máquinas, conhecido como programação permutacional (*permutational scheduling*). Um caso particular dentre os problemas de minimização dos desvios absolutos ocorre quando as tarefas possuem uma data de entrega comum. Tais ambientes podem ser encontrados por exemplo na indústria química, onde podemos encontrar diferentes produtos a serem fabricados através de um mesmo processo, e se deseja misturá-los o mais próximo possível de um determinado instante para evitar a sua deterioração (Sarper, 1995).

Diversos problemas de programação de tarefas com penalidades de atraso e adiantamento e data de entrega comum já foram abordados na literatura, como se pode notar nas revisões de trabalhos feitas por Baker & Scudder (1990) e Gordon *et al.* (2002).

Para o problema de uma máquina, Kanet (1981) propõe um algoritmo que fornece a solução ótima em tempo polinomial (complexidade $O(n \log n)$) quando a data de entrega é maior ou igual à soma dos tempos de processamento das tarefas. Ainda para este ambiente, Bagchi *et al.* (1986) expandem os resultados de Kanet (1981) e apresentam um algoritmo para a determinação de múltiplas programações ótimas quando a data de entrega é maior do que Δ (onde Δ é a somatória dos tempos de processamento das tarefas nas posições ímpares se n é impar e das tarefas nas posições pares caso contrário, estando as tarefas ordenadas de forma não-descrescente dos tempos de processamento). Além disso, é apresentado um procedimento de enumeração implícita quando é fornecida uma data de entrega comum

restrita, ou seja, menor do que aquela definida por Kanet (1981). Quanto a sua complexidade, o problema com uma máquina e data de entrega restrita é *NP-Hard.*

Uma das referências sobre *flowshop* com o objetivo de minimizar a soma dos desvios sobre uma data de entrega comum é o trabalho de Sung & Min (2001), onde há a presença de duas máquinas de processamento em bateladas (BPM, *Batch Processing Machines*), que são máquinas capazes de processar um dado número de tarefas simultaneamente. São apresentadas propriedades para os três tipos de problemas com BPM, que envolvem além do seqüenciamento em ambas as máquinas, critérios eficazes para a formação das bateladas. Yeung *et al.* (2004) tratam de um *flowshop* com duas máquinas, com o objetivo de minimizar a soma dos desvios sobre uma janela de tempo comum ao invés de uma data de entrega comum. Os autores apresentam uma heurística construtiva baseada em uma série de propriedades e em trocas de pares de tarefas, além de limitantes utilizados em conjunto com o método *Branch-and-Bound.*

Para o ambiente *flowshop* com duas máquinas e minimização da soma dos desvios em relação a uma data comum, o único trabalho encontrado é o de Sarper (1995). Tal autor apresenta um modelo de Programação Linear Inteira Mista (PLIM) para o problema com múltiplas seqüências, além de um algoritmo de programação de tarefas para uma dada seqüência das mesmas. Esse algoritmo baseia-se na afirmação do autor de que não é necessário haver tempo ocioso entre tarefas na primeira máquina, e funciona de seguinte maneira: numa primeira etapa, as tarefas são programadas em ambas máquinas para iniciarem seu processamento o quanto antes; em seguida adiciona-se uma unidade ao instante de início de processamento de todas as tarefas na segunda máquina enquanto houver melhora no valor da função objetivo. Este algoritmo é utilizado juntamente com três heurísticas de seqüenciamento para avaliação das seqüências parciais, gerando três heurísticas construtivas.

A heurística de melhor desempenho de Sarper (1995), segundo o próprio autor, é a H3, que utiliza um procedimento de construção de seqüências parciais semelhante ao do algoritmo NEH (Nawaz *et al.*, 1983). Inicialmente é feita uma ordenação das tarefas em LPT (*Longest Processing Time*) utilizando apenas os tempos de processamento na segunda máquina. Em seguida, as tarefas são inseridas na seqüência segundo essa ordenação inicial

em todas as posições possíveis da seqüência anterior. Para avaliar a qualidade das seqüências parciais geradas, utiliza-se o algoritmo de programação proposto pelo autor. Após a obtenção da seqüência final, as variáveis binárias que indicam a seqüência são fixadas no modelo proposto e o mesmo é utilizado para determinar a programação ótima. Comparando os resultados obtidos com os valores ótimos para os problemas de dimensão menor (5 e 6 tarefas), a H3 possui um desvio médio de menos de 3%. Tal desempenho é inferior ao da H1, mas o autor afirma que para problemas maiores (10 a 20 tarefas) o desempenho da H3 é superior, tanto na qualidade da solução obtida como no custo computacional.

A proposta do trabalho de Sakuraba & Ronconi (2006) é apresentar propriedades para este problema, além de heurísticas que obtenham resultados superiores aos de Sarper (1995), tanto em termos de custo computacional quanto em qualidade das soluções. Foram consideradas datas de entrega restritas, pois para datas irrestritas, uma adaptação do algoritmo de Kanet (1981) pode ser utilizada para encontrar uma solução ótima em tempo polinomial. Os principais resultados deste trabalho estão descritos a seguir.

Propriedades

Sarper (1995) afirma que não é necessário adicionar qualquer tempo ocioso na primeira máquina, sendo que as tarefas podem ser programadas para iniciar seu processamento tão cedo quanto possível sem prejudicar o valor de função objetivo. Entretanto, tal afirmação não é válida para a programação na segunda máquina. Outras propriedades do problema em estudo serão apresentadas a seguir. Considere que os valores de tempos de processamento e de datas de entrega são números inteiros.

Propriedade 1: *O ganho na função objetivo obtido ao atrasarmos apenas um bloco (conjunto de tarefas consecutivas) de tarefas na segunda máquina somando uma unidade ao instante de início de processamento de cada uma das tarefas do bloco é dado pelo número de tarefas adiantadas menos o número de tarefas não-adiantadas neste bloco antes desta alteração. Este mesmo ganho pode ser novamente obtido atrasando este bloco enquanto o número de tarefas adiantadas neste bloco não se alterar devido ao seu atraso.*

Prova: Quando um bloco de tarefas na segunda máquina é atrasado em uma unidade de tempo, o valor da função objetivo é melhorado em uma unidade para cada tarefa adiantada e piorado em uma unidade para cada tarefa do bloco atrasada ou que termina exatamente na data de entrega antes do atraso da programação. Enquanto não houver aumento do número de tarefas atrasadas deste bloco causado pelo atraso da programação, com uma tarefa que terminava na data de entrega passando a terminar depois dela, o ganho será o mesmo para cada unidade atrasada. Assim, se um ganho for obtido atrasando este bloco em uma unidade, podemos atrasá-lo até que tenhamos uma tarefa terminando exatamente na data de entrega, ou até que a última tarefa do bloco termine no instante de início da tarefa seguinte, continuando a obter uma melhora no valor da função objetivo. ■

Propriedade 2: *Numa programação ótima, não há tempo ocioso entre tarefas que iniciam seu processamento na segunda máquina antes da data de entrega.*

Prova: Qualquer tempo ocioso entre duas tarefas que iniciam seu processamento na segunda máquina antes da data de entrega pode ser eliminado atrasando-se o instante de início de processamento das tarefas anteriores de forma que ela termine seu processamento exatamente no instante de início da próxima tarefa. Assim, a penalização por adiantamento desta tarefa também diminuirá, sem prejudicar o restante da programação. ■

Propriedade 3: *Numa programação ótima, as tarefas atrasadas são programadas para iniciarem seu processamento no instante máximo entre o instante seguinte ao término de seu processamento na primeira máquina e o instante de término de processamento da tarefa anterior na segunda máquina.*

Prova: Como o objetivo da programação é minimizar a soma dos desvios dos instantes de conclusão das tarefas em relação a uma data de entrega comum, não existe nenhuma motivação para que o início do processamento das tarefas atrasadas seja postergado além do instante de tempo imposto pelas restrições físicas do problema. ■

Propriedade 4: *Nenhuma programação ótima pode ter mais do que n/2 tarefas adiantadas.*

Prova: Qualquer programação com i tarefas adiantadas pode ser atrasada em uma unidade (somando uma unidade a todos os instantes de início de processamento das tarefas na segunda máquina), o que gera um ganho na função objetivo de $2i - n$. Esse ganho corresponde ao número de tarefas adiantadas (i) menos o número de tarefas não-adiantadas ($n - i$). Assim, se $i > n/2$ teremos sempre um ganho no valor da função objetivo e uma programação com $i > n/2$ tarefas adiantadas nunca poderá ser ótima. ■

Heurística

As heurísticas propostas possuem duas etapas: em uma delas se constrói a seqüência de tarefas e na outra se determina a sua programação. Para a construção de seqüências, utilizaremos algoritmos baseados na heurística NEH (Nawaz *et al.*, 1983) e em busca local, e para sua programação apresentaremos um algoritmo que fornece uma programação ótima dada uma seqüência de tarefas. Tal algoritmo, que utiliza as propriedades do problema descritas anteriormente, foi denominado Algoritmo de Programação de Tarefas ou APT.

O APT é baseado no atraso de blocos de tarefas. Estas tarefas são programadas inicialmente para iniciarem seu processamento o quanto antes possível, impossibilitando seu adiantamento a partir dessa programação. As tarefas adiantadas são atrasadas de forma que não haja folga entre elas na segunda máquina (Propriedade 2) sem alterar os instantes de início de processamento das tarefas atrasadas. Caso existam mais tarefas adiantadas do que atrasadas, a programação na segunda máquina pode ser atrasada de forma que a tarefa na posição $\lceil n/2 \rceil$ termine exatamente na data de entrega (Propriedade 4) e as tarefas atrasadas continuem sendo programadas para serem processadas o quanto antes possível (Propriedade 3).

Numa próxima etapa, as folgas entre as tarefas atrasadas são eliminadas atrasando todas as tarefas anteriores àquela folga, caso esse atraso gere um ganho para a programação. As folgas são eliminadas começando da primeira folga existente na programação, de forma a maximizar o ganho obtido com esse movimento, pois atrasamos todas as tarefas adiantadas e um mínimo de tarefas atrasadas. Para garantir que este movimento irá gerar um ganho, é preciso que nesse bloco de tarefas a ser atrasado o número de tarefas adiantadas seja maior do que o número de tarefas não-adiantadas, e o

atraso não seja tão grande a ponto do número de tarefas atrasadas aumentar (Propriedade 1). Assim, esse atraso na programação é realizado em passos de forma tal que o tamanho do passo seja o tamanho da folga ou seja a diferença entre d e a data de conclusão da última tarefa adiantada. Esse atraso é repetido enquanto houver ganho na programação.

Por fim, as tarefas na primeira máquina são atrasadas de forma a não deixar tempos ociosos entre o processamento de cada tarefa entre as duas máquinas ($W_{j1} = 0$ para $j = 1,2,...n$), visando minimizar o custo de estoques intermediários entre as máquinas, apesar desta medida não estar sendo considerada no problema. Segue uma descrição detalhada dos passos do algoritmo.

Seja uma seqüência com n tarefas $\Pi = (\pi(1), \pi(2),..., \pi(j),..., \pi(n))$, onde $\pi(j)$ representa a tarefa alocada na posição j e seja $\pi(l)$ a última tarefa adiantada ou que termina exatamente na data de entrega. O APT é descrito a seguir:

Passo 1: *Programe as tarefas para começar seu processamento o quanto antes na primeira máquina* ($c_{j1} = c_{j-1,1} + p_{\pi(j),1}$, $j = 1,2,...n$ e $c_{01} = 0$).

Passo 2: *Programe as tarefas para começar seu processamento o quanto antes na segunda máquina* ($c_{j2} = \max [c_{j1}; c_{j-1,2}] + p_{\pi(j),2}$, $j = 1,2,...n$ e $c_{02} = 0$).

Passo 3: *Faça* $c_{l2} = \min [d; c_{\pi(l+1),2} - p_{\pi(l+1),2}]$ *e* $c_{j2} = c_{j+1,2} - p_{\pi(j),2}$, $i = l-1, l-2,...1$.

Passo 4: *Se* $l \geq n/2$, *faça* $l = \lceil n/2 \rceil$, $c_{l2} = d$ *e* $c_{j2} = \max [c_{j1}; c_{j-1,2}] + p_{\pi(j),2}$, $j = l+1, l+2,...n$.

Passo 5: *Calcule SDA = soma dos desvios absolutos.*

Passo 6: *Faça* Δ *= folga entre as primeiras duas tarefas da programação que possuem folga entre si na segunda máquina, ou seja,* $\Delta = c_{f+1,2} - p_{\pi(f+1),2} - c_{f2}$ *tal que* $\Delta > 0$ *e* $f = \min [l, l+1,...n]$. *Se não existir* Δ *que satisfaça essa condição* ($\Delta = 0$ *para todo f*), *vá para o Passo 9.*

Passo 7: *Faça u = posição da última tarefa adiantada. Se* $\Delta > d - c_{\pi(u),2}$, $\Delta = d - c_{\pi(u),2}$.

Passo 8: *Faça* $c_{j2} = c_{j2} + \Delta$, $j = 1, 2,...f$ *e calcule novaSDA = soma dos desvios absolutos. Caso novaSDA < SDA, faça SDA = novaSDA e volte para o Passo 6. Caso contrário, faça* $c_{j2} = c_{j2} - \Delta$, $j = 1, 2,...f$.

Passo 9: *Faça* $c_{j1} = c_{j2} - p_{\pi(j),2}$ *para* $j = 1, 2,...n$.

Passo 10: *Fim do algoritmo. SDA possui o valor da soma dos desvios absolutos.*

Proposição: *O APT fornece uma programação ótima para uma dada seqüência de tarefas.*

Prova: Seja uma seqüência $\Pi = (\pi(1), \pi(2),..., \pi(j),..., \pi(n))$, onde $\pi(j)$ representa a tarefa alocada na posição j. Suponha que $S = S(\Pi)$ seja a programação fornecida pelo APT para a seqüência Π. Observe que não existe tempo ocioso entre as tarefas que iniciam seus processamentos antes da data de entrega ($s_{j2} < d$) em S devido ao movimento realizado no Passo 3 do algoritmo.

Seja $\pi(l)$ a última tarefa não atrasada, logo $l \leq \lceil n/2 \rceil$, pois no Passo 4, se $l \geq \lceil n/2 \rceil$, a programação é atrasada de tal forma que $\pi(\lceil n/2 \rceil)$ seja finalizada exatamente na data de entrega. Note que os próximos passos somente tentam atrasar a programação, mantendo sempre a relação $l \leq \lceil n/2 \rceil$.

Seja f ($l \leq f \leq n$) a posição da última tarefa sem tempo ocioso antes do seu início na segunda máquina. No Passo 8, as tarefas são atrasadas de tal forma que se o tempo ocioso entre $\pi(f)$ e $\pi(f+1)$ for diminuído não existirá nenhum ganho na função objetivo. Isto significa que $\pi(\lfloor f/2 \rfloor + 1)$ completa seu processamento antes da data de entrega (Propriedade 1). Então, no programa final, $l \leq \lceil f/2 \rceil$. Um programa final S obtido pelo APT é ilustrado na Figura 10.

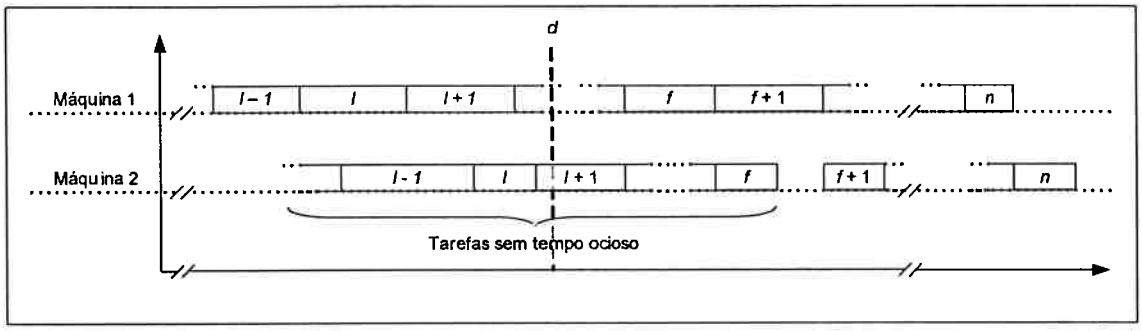

Figura 10. Programa final gerado pelo algoritmo APT.

Este programa será o programa ótimo S^* se não existir nenhum movimento de tarefas que gere ganho na função objetivo. Para obter algum ganho, deveria existir algum movimento que atrase mais tarefas adiantadas que não adiantadas ou que adiante mais tarefas atrasadas que não atrasadas.

Primeiramente, será analisado o possível ganho obtido com o adiantamento de tarefas. Esta análise será dividida em dois casos. O primeiro ocorre quando, após o Passo 3, temos $l < n/2$, isto é, menos que a metade das tarefas está adiantada e a Propriedade 4 já está obedecida. Com a execução do Passo 4, todas as tarefas atrasadas são programadas para

iniciar seu processamento na segunda máquina imediatamente após o seu processamento na primeira máquina, logo estas não podem ser adiantadas. Analisando os outros passos do método, nota-se que as únicas tarefas que poderiam ser adiantadas no final do algoritmo são as que foram atrasadas no Passo 8, e que para estas tarefas a verificação de se este atraso seria vantajoso já foi realizada, o que significa que adiantamentos não gerarão ganhos.

Consideraremos agora o caso no qual após o Passo 3 temos $l \geq n/2$. O movimento executado no Passo 4 resulta em um programa com $\lceil n/2 \rceil$ tarefas não atrasadas, garantindo a Propriedade 4. Além disso, todas as tarefas atrasadas são programadas para iniciar seu processamento na segunda máquina no tempo mínimo considerando as restrições físicas do problema. Logo, qualquer adiantamento no programa ou resultará em uma piora da função objetivo ou será inviável.

Será analisado agora o possível ganho obtido por atrasos nas tarefas. No programa final, existem no máximo $\lfloor n/2 \rfloor$ tarefas adiantadas, logo qualquer atraso no programa completo não poderá melhorá-lo. Se existir algum atraso que melhore a função objetivo deste programa, este deverá atrasar todas as tarefas adiantadas e um mínimo de tarefas atrasadas, ou seja, da tarefa $\pi(1)$ até a tarefa $\pi(f)$. Porém, este atraso já foi considerado no Passo 8 e nenhuma melhoria pôde ser obtida. Logo concluímos que nenhum movimento de adiantamento ou atraso de tarefas melhorará o programa obtido pelo algoritmo APT. ■

Experimentos Computacionais

Os programas computacionais foram escritos em linguagem C e os testes realizados em um Pentium IV - 2.4 GHz com 512 Mb RAM. Três heurísticas foram avaliadas, a primeira delas utiliza o procedimento de inserção de tarefas da heurística NEH (Nawaz *et al.*, 1983) enquanto a segunda utiliza um procedimento de busca local. A última é uma junção das anteriores buscando uma melhora na qualidade dos resultados. Cada uma das heurísticas foi denominada *Scheduling Heuristic* # (SH#), onde # representa o índice relativo à heurística de seqüenciamento inicial utilizada.

Inicialmente os testes foram realizados utilizando problemas pequenos, $n \leq 20$, gerados conforme sugerido por Sarper (1995). Resultados de melhor qualidade do que os apresentados por este autor foram obtidos. Devido à dimensão reduzida deste conjunto de

problemas, um novo *benchmark* com problemas de dimensão maior foi gerado utilizando a metodologia sugerida por Taillard (1993). Este autor propõe um algoritmo que gera números aleatórios inteiros respeitando uma distribuição uniforme dentro do intervalo [1;99] para serem utilizados como tempos de processamento em problemas de *flowshop*. Esses números são gerados independentemente da máquina utilizada, variando conforme uma semente inicial.

Este algoritmo foi utilizado com as mesmas sementes utilizadas por Taillard (1993). Como o autor propõe diversos ambientes *flowshop* com pelo menos cinco máquinas, selecionamos um grupo de sementes para cada dimensão, escolhendo sempre o grupo com o menor número de máquinas (20x5, 50x5, 100x5, 200x10 e 500x20). Estas sementes foram utilizadas para gerar problemas com duas máquinas e 20, 50, 100, 200 e 500 tarefas. Para cada dimensão temos 10 sementes, totalizando 50 problemas.

Sarper (1995) calcula as datas de entrega comuns em seus problemas através de números aleatórios. Esses números são gerados dentro de intervalos envolvendo a soma dos tempos de processamento das tarefas, e não se faz distinção entre datas restritas e irrestritas. Caso datas folgadas (irrestritas) fossem utilizadas, o problema poderia ser resolvido utilizando a seqüência gerada pelo algoritmo de Kanet (1981). Assim, a seqüência gerada por uma adaptação deste algoritmo (KanAd) foi utilizada para gerar um limitante superior da data de entrega comum restrita, procedendo da seguinte maneira:

- *O algoritmo de Kanet é aplicado utilizando os tempos de processamento na segunda máquina e uma data irrestrita (por exemplo,* $d = \sum_{i=1}^{n} p_{i1} + p_{i2}$*) para determinar uma seqüência ótima de tarefas;*
- *Obtida uma seqüência ótima, as tarefas são programadas na primeira máquina a partir do instante zero sem tempos ociosos entre tarefas;*
- *As tarefas são programadas na segunda máquina na mesma seqüência da primeira e sem tempos ociosos entre tarefas consecutivas, iniciando o processamento da primeira tarefa na segunda máquina logo após o término da última tarefa na primeira máquina;*

- *Calcula-se a folga F_j entre o processamento da tarefa na posição j entre as duas máquinas, ou seja, a diferença entre o início do processamento da tarefa na segunda máquina e seu término na primeira;*
- *Toda a programação na segunda máquina é adiantada subtraindo dos instantes de início de processamento de todas as tarefas o valor da menor folga encontrada;*
- *A data de entrega comum é definida como a data de término de processamento da tarefa na posição $\lceil n/2 \rceil$*

Através da metodologia descrita acima, determinamos a menor data com a qual se pode gerar a programação definida pelo algoritmo de Kanet (1981), além da programação ótima para o caso irrestrito. Esta data foi utilizada como limitante superior da data de entrega comum restrita a ser gerada. Grupos de problemas com datas de entrega iguais a essas datas multiplicadas pelo fator h = 0,2; 0,4; 0,6 e 0,8, a exemplo de Biskup & Feldmann (2001), foram gerados. Estes autores estudaram problemas de minimização do desvio ponderado em relação a uma data comum com uma máquina, e utilizaram estes mesmos multiplicadores sobre a somatória dos tempos de processamento das tarefas.

O desempenho das diversas heurísticas propostas foi avaliado através do índice de desvio relativo (IDR) utilizado por Kim (1993) e Ronconi (2004). O IDR é calculado como segue:

$$IDR = \frac{\text{Valor comparado} - \text{Melhor valor}}{\text{Pior valor} - \text{Melhor valor}}$$

Para cada problema é feita uma comparação do valor obtido por uma determinada heurística (valor comparado) com o menor (melhor) e maior (pior) valor encontrado por todas as heurísticas avaliadas para aquele problema. Assim, os melhores resultados correspondem aos menores valores de IDR. Comparou-se também o número de vezes que as heurísticas obtiveram o melhor valor dentre as heurísticas testadas. A heurística H3 de Sarper (1995), que apresenta os melhores resultados segundo esse autor, também foi avaliada. Os resultados são expostos a seguir após a descrição das heurísticas de seqüenciamento utilizadas.

Como primeira heurística, utilizamos o procedimento de inserção de tarefas da heurística NEH (Nawaz *et al.*, 1983), semelhante àquele utilizado por Sarper (1995): as tarefas são pré-ordenadas e em seguida inseridas em todas as posições possíveis da seqüência, mantendo-se a melhor seqüência parcial para a próxima iteração. Esse procedimento é denominado PNEH e o algoritmo APT será utilizado para comparar as seqüências parciais. As regras de ordenação inicial analisadas estão descritas na Tabela 17.

Tabela 17. Descrição das regras de ordenação inicial.

Regra Nº	Descrição
1	LPT utilizando os tempos de processamento na primeira máquina.
2	LPT utilizando os tempos de processamento na segunda máquina.
3	LPT utilizando a soma dos tempos de processamento nas duas máquinas.
4	SPT utilizando o tempo de processamento na primeira máquina.
5	SPT utilizando os tempos de processamento na segunda máquina.
6	SPT utilizando a soma dos tempos de processamento nas duas máquinas.
7	Ordem não-crescente do índice $[p_{i2} - p_{i1}]$.
8	Utilizando a regra de Johnson (Johnson, 1954), que separa as tarefas em dois grupos, sendo o primeiro composto daquelas com $p_{i1} < p_{i2}$ e o segundo das demais; no primeiro grupo, as tarefas são ordenadas em SPT usando p_{i1} e no segundo em LPT usando p_{i2}.
9	Utilizando a regra de seqüenciamento de Kanet (1981) com os tempos de processamento da segunda máquina.

O procedimento completo de ordenação inicial das tarefas e a aplicação do PNEH utilizando o algoritmo APT para avaliar e comparar as seqüências geradas será denominado SH1# (onde # representa o número da regra utilizada), conforme mostrado na Figura 11.

Figura 11. Estrutura de funcionamento de heurística SH1.

A Tabela 18 apresenta os valores médios de IDR para cada dimensão, assim como o número de melhores soluções obtidas por cada heurística.

Tabela 18. Comparação entre os IDR das heurísticas.

Nº. de Tarefas	Sarper	SH11	SH12	SH13	SH14	SH15	SH16	SH17	SH18	SH19
20	0.67	0.63	0.65	0.60	0.07	0.23	0.06	0.40	0.49	0.25
50	0.74	0.58	0.71	0.71	0.12	0.09	0.05	0.39	0.54	0.33
100	0.87	0.48	0.84	0.73	0.11	0.07	0.05	0.43	0.62	0.29
200	0.79	0.39	0.76	0.57	0.12	0.05	0.05	0.38	0.53	0.28
500	0.77	0.40	0.75	0.57	0.12	0.05	0.05	0.37	0.53	0.28
Média	0.77	0.50	0.74	0.64	0.11	0.10	0.05	0.40	0.54	0.29
Nº. de melhores soluções	0	1	0	0	32	62	83	0	0	24

Analisando os resultados, observa-se que a heurística que apresenta o melhor resultado em todos os critérios é a SH16. A SH15 e a SH14 obtiveram respectivamente o segundo e o terceiro melhor resultado, mostrando a importância de priorizarmos as tarefas com tempos de execução menores para sua ordenação inicial ao aplicarmos o PNEH, já que as heurísticas de melhor desempenho utilizam o critério SPT para ordenação inicial das tarefas. Observamos também que todas as heurísticas SH1# superaram a heurística de Sarper (1995).

O desempenho da heurística proposta SH16 será novamente avaliado tomando como base de comparação os resultados de Sarper, através da seguinte medida:

$$\textit{Diferença Percentual} = \frac{\textit{Resultado da heurística proposta} - \textit{Resultado da base de comparação}}{\textit{Resultado da base de comparação}} \times 100$$

Os resultados são apresentados na Tabela 18. Observa-se que um ganho médio de 12,54% foi obtido pela heurística SH16 e que tal ganho se mostrou maior para os problemas com um maior número de tarefas e datas de entrega mais restritas.

Tabela 18. Comparação entre os resultados da SH16 e os de Sarper.

		Diferença Percentual				Média
	h	0,2	0,4	0,6	0,8	
Número de Tarefas	20	-5.22	-12.62	-11.41	-10.35	-9.90
	50	-12.45	-14.68	-10.86	-8.29	-11.57
	100	-15.98	-16.64	-12.28	-9.05	-13.49
	200	-17.01	-17.10	-11.94	-9.73	-13.94
	500	-17.86	-17.64	-11.60	-8.10	-13.80
Média		-13.71	-15.73	-11.62	-9.10	-12.54

O tempo de processamento médio de um problema para cada dimensão utilizando a SH16 e a heurística de Sarper (1995) para problemas com 100 ou mais tarefas é apresentado na Tabela 19. Os tempos de processamento para os problemas com $n < 100$

foram omitidos por serem em sua maioria da ordem de milésimos de segundos ou inferiores. A heurística SH16 apresenta um tempo de processamento significantemente menor quando comparado ao tempo da heurística de Sarper. Os tempos da heurística de Sarper são elevados devido à utilização do modelo de programação linear, proposto pelo autor, ao seu término. Em um teste adicional, este modelo foi excluído e observou-se que os tempos de execução desta heurística se assemelham aos da heurística SH16.

Tabela 19. Comparação entre os tempos de CPU das heurísticas (em segundos).

Número de Tarefas\h	0.2	0.4	0.6	0.8
	SARPER			
100	1.60	1.60	2.20	3.30
200	25.0	25.0	28.0	44.0
500	519	563	752	1241
	SH16			
100	0.03	0.03	0.07	0.04
200	0.2	0.9	0.8	0.4
500	4	26	24	8

A segunda heurística avaliada consiste em ordenarmos as tarefas inicialmente segundo uma regra de prioridades e em seguida aplicarmos um algoritmo de busca local reduzida sobre a seqüência gerada, conforme apresentado na Figura 12.

Figura 12. Estrutura da heurística de busca local proposta.

O método proposto foi desenvolvido com base em experiências preliminares com algoritmos de busca local. Ao testarmos tais algoritmos utilizando a vizinhança de troca de duas tarefas, observamos que apesar dos resultados apresentarem uma melhora expressiva, o aumento no tempo de execução foi muito significativo. Com o intuito de obter soluções de qualidade com tempos razoáveis a vizinhança de busca foi reduzida gerando o algoritmo descrito a seguir.

Passo 1: *Faça $i = 1$ e MelhorCusto = custo da solução gerada pela regra de ordenação inicial avaliada pela APT.*

Passo 2: *Faça MelhorGanho = 0, $j = i + 1$*

Passo 3: *Troque as tarefas nas posições i e j. Faça NovoGanho = MelhorCusto – valor do custo da nova solução. Caso NovoGanho > MelhorGanho, k = j e MelhorGanho = NovoGanho.*

Passo 4: *Troque novamente as tarefas nas posições i e j e faça j = j + 1. Se j ≤ n, volte para o Passo 3. Caso contrário, vá para o Passo 5.*

Passo 5: *Se MelhorGanho > 0, troque as tarefas nas posições i e k faça MelhorCusto = MelhorCusto – MelhorGanho. Faça i = i + 1. Se i < n, volte para o Passo 2. Caso contrário termine o algoritmo.*

Com este algoritmo, buscou-se varrer toda a seqüência de tarefas uma única vez para cada posição avaliada, trocando a tarefa desta posição com aquela que apresentar o maior ganho para a função objetivo. Este algoritmo será denominado SH2# (onde # representa o número da regra de ordenação inicial utilizada, conforme a Tabela 17), e na Tabela 20 apresentamos uma comparação destes algoritmos utilizando a medida IDR.

Tabela 20. Comparação entre as heurísticas de busca local utilizando IDR.

Nº. de Tarefas	Sarper	SH21	SH22	SH23	SH24	SH25	SH26	SH27	SH28	SH29
5	0.68	0.18	0.34	0.64	0.32	0.78	0.06	0.36	0.65	0.26
6	0.74	0.25	0.26	0.49	0.29	0.74	0.00	0.33	0.60	0.17
10	0.76	0.24	0.24	0.51	0.28	0.73	0.00	0.29	0.60	0.19
15	0.77	0.23	0.24	0.51	0.29	0.72	0.00	0.30	0.57	0.17
20	0.75	0.27	0.24	0.37	0.28	0.73	0.00	0.32	0.60	0.19
Média	0.74	0.24	0.26	0.51	0.29	0.74	0.01	0.32	0.60	0.20
Nº. de melhores soluções	0	12	3	11	1	0	172	2	1	8

Novamente a regra de ordenação inicial de melhor desempenho foi a de número 6, desta vez apresentando um desempenho muito superior ao das outras regras tanto em termos de IDR quanto de número de melhores soluções obtidas. Comparando os resultados obtidos pela SH26 com a heurística original de Sarper (1995) em termo de desvio médio percentual, obtivemos o resultado da Tabela 21. Os tempos de processamento computacional são mostrados na Tabela 22. Como podemos notar, obtivemos um ganho em relação à SH16 tanto em termos de qualidade da solução quanto em tempo computacional na maioria dos casos (veja Tabelas 18 e 19).

Tabela 21. Comparação entre os resultados da SH26 e os de Sarper.

		Diferença Percentual				Média
	h	0.2	0.4	0.6	0.8	
Número de Tarefas	20	-3.77	-11.03	-11.46	-10.45	-9.18
	50	-11.25	-14.71	-14.33	-9.67	-12.49
	100	-14.70	-16.60	-15.32	-11.18	-14.45
	200	-15.91	-17.91	-17.08	-11.94	-15.71
	500	-17.28	-18.82	-16.92	-10.40	-15.85
Média		-12.58	-15.81	-15.02	-10.73	-13.54

Tabela 22. Tempos de CPU da SH26 (em segundos).

Número de Tarefas*h*	0.2	0.4	0.6	0.8
100	0.05	0.08	0.11	0.04
200	0.5	0.7	0.9	0.3
500	7	17	21	6

Devido aos bons resultados obtidos com as heurísticas SH16 e SH26 apresentamos a seguir a estrutura da heurística SH3 (Figura 13) que combina características dos métodos recém descritos.

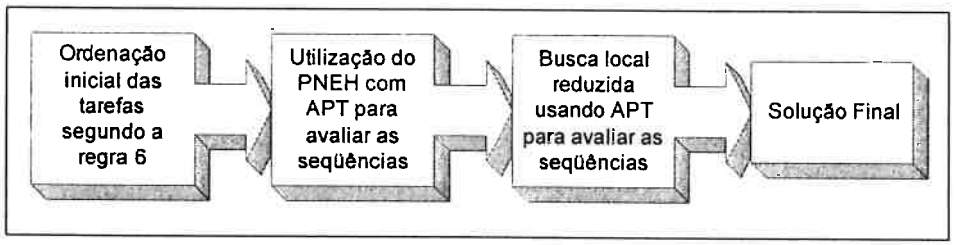

Figura 13. Estrutura de funcioanmento da heurística SH3.

Os resultados obtidos com a aplicação da SH3 em relação aos resultados da heurística de Sarper (1995) e os tempos computacionais são mostrados nas Tabelas 23 e 24. Observando estas tabelas, nota-se uma melhora significativa na qualidade dos resultados. Apesar do aumento do tempo computacional, este ainda se mantém nos padrões esperados para uma heurística, demorando pouco mais de um minuto para os problemas maiores.

Tabela 23. Ganho da SH3 em relação a Sarper.

		Diferença Percentual				Média
	h	0.2	0.4	0.6	0.8	
Número de Tarefas	20	-6.12	-13.61	-13.00	-12.64	-11.34
	50	-12.97	-16.18	-14.03	-10.29	-13.37
	100	-16.23	-17.78	-15.52	-10.85	-15.09
	200	-17.26	-18.72	-16.14	-11.72	-15.96
	500	-18.00	-19.31	-15.78	-10.16	-15.81
Média		-14.12	-17.12	-14.90	-11.13	-14.31

Tabela 24. Tempos de CPU da heurística SH3 (em segundos).

Número de Tarefas\h	0.2	0.4	0.6	0.8
100	0.07	0.07	0.19	0.11
200	0.6	1.7	2.4	1.2
500	10	49	75	30

Podemos resumir os principais resultados deste trabalho como segue. Propriedades do problema de programação de tarefas no ambiente *flowshop* com duas máquinas visando à minimização da SAD sobre uma data de entrega comum foram apresentadas. Em seguida, um algoritmo de programação (APT) que encontra uma programação ótima para uma seqüência de tarefas foi proposto. Esse algoritmo, utilizado em conjunto com o procedimento de inserção PNEH e com um algoritmo de busca local, permitiu a elaboração de um grupo de heurísticas que diferem entre si na regra de ordenação inicial das tarefas. Os resultados destas heurísticas foram comparados entre si e com a heurística de melhor desempenho de Sarper (1995) utilizando um conjunto de problemas com até 500 tarefas e datas de entrega geradas pelo algoritmo KanAd. Tal metodologia pode ser facilmente reproduzida e utilizada como um *benchmark* para trabalhos futuros.

Melhores resultados foram obtidos com a heurística de busca local SH26, com um ganho médio de 13,54% em relação à heurística de Sarper, além de um tempo computacional muito inferior. Por fim, combinamos o PNEH com a busca local, obtendo um ganho de 14,31% sobre a H3 proposta por Sarper (1995). Obtivemos assim duas opções de heurísticas com bom desempenho: a SH26 com um tempo computacional reduzido e a SH3 com um tempo um pouco maior, mas um ganho em termos de qualidade da solução.

CAPÍTULO 3. PROBLEMAS DE EMPACOTAMENTO

O problema de empacotar um dado conjunto de peças dentro de regiões definidas de forma a maximizar o número de peças ocorre em uma ampla classe situações práticas, por exemplo nas atividades logísticas de movimentação, armazenagem e transporte de materiais, ao carregar produtos embalados (itens) sobre paletes ou dentro de contêineres (objetos). Parte da motivação para estudar estes problemas está relacionada à importância técnica e econômica das aplicações práticas. Com o aumento da prática de unitização de carga na armazenagem e distribuição física de produtos, pequenos ganhos no aproveitamento da superfície de paletes, ou do espaço interno de contêineres, tornam-se significativos ao envolverem toda a cadeia logística. Diversos artigos foram publicados na literatura abordando problemas de empacotamento. Para uma ampla revisão da literatura veja os trabalhos de Dyckhoff *et al.*(1997) e Lodi *et al.* (2003).

Nas próximas seções descreveremos trabalhos recentes que lidam com o problema de empacotar itens dentro de objetos utilizando heurísticas baseadas em subproblemas de programação não linear. Estes trabalhos foram desenvolvidos em colaboração com os Profs. J.M. Martínez (IMEC-Unicamp) e E.G. Birgin (IME-USP) além de outros co-autores. Em Birgin *et al.* (2005a) um procedimento baseado em um problema de decisão não linear é proposto para o empacotamento de cilindros idênticos em um contêiner retangular. Com base no mesmo procedimento Birgin *et al.* (2005b) abordaram o problema de empacotamento de retângulos ortogonais em regiões convexas.

3.1. Cilindros Idênticos em Objetos Retangulares

O problema de carregamento de contêineres é um problema tridimensional que consiste em arranjar itens de diferentes tamanhos dentro de objetos maiores de tal forma que a função objetivo (em geral, o volume carregado) seja otimizada. Este problema possui importantes aplicações comerciais e industriais. As mercadorias são empacotadas em contêineres de tamanho padrão para facilitar o seu manuseio de forma que estes possam ser

transportados por navios, caminhões ou trens com o mínimo prejuízo. Um aumento no número de itens em um contêiner acarreta uma diminuição nos custos e por tal razão o problema de otimização associado possui importância econômica.

Em Birgin *et al.* (2005a) aborda-se o problema do carregamento mais denso de cilindros idênticos em um contêiner. Todos os cilindros possuem a mesma altura, conseqüentemente, o problema pode ser resolvido como um problema bidimensional onde o método de solução deve apresentar a posição das bases circulares dos cilindros dentro da base retangular do contêiner.

No trabalho de Isermann (1991) algoritmos para empacotar círculos idênticos com padrões geométricos foram propostos, enquanto uma aplicação prática deste problema pode ser encontrada em Fraser & George (1994). Estes autores apresentam algoritmos heurísticos, também baseados em padrões, para o carregamento de bobinas de papel. Em Correia *et al.* (2000) algoritmos baseados na meta-heurística *Simulated Annealing* são descritos. Além disso, os mesmos autores em Correia *et al.* (2001) propõem um novo limitante superior para este problema. Em George *et al.* (1995) diversos procedimentos heurísticos são analisados para o empacotamento de cilindros de diferentes diâmetros dentro de um contêiner. Uma formulação inteira mista não linear é proposta porém os autores alertam que o esforço computacional associado a este modelo é excessivo.

Birgin *et al.* (2005a) propõem um procedimento baseado em um problema de decisão não linear para resolver o problema de empatar cilindros idênticos. A formulação deste problema é baseada no fato de que os centros dos cilindros devem estar dentro da base retangular do contêiner (um raio distante das bordas) e distantes um dos outros de no mínimo um diâmetro. O procedimento busca encontrar o máximo número de cilindros que respeitam estas restrições. Um estudo comparativo com outros trabalhos da literatura é realizado. Os principais resultados deste trabalho são apresentados a seguir.

Problema de Decisão

O problema de decisão considerado é: *Dados k círculos de raio r e uma base retangular de dimensão d_1 x d_2 determinar se é possível ou não alocar todos os círculos dentro desta base.* Um modelo não linear é introduzido para a resolução deste problema.

Desejamos alocar k círculos de raio r dentro de uma caixa retangular $[0, d_1]$x $[0, d_2]$ de tal forma que a intersecção entre qualquer par de círculos i e j, $i \neq j$, seja de no máximo um ponto, *i.e.*, os círculos não estão sobrepostos. Logo dados k, r, d_1 e d_2, o objetivo é determinar $p^1,\ldots, p^k \in \{r, d_1 - r]$ x $[r, d_2 - r]$ resolvendo o seguinte problema:

$$
\begin{array}{lll}
\textit{minimizar} & \sum_{i \neq j} max(0,(2r)^2 - \|p^i - p^j\|_2^2)^2 & \\
\textit{sujeito a} & r \leq p_1^i \leq d_1 - r & i = 1,\ldots, k, \\
 & r \leq p_2^i \leq d_2 - r & i = 1,\ldots, k.
\end{array}
$$

Os pontos $p^1,\ldots, p^k$ são os centros dos círculos que desejamos encontrar. p_1^i e p_2^i representam a abscissa e a ordenada de $p^i \in \Re^2$, $i = 1, \ldots, k$. Se acharmos $p^1,\ldots, p^k$ tais que o valor da função objetivo é zero então a resposta para o problema de decisão é SIM, senão a resposta é NÃO.

Para resolver este problema, o método GENCAN (Birgin & Martínez, 2002) foi utilizado. Este método encontra pontos estacionários de primeira ordem (muito provavelmente minimizadores locais) para o problema recém formulado. Para aumentar a probabilidade de encontrar minimizadores globais, este método é reiniciado partindo de N soluções geradas de forma aleatória. Se uma solução com custo ótimo zero é encontrada então a resposta para o problema de decisão é SIM, senão a resposta é "não sabemos" e assume-se o NÃO. O fluxograma da Figura 14 descreve o funcionamento desta estratégia.

Com o procedimento ilustrado na Figura 14 é possível responder se é possível ou não empacotar k círculos dentro da base. Para alocar o máximo de círculos possíveis a seguinte estratégia é aplicada. Inicialmente, tenta-se empacotar somente um círculo ($k = 1$) e pergunta-se se é possível ou não empacotar tal número de círculos. Se a resposta for SIM, $k = k + 1$ e repete-se a pergunta. Se a resposta for NÃO, o procedimento termina. Note que não é necessário começar com $k = 1$, se é conhecido que a resposta do problema de decisão para, por exemplo $k = \bar{k}$ é SIM, neste caso, é suficiente iniciar com $k = \bar{k} + 1$.

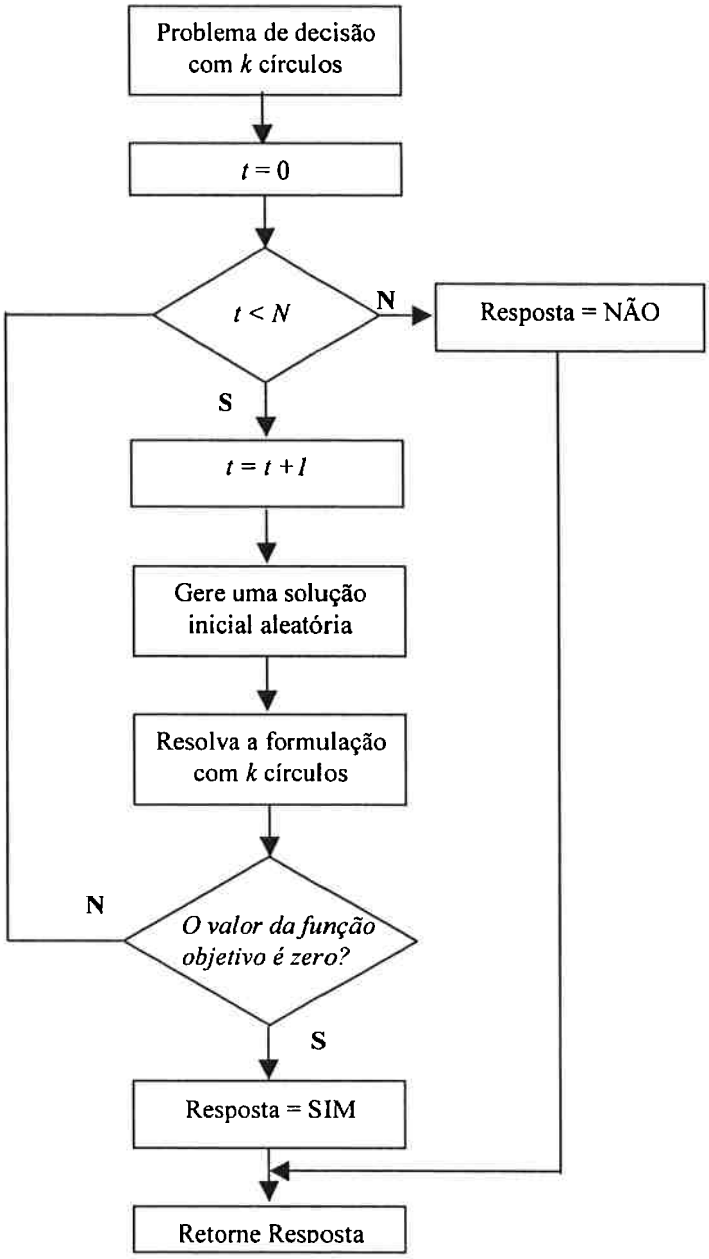

Figura 14. Resolvendo o problema de decisão.

Experimentos Computacionais

Os programas computacionais foram escritos em linguagem Fortran 77 e os testes realizados em uma estação de trabalho Sun SparcStation 20 - 70 MHz com 128 Mb RAM. Buscando o minimizador global dos problemas de decisão, o algoritmo GENCAN foi executado a partir de 50.000 diferentes soluções iniciais (N = 50.000) ou até encontrar o minimizador global (valor da função objetivo nulo).

O primeiro conjunto de problemas teste consiste de 3 problemas extraídos do trabalho de Isermann (1991). A descrição dos problemas, o número de círculos empacotados pelos métodos desenvolvidos por Isermann e pelo método proposto (denominado doravante de GENPACK) estão apresentados na Tabela 25. Nesta tabela, *NDPS* significa ``número de problemas de decisão resolvidos" e *Tempo* é o tempo de CPU em segundos utilizado para resolver todos os problemas de decisão. As outras colunas são auto-explicativas. Observe que, para o terceiro problema, GENPACK foi capaz de empacotar mais círculos que o método de Isermann. A Figura 15 mostra a solução encontrada. Note que nenhum padrão definido pode ser detectado e que este tipo de empacotamento não é considerado no trabalho de Isermann.

Tabela 25. Desempenho de GENPACK para os problemas apresentados em Isermann.

Problema			***Número de círculos empacotados***		***GENPACK***	
Nome	***Dimensão da base***	***Raio do círculo***	***Isermann***	***GENPACK***	***NDPS***	***Tempo***
1	1200 x 800	102	22	22	5	0,32
2	1200 x 800	101	23	23	32	3,51
3	471 x 196	14	124	**126**	289	1824,85

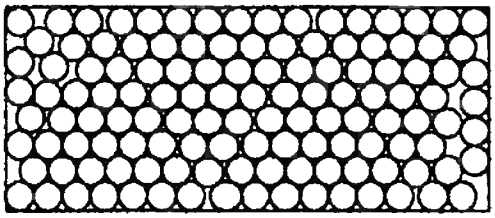

Figura 15. 126 círculos de raio 14 empacotados em uma base de dimensão 471 x 196.

O segundo conjunto de problemas foi extraído de Correia *et al.* (2000, 2001) e está descrito na Tabela 26. Como mencionado por estes autores, as instâncias foram geradas de forma a serem representativas de possíveis relações entre a base retangular e a área do círculo: nos problemas 1.x, 2.x, 3.x, 4.x e 5.x, a área do círculo é aproximadamente 1%, 2,5%, 5%, 10% e 15% da área da base retangular, respectivamente. Em termos do formato da base, os problemas x.1, x.2, x.3 são retangulares longos, os problemas x.4, x.5, x.6 são retangulares curtos e os problemas x.7, x.8, x.9 são quadrados.

Como podemos notar na Tabela 26, vários resultados para problemas nos subconjuntos 1.x e 2.x foram melhorados. Estes conjuntos, aparentemente, são os mais difíceis, como mencionado em Correia *et al.* (2000). A razão pela qual nenhuma solução dos subconjuntos 3.x, 4.x e 5.x (problemas considerados mais fáceis) foi melhorada é devido a, provavelmente, o número de círculos empacotados em Correia *et al.* (2000) ser o ótimo. A Figura 16 mostra o posicionamento dos círculos para os 6 problemas nos quais a metodologia proposta encontrou melhores valores da função objetivo.

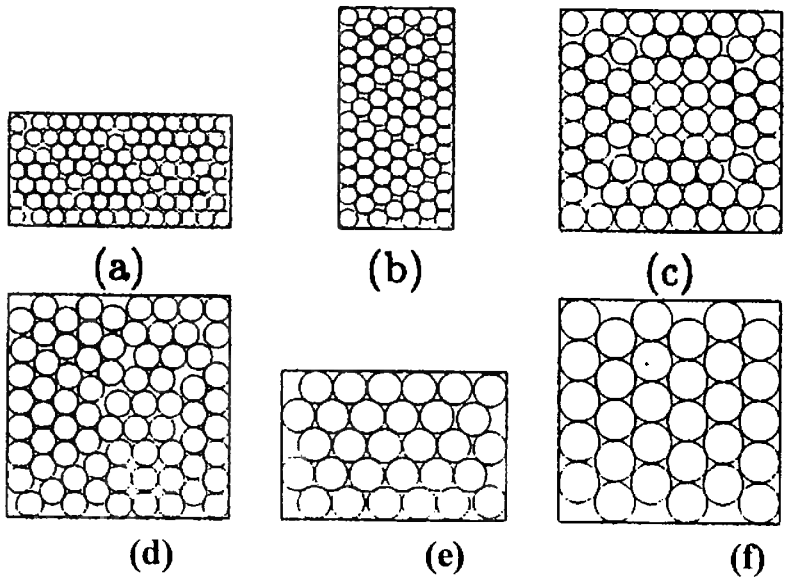

Figura 16. Problemas (a), (b), (c), (d), (e) e (f) correspondem aos problemas 1.1, 1.3, 1.8, 1.9, 2.5 e 2.9 extraídos de Correia et al. (2000).

Tabela 26. Desempenho de GENPACK para os problemas apresentados em Correia et al. (2000).

Problema			***Número de círculos empacotados***		***GENPACK***	
Nome	***Dimensão da base***	***Raio do círculo***	***Correia et al.***	***GENPACK***	***NDPS***	***Tempo***
1.1	160 x 80	6	90	**91**	61	734,05
1.2	100 x 200	8	84	84	4304	5791,83
1.3	120 x 240	10	73	**74**	2723	4065,91
1.4	100 x 80	5	86	86	27987	37108,39
1.5	120 x 80	6	68	68	16	23,35
1.6	120 x 100	6	87	87	864	2273,14
1.7	80 x 80	5	68	68	16794	12336,67
1.8	100 x 100	6	70	**71**	80	225,57
1.9	120 x 120	7	73	**74**	5	734,05
2.1	160 x 80	10	32	32	2	0,67
2.2	100 x 200	13	29	29	11	2,51
2.3	120 x 240	15	32	32	1	0,42
2.4	100 x 80	8	32	32	2016	232,45
2.5	120 x 80	9	28	**30**	25	3,12
2.6	120 x 100	10	30	30	12	1,42
2.7	80 x 80	7	32	32	10	7,91
2.8	100 x 100	9	30	30	1	0,18
2.9	120 x 120	11	29	**30**	33	11,21
3.1	160 x 80	14	15	15	2	0,02
3.2	100 x 200	18	15	15	26	0,41
3.3	120 x 240	21	15	15	1	0,01
3.4	100 x 80	11	16	16	30	0,75
3.5	120 x 80	12	15	15	1	0,01
3.6	120 x 100	14	14	14	17	0,30
3.7	80 x 80	10	16	16	2	0,03
3.8	100 x 100	13	13	13	1	0,02
3.9	120 x 120	15	16	16	2	0,04
4.1	160 x 80	20	8	8	1	0,04
4.2	100 x 200	25	8	8	1	0,11
4.3	120 x 240	30	8	8	1	0,11
4.4	100 x 80	16	6	6	1	0,00
4.5	120 x 80	17	7	7	1	0,00
4.6	120 x 100	20	6	6	1	0,01
4.7	80 x 80	14	6	6	1	0,00
4.8	100 x 100	18	6	6	1	0,00
4.9	120 x 120	21	6	6	1	0,00
5.1	160 x 80	25	3	3	1	0,00
5.2	100 x 200	31	3	3	1	0,00
5.3	120 x 240	37	3	3	1	0,00
5.4	100 x 80	19	4	4	1	0,00
5.5	120 x 80	21	4	4	1	0,00
5.6	120 x 100	24	4	4	1	0,00
5.7	80 x 80	17	4	4	1	0,00
5.8	100 x 100	22	4	4	1	0,00
5.9	120 x 120	26	4	4	1	0,00

Uma Extensão: Empacotamento de Círculos em Círculos

A metodologia proposta pode ser estendida para o empacotamento de círculos idênticos em um círculo de raio Δ no lugar de bases retangulares. O problema de decisão é então definido por:

$$\begin{array}{lll} \textit{minimizar} & \sum_{l \neq j} max(0,(2r)^2 - \|p^i - p^l\|_2^2)^2 & \\ \textit{sujeito a} & \|p^i\|_2^2 \leq (\Delta - r)^2 & i = 1,\ldots, k, \end{array}$$

Este problema no lugar de possuir limites nos valores das variáveis possui restrições que formam um conjunto convexo. Neste caso, o método GENCAN não é aplicável pois o mesmo é apropriado para restrições do tipo "caixa". Para o novo modelo considerado utilizamos o método SPG (Birgin *et al.* 2001) que trata de problemas não lineares com regiões viáveis convexas.

Em Graham *et al.* (1998) um problema similar é considerado. O raio da base circular Δ e o número de círculos que devem ser empacotados são fixados e o problema consiste em determinar o raio máximo de círculos idênticos que podem ser empacotados. O raio ótimo r_l* para este problema com $l = 1,\ldots,65$ círculos e os centros dos l círculos dentro da base circular de raio unitário são conhecidos na literatura (veja por exemplo Graham *et al.* 1998). A solução desde problema fornece um limitante inferior para o problema de maximização do número de itens empacotados. Observe que estes problemas não são equivalentes dado que sete círculos de raio r_6* podem ser empacotados em uma base circular de raio Δ (veja Graham *et al.* 1998).

A Tabela 27 mostra as principais características de três problemas extraídos de Graham *et al.* (1998) e o desempenho de GENPACK, enquanto a Figura 17 ilustra a estrutura dos empacotamentos encontrados.

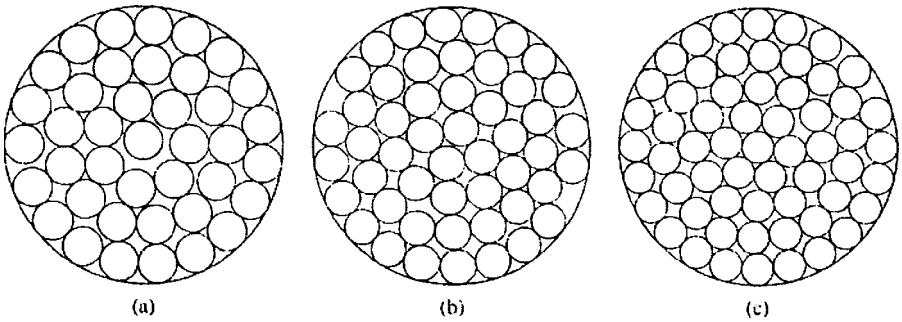

Figura 17. Exemplos de empacotamentos de círculos em círculos.

Tabela 27. Desempenho de GENPACK para o empacotamento de círculos em círculos.

Problema			***Número de círculos empacotados***		***GENPACK***	
Nome	***Raio da base circular***	***Raio do círculo***	***Graham et al.***	***GENPACK***	***NDPS***	***Tempo***
1	1,1632960610	0,1632960610	40	40	1	5,23
2	1,1439363515	0,1439363515	50	50	332	456,11
3	1,1307835795	0,1307835795	60	60	155	363,62

Considerações finais

Os resultados numéricos mostraram que a metodologia proposta, baseada em um problema de decisão não linear, é uma abordagem promissora. Deve ser observado que, a princípio, o problema não considera a complexidade geométrica da solução obtida. Se a aplicação prática impõe restrições à distribuição dos círculos, estas restrições devem ser consideradas através de reformulação adequada do modelo não linear.

O modelo introduzido para empacotar círculos em retângulos e círculos em círculos também pode ser expandido para considerar k círculos diferentes com raios r^1, r^1,..., r^k como segue:

$$
\begin{array}{lll}
\textit{minimizar} & \sum_{i \neq j} max(0,(r^i + r^j)^2 - \|p^i - p^j\|_2^2)^2 & \\
\textit{sujeito a} & r^i \leq p_1^i \leq d_1 - r^i & i = 1,\ldots, k, \\
 & r^i \leq p_2^i \leq d_2 - r^i & i = 1,\ldots, k.
\end{array}
$$

e

$$
\begin{array}{lll}
minimizar & \sum_{i \neq j} max(0,(r^i + r^j)^2 - \|p^i - p^j\|_2^2)^2 & \\
sujeito\ a & \|p^i\|_2^2 \leq (\Delta - r)^2 & i = 1,\ldots, k,
\end{array}
$$

O esquema proposto neste trabalho para resolver os problemas de decisão recém apresentados é aplicável. Por outro lado, não é claro quais problemas de decisão devem ser resolvidos de forma a maximizar a área utilizada. Como um trabalho futuro, heurísticas adequadas devem ser desenvolvidas.

3.2. Retângulos Ortogonais em Convexos Arbitrários

Um dos problemas mais básicos nesta área é o de encontrar o número máximo de retângulos que podem ser ortogonalmente empacotados em um retângulo maior. Embora para a versão guilhotinada deste problema existam algoritmos polinomiais (Tarnowski *et al.*, 1994), para a versão não-guilhotinada existem fortes conjecturas sobre a sua alta complexidade (Dowsland, 1987; Morabito & Morales, 1998). Lins *et al.* (2003) elaboraram uma heurística eficiente para tratar este problema e obtiveram soluções para instâncias difíceis que não haviam sido anteriormente resolvidas na literatura. Além disso, com o intuito de aprimorar o desempenho do método uma nova implementação foi proposta em Birgin *et al.* (2005d).

Uma formulação não linear para o problema bidimensional de corte não-guilhotinado restrito (existem limitações quanto ao número de retângulos de cada tipo que pode ser empacotado) foi apresentada em Beasley (2004), onde o modelo foi utilizado para a definição de uma heurística populacional. O Método dos Sentinelas para empacotar polígonos dentro de objetos arbitrários foi introduzido por Birgin *et al.* (2005c) e é baseado em análise convexa e em formulações não lineares para analisar sobreposições. Modelos não lineares foram também utilizados com sucesso para outros problemas de empacotamento, como por exemplo o empacotamento de moléculas com uma distância mínima entre átomos (Martínez & Martínez, 2003) e o empacotamento de círculos idênticos ou diferentes em diversos tipos de regiões (Birgin *et al.* 2005a, Mladenovic *et al.*2005, Stoyan & Yaskov 2004, Wang *et al.* 2002).

No trabalho apresentado por Birgin *et al.* (2005b) considera-se o problema de empacotamento de retângulos em uma região arbitrária, geralmente convexa. Os lados dos retângulos devem estar paralelos aos eixos de coordenadas e o carregamento não está restrito a padrões guilhotinados. A estratégia para resolver este problema está baseada no fato de que a pergunta *É possível empacotar m retângulos?* pode ser resolvida através da resolução de um número finito (embora possível alto) de problemas de otimização não linear. Logo, a estratégia para empacotar o maior número possível de retângulos consiste em responder a questão apresentada para valores crescentes de m. Os principais resultados do trabalho de Birgin *et al.* (2005b) se encontram resumidos a seguir.

Abordagem não linear

Em Birgin *et al.* (2005b) considera-se o problema de empacotamento do número máximo de retângulos idênticos permitindo rotações ortogonais. Seja Ω um subconjunto convexo de $\Re^2$. Desejamos empacotar m retângulos de tal forma que estes estejam contidos em Ω e o interior da intersecção de qualquer par de retângulos diferentes seja vazio. Dado que Ω é convexo, o fato dos vértices de um retângulo estarem contidos em Ω é suficiente para garantir que o retângulo esteja contido em Ω. Além disso, os eixos de todos os retângulos devem estar paralelos, logo, consideraremos que estes estarão paralelos aos eixos x e y de $\Re^2$. Esta consideração não representa uma perda de generalidade se o problema original considerado possuir tal restrição.

Considere o problema *Fixed-Orientation-Pack* (FOP) que consiste em empacotar um conjunto de m retângulos em uma região Ω dada. Rotações não são permitidas. Este problema é a base para o procedimento geral. Para todo $i = 1,...,m$, seja $C^i = (c_1^i, c_2^i)$ o centro do retângulo $R^i = (a^i, b^i)$ e sejam a^i, $b^i > 0$ os valores dos lados horizontal e vertical, respectivamente. Sejam:

$$V_{sw}^i \equiv V_{sw}^i(R^i, C^i) = (c_1^i - a^i/2, c_2^i - b^i/2),$$

$$V_{se}^i \equiv V_{se}^i(R^i, C^i) = (c_1^i + a^i/2, c_2^i - b^i/2),$$

$$V_{nw}^i \equiv V_{nw}^i(R^i, C^i) = (c_1^i - a^i/2, c_2^i + b^i/2),$$

$$V_{ne}^i \equiv V_{sw}^i(R^i, C^i) = (c_1^i + a^i/2, c_2^i + b^i/2),$$

os quatro vértices de R^i com centro em C^i. Doravante nos referiremos aos vértices como V_k^i para todo $k \in D = \{sw, se, ne, nw\}$. Logo, o problema FOP pode ser definido como segue:

Dado um conjunto convexo $\Omega \subset \Re^2$ e um conjunto de m retângulos $R^i = (a^i, b^i)$, $i = 1,...,m$, encontre $C^i \in \Re^2$, $i = 1,...m$ tais que:

1. *Para todo $i = 1,...,m$ e $k \in D$,*
$$V_k^i \in \Omega .$$
2. *Para todo $i,j = 1,...,m$, $i \neq j$,*
$$\left|c_1^i - c_1^j\right| \geq (a^i + a^j)/2 \quad ou \quad \left|c_2^i - c_2^j\right| \geq (b^i + b^j)/2 .$$

A condição 1 implica que os vértices de R^i estão em Ω. O fato de ao menos uma das expressões da condição 2 ser satisfeita significa que não existe sobreposição entre os retângulos R^i R^j. Podemos obter uma solução para o FOP através da resolução do seguinte modelo não linear:

$$minimizar \quad \bar{f}(C^1,...,C^m) = \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\left[max\left\{0, \frac{(a^i + a^j)^2}{4} - (c_1^i - c_1^j)^2\right\}^2 . \, max\left\{0, \frac{(b^i + b^j)^2}{4} - (c_2^i - c_2^j)^2\right\}^2\right]$$

$$sujeito\ a \quad V_k^i \in \Omega \qquad i = 1,...,m \text{ e } k \in D$$

O problema FOP consiste em encontrar $C^1,...,C^m \in \Re^2$ tal que a função objetivo tenha valor nulo respeitando as restrições impostas. Considere o caso onde Ω é dado por $\Omega = \Omega_1 \cap \Omega_2$, onde Ω_1 é um conjunto simples e $\Omega_2 = \{x \in \Re^2 \mid g(x) \leq 0\}$ não. As restrições $V_k^i \in \Omega$ podem ser dividas em: $V_k^i \in \Omega_1$ e $V_k^i \in \Omega_2$.

Notemos que encontrar $C^1,...,C^m$ que satisfaçam as restrições $V_k^i \in \Omega_2$ é equivalente a encontrar $C^1,...,C^m$ que anulem $\sum_{i=1}^{m}\sum_{k \in D} max\{0, g(V_k^i)\}^2$. Logo o modelo anterior pode ser reformulado como segue:

$$\begin{array}{ll} \textit{minimizar} & \bar{f}(C^1,\ldots,C^m)+\sum_{i=1}^{m}\sum_{k\in D} max\{0, g(V_k^i)\}^2 \\ \textit{sujeito a} & V_k^i \in \Omega_1 \qquad i=1,\ldots,m \text{ e } k \in D \end{array}$$

Neste estudo resolvemos este modelo considerando que $\Omega = \Omega_1 \cap \Omega_2$, onde Ω_1 consiste das restrições de caixas em Ω e Ω_2 representa todas as outras restrições. Logo, o problema tratado se transforma em um problema de otimização não linear com restrições de caixa e o método GENCAN (Birgin & Martínez, 2002) é uma boa opção para resolvê-lo.

Permitindo rotações de 90 graus

Em diversas aplicações práticas, o retângulo R^i com o lado horizontal a^i e o lado vertical b^i pode ser considerado equivalente ao retângulo com o lado horizontal b^i e o lado vertical a^i. Isto significa que rotações de 90 graus são permitidas para efeitos de empacotamento. Suponha que, para todo $i = 1,\ldots,m$, escolha-se que o lado horizontal de R^i será a^i ou b^i. Se definida esta opção e o correspondente problema FOP for resolvido e um valor nulo da função objetivo for encontrado, o problema de empacotamento com rotações ortogonais está resolvido.

Claramente, existem 2^m possíveis opções, porém este número pode ser consideravelmente reduzido. Por exemplo, se $a^i = a$, $b^i = b$ para todo $i = 1,\ldots,m$, então o número de possibilidades diferentes é reduzido a $m+1$. Neste caso, para todo $p = 0, 1,\ldots,m$, podemos supor que p retângulos possuem o lado horizontal igual a a e o lado vertical igual a b e m-p retângulos possuem o lado horizontal igual a b e o lado vertical igual a a. Logo, teremos $m+1$ problemas do tipo FOP para serem resolvidos. Se na resolução de um deles encontramos uma solução com valor nulo da função objetivo então o problema de empacotamento de m retângulos está resolvido. O problema *Orthogonal-Rotations-Pack* (ORP) pode ser definido como segue:

Dado um conjunto convexo $\Omega \subset \Re^2$ e um conjunto de m retângulos idênticos, $R^i \equiv R^i(a,b)$, encontre $C^1,\ldots,C^m \in \Re^2$ e p $(0 \leq p \leq m)$ tais que $C^1,\ldots,C^m$ seja uma solução do problema

FOP defindo pelos retângulos $\bar{R}^i \equiv \bar{R}^i(a,b)$, $i = 1,...,p$, $\bar{R}^i \equiv \bar{R}^i(b,a)$, $i = p+1,...,m$ *e pelo conjunto* Ω.

<u>*Empacotando o máximo de retângulos possíveis*</u>

Dado um conjunto convexo Ω e um número ilimitado de retângulos idênticos, o problema de empacotar tantos retângulos quando possível (permitindo rotações de 90 graus) é equivalente a resolver o problema ORP para o maior valor possível de m. Para um valor fixo de m, resolver o problema ORP é equivalente a resolver um dos $m+1$ problemas FOP associados. Resolver um destes problemas FOP é equivalente a encontrar a solução ótima global do modelo de programação não linear associado com valor nulo para a função objetivo. A maioria dos métodos de otimização não linear encontra pontos estacionários de primeira ordem (muito provavelmente ótimos locais) para o modelo apresentado. Uma alternativa para aumentar a probabilidade de convergência para um ótimo global é resolver o modelo partindo de diferentes soluções inicias.

Na prática, primeiro estabelecemos um tempo máximo de CPU, T. Se este tempo máximo é atingido, o algoritmo proposto é finalizado e o melhor empacotamento encontrado é apresentado como resultado final, ou seja, $m = k-1$ itens são empacotados. Enquanto o tempo máximo não é atingido, os seguintes passos são seguidos:

Passo 1: *Seja* $k = m_{lb}$, *onde* m_{lb} *é um limitante inferior conhecido do número de retângulos que podem ser empacotados.*

Passo 2: *Para resolver o problema ORP de empacotar k retângulos, selecionamos aleatoriamente um número inteiro* $p \in \{0, 1,...,k\}$ *e tentamos resolver o problema FOP associado. Para resolver o problema FOP associado partimos de uma solução inicial aleatória com a esperança de encontrar o ótimo global.*

Passo 3: *Se uma solução com valor nulo para a função objetivo foi encontrada e* $k = m_{ub}$, *onde* m_{ub} *é um limitante superior conhecido do número de retângulos que podem ser empacotados, o algoritmo é finalizado.*

Passo 4: *Se uma solução com valor nulo para a função objetivo foi encontrada e* $k < m_{ub}$ *então incremente k de uma unidade e volte ao Passo 2. Neste caso o problema*

ORP com k retângulos foi resolvido e tentaremos resolver com um retângulo a mais.

***Passo 5**: Se uma solução com valor nulo para a função objetivo não foi encontrada, retornamos ao Passo 2. Neste caso o problema ORP que estamos tentando resolver continua o mesmo e simplesmente tentamos resolvê-lo escolhendo outro número aleatório p de retângulos horizontais e outra solução inicial.*

Experimentos Computacionais

Os programas computacionais foram escritos em linguagem Fortran 77 e os testes realizados em um 1.8 GHz AMD Opteron 244 com 2 Gb RAM. Consideramos o conjunto de problemas de empacotamento de retângulos em regiões convexas arbitrárias introduzido em Birgin *et al.* (2005c), onde a possibilidade de rotações arbitrárias foi considerada. Além disso, consideramos um interessante problema onde a região convexa é dada pela curva Lamé. Maiores detalhes sobre os problemas considerados podem ser encontrados em Birgin *et al.* (2005b).

Nos experimentos utilizamos como tempo máximo de CPU $T = 6$ horas. Dado que para regiões convexas arbitrárias não é fácil encontrar bons limitantes, fixamos $m_{lb}=1$ e m_{ub} igual à área da região dividida pela área do retângulo. A Tabela 28 apresenta, para cada problema, o número de retângulos que são empacotados em Birgin *et al.* (2005c) permitindo rotações arbitrárias e o número de retângulos empacotados neste estudo considerando somente empacotamentos ortogonais. Esta tabela também mostra o número total de problemas não lineares resolvidos e o tempo de CPU gasto para que a melhor solução fosse encontrada. A Figura 18 ilustra as soluções obtidas.

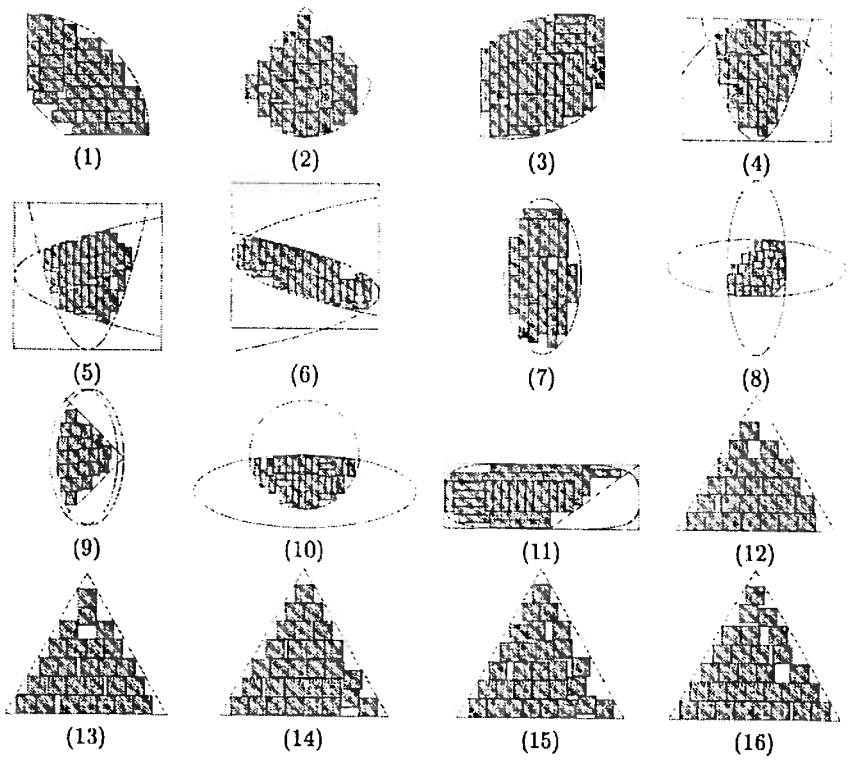

Figura 18. Representação gráfica das soluções.

Tabela 28. Desempenho do procedimento não linear de empacotamento ortogonal.

Problema	Limitante Superior	Número de retângulos empacotados		Número de tentativas	Tempo de CPU em segundos
		Rotações arbitrárias	Rotações ortogonais		
1	37	32	32	94921	6453,80
2	35	30	28	417	31,94
3	45	38	**40**	148870	19508,71
4	33	28	26	14734	485,29
5	40	35	33	4778	329,65
6	37	28	**30**	14573	780,77
7	25	20	19	6396	157,71
8	37	30	**32**	19422	880,40
9	28	23	22	6041	108,11
10	40	32	**34**	149935	5554,20
11	36	não disponível	31	70613	4199,44
12	32	27	25	68	1,13
13	33	28	26	67	1,21
14	36	29	29	49	0,35
15	37	30	29	42	0,34
16	37	31	30	68	1,19

Alguns esclarecimentos devem ser feitos sobre a Tabela 28. Dado que as soluções viáveis dos problemas com rotações ortogonais estão incluídas nas soluções deste problema com rotações arbitrárias, o número de retângulos ortogonais empacotados não deveria ser maior que o número de retângulos empacotados com rotações arbitrárias. Este fato é confirmado na maioria dos problemas considerados com exceção dos problemas 3, 6, 8 e 10 nos quais o procedimento para empacotamento ortogonal encontrou soluções com um maior número de retângulos.

Os resultados dos experimentos numéricos indicam que a estratégia proposta, baseada em um modelo não linear, é promissora. Para o caso onde o problema de empacotamento impõe a ortogonalidade entre os itens mas não impõe a condição dos lados dos itens retangulares estarem paralelos a algum eixo fixo, as formulações não lineares apresentadas são aplicáveis, porém podem fornecer uma resposta incompleta. A estratégia introduzida neste trabalho pode ser estendida para fornecer uma resposta completa para o problema mencionado adicionando um ângulo de rotação idêntico para todos os itens. Esta pesquisa será objeto de trabalhos futuros.

CAPÍTULO 4. APLICAÇÕES

Nos últimos anos, tem havido um crescente interesse na aplicação de técnicas de otimização nos sistemas de produção brasileiros (veja, por exemplo Taube-Netto (1996), Yoshizaki *et al.* (1996), Colin *et al.* (1999) e Caixeta-Filho (1999)). Uma possível razão para este interesse é a preocupação dos empresários com a eficácia operacional de suas empresas. Outro fator motivador para o crescente interesse é que, devido ao uso moderado destas técnicas no Brasil, existem oportunidades de melhorias utilizando métodos clássicos (Colin *et al.*,1999).

Nas próximas seções resumiremos trabalhos que consideram aplicações de técnicas de otimização na resolução de problemas reais. Em Kawamura *et al.* (2005) um modelo de programação linear de grande porte foi desenvolvido para um sistema de produção e distribuição de açúcar e álcool. Outro trabalho que também utiliza formulações para problemas práticos pode ser encontrado em Santoro & Ronconi (2005) onde se aborda a localização de pontos de armazenagem para a distribuição de cloreto de potássio.

4.1. Planejamento da Distribuição de Açúcar e Álcool

No trabalho de Kawamura *et al.* (2005) descreve-se a aplicação de programação linear multiperíodos a um sistema de produção e distribuição de açúcar e álcool. O modelo foi desenvolvido em uma cooperativa brasileira de empresas do agronegócio da cana-de-açúcar, a COPERSUCAR (Cooperativa de Produtores de Cana, Açúcar e Álcool do Estado de São Paulo LTDA).

Fundada em julho de 1959, a COPERSUCAR é composta por um grupo de 34 usinas produtoras de açúcar e álcool e possui uma administração centralizada de distribuição e comercialização. Esse grupo de usinas atende mercados distribuídos pelo Brasil e tem suas unidades produtoras concentradas no Estado de São Paulo. Com uma produção de 4,4 milhões de toneladas de açúcar e 2,7 bilhões de litros de álcool (dados da safra 99/00), a

COPERSUCAR é a maior produtora do país, sendo líder de mercado com um *market-share* na Região Centro-Sul de cerca de 26% no mercado de açúcar e 23% no de álcool.

A indústria sucroalcooleira tem como matéria-prima a cana-de-açúcar, cujo caldo é processado para a obtenção dos produtos finais: álcool e açúcar. A produção desses produtos ocorre em caráter sazonal, já que a disponibilidade de cana-de-açúcar dá-se apenas em uma parte do ano (geralmente de maio até novembro). Essa característica sazonal da produção faz com que exista a necessidade de manter um sistema eficiente de planejamento de produção, armazenagem e distribuição de modo a atender o mercado durante todo o ano.

Atualmente, esse grupo de usinas utiliza um modelo de otimização para realizar o planejamento de produção e distribuição de açúcar. Nesse modelo, as cotas dos diversos tipos de açúcar são divididas entre as usinas de modo a minimizar os custos com transporte. No caso do álcool, a produção é previamente definida por cada usina, cabendo à administração central a função de comercialização e distribuição. Isso é possível pois se trata de um mercado de "commodities", no qual as vendas podem ser estimuladas pela redução do preço, escoando-se assim a produção total. Embora os custos com armazenagem externa para ambos os produtos e os custos com transporte do álcool sejam significantes, 29% do custo total, estes custos não são considerados neste modelo. A questão da armazenagem, nesse sistema, fica em segundo plano, pois a saída de produtos é feita de modo proporcional à produção, ocorrendo em determinados períodos ociosidade de armazenagem em algumas usinas enquanto outras recorrem à armazenagem externa alugada.

A partir dessas considerações, foi possível identificar pontos falhos e propor um sistema com vistas a amenizar os problemas encontrados. O sistema proposto é baseado num modelo de otimização que visa à minimização dos custos com transporte e armazenagem externa alugada de ambas as famílias de produto (álcool e açúcar). Nesse sistema, as cotas de produção de todos os produtos são determinadas de forma integrada. Assim cada usina deve informar a quantidade total de cana-de-açúcar disponível à administração central, que, por sua vez, ficará encarregada de avaliar a demanda e dividir

as cotas de produto com base na localização da demanda, na disponibilidade de matéria-prima e nas capacidades de produção, armazenagem e distribuição das usinas.

O modelo proposto de programação linear multiperíodos e multiprodutos tem como base o modelo de transbordo clássico, adicionado de limitações de capacidade logística e de produção, e considera o atendimento total da demanda prevista. O objetivo do modelo é minimizar custos com transporte e armazenagem externa através de um planejamento de produção e distribuição ao longo de um período de 12 meses. Como resultado final, é possível saber em cada período estudado: a quantidade a ser produzida de cada produto em cada origem; o estoque disponível de cada produto em cada origem (e ponto de transbordo), a quantidade de cada família de produto (açúcar ou álcool) a ser armazenada externamente em cada origem; a quantidade enviada por cada origem (e ponto de transbordo) de cada produto em determinada embalagem para cada destino (destino final ou ponto de transbordo) e os custos envolvidos em cada uma das operações de transporte e armazenagem externa acima descritas. Detalhes da estrutura do modelo podem ser encontrados em Kawamura *et al.* (2005).

O modelo foi implementado no software "What's Best!" Versão Industrial 5.0 no microcomputador Pentium III 800 MHz com 512 Mb de memória RAM. Foram considerados: 2 famílias de produto (açúcar e álcool); 7 produtos (álcool hidratado, álcool anidro, açúcar tipo 1, açúcar tipo 2, açúcar tipo 3, açúcar tipo 4, açúcar VHP); 4 tipos de embalagem (granel, contêiner, sacaria simples, sacaria dupla); 34 origens; 70 destinos; 4 pontos de transbordo; 9 regiões produtoras; 5 períodos (Mai-Ago, Set, Out, Nov, Dez-Abr) e 2 modais de transporte (rodoviário e ferroviário). O modelo final, composto de 16.104 variáveis e 3.950 restrições, foi processado dentro da capacidade do software (32.000 variáveis e 16.000 restrições). A solução foi obtida após um tempo de processamento de 1 minuto e 26 segundos.

A entrada de dados, nesse software, assim como a saída dos resultados, é feita através de planilhas eletrônicas, fazendo com que a interface com o operador seja bastante amigável. Schuster & Allen (1998) abordam as vantagens e desvantagens desse tipo de software. Entre as principais vantagens estão a adequação da ferramenta a problemas de programação linear e a interface amigável com o usuário final. A maior desvantagem

apresentada é a dificuldade em manipular problemas extensos, considerando o excessivo trabalho para atualizar e modificar a estrutura do modelo.

A solução obtida apresenta uma redução de custos em relação ao sistema utilizado atualmente. A Tabela 29 ilustra o resultado final obtido pelo modelo em comparação ao custo estimado para o sistema atual.

Tabela 29. Resumo do Resultado Final.

	Custos (milhões US$)	***Atual***	***Proposto***	***Diferença***	***Diferença %***
Açúcar	Transporte	32,00	30,66	1,34	4,19
	Armazenagem Externa	1,90	0,40	1,50	78,95
Álcool	Transporte	11,40	10,92	0,48	4,21
	Armazenagem Externa	0,00	0,00	0,00	0,00
	Total Geral	**45,30**	**41,98**	**3,32**	**7,33**

Analisando o quadro acima, é possível inferir que há uma redução de custos significativa tanto do ponto de vista absoluto (3,3 milhões US$) quanto do ponto de vista relativo (7,3%). Todos os custos são reduzidos, com destaque para a armazenagem externa de açúcar. Além disso, a implantação do modelo poderia ser imediata, dada a não necessidade de investimentos adicionais já que o modelo elaborado considera as restrições da infra-estrutura existente.

Uma análise detalhada permite afirmar que a redução de custos de transporte deve-se principalmente à definição do "mix" de açúcar e álcool em cada usina, proporcionada pela flexibilidade de alocação da produção. Continuando a análise, a redução de custos de armazenagem externa deve-se especialmente à melhor utilização dos armazéns através da otimização das saídas dos produtos. O critério anterior de saídas (proporcional em relação ao estoque) é substituído pelo critério de necessidade de saídas, ou seja, a decisão é dada pela quantidade ao longo do tempo necessária para evitar armazenagem externa. Por outro lado, ao modificar o critério de saídas, ocorre um desbalanceamento nas operações de carregamento e expedição de produtos, fazendo com que existam períodos de utilização máxima da capacidade de expedição e outros de saídas insignificantes. O mesmo ocorre com a produção dos diversos tipos de produto, que também sofre alterações significativas ao longo do tempo, fazendo com que ocorram períodos de produção máxima de certo tipo

de produto intercalados com períodos de picos de produção de outros produtos. Essa alteração deve-se à necessidade de atendimento completo da demanda conjugada à minimização dos custos de armazenagem.

Com a intenção de prover informações úteis para a cooperativa sem modificar drasticamente a configuração atual, uma análise de sensibilidade foi realizada para investigar uma possível expansão da capacidade. As restrições que estabelecem as capacidades máximas disponíveis foram modificadas e o modelo foi resolvido com diversos novos valores de capacidade. Para cada incremento permitido é possível identificar que usina deve ser expandida e o ganho esperado. A Figura 19 ilustra o comportamento do sistema para cada cenário relaxado. Pode ser observado que o maior ganho (quando comparado ao cenário anterior) é obtido com um aumento de 10% na capacidade. Este aumento relativo diminui com o aumento da capacidade e tende a ser nulo quando o incremento na capacidade é maior que 120%. Através desta figura, os gerentes podem analisar de forma clara os possíveis benefícios que poderiam ser obtidos na ocorrência de investimentos na capacidade geral de produção do sistema.

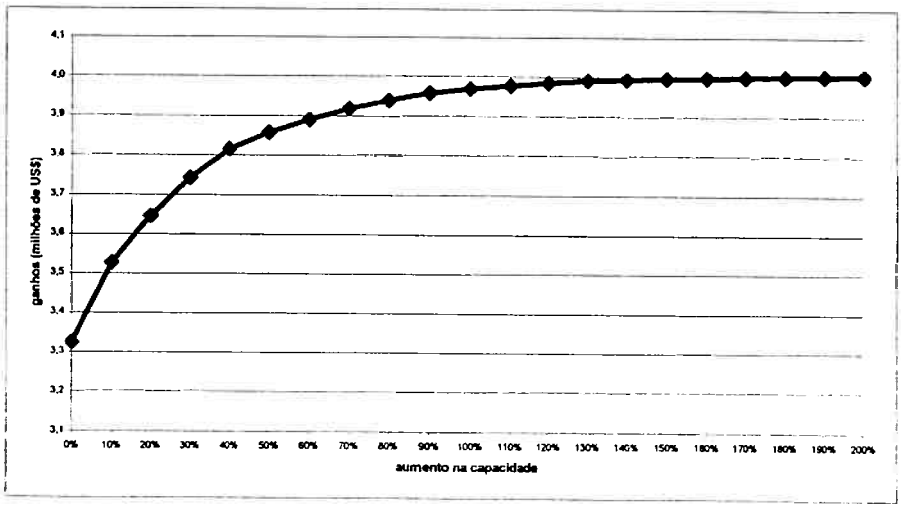

Figura 19. Ganhos obtidos com o aumento da capacidade das usinas.

Este trabalho apresentou um modelo de programação linear multiperíodos e multiprodutos elaborado com o objetivo de auxiliar o processo decisório de planejamento de produção em uma cooperativa de empresas do agronegócio da cana-de-açúcar, a COPERSUCAR. Com base nos resultados computacionais apresentados, é possível afirmar

que o modelo desenvolvido é capaz de fornecer soluções viáveis operacionalmente que atendem às necessidades da empresa.

4.2.Localização Dinâmica de Pontos de Armazenagem de Cloreto de Potássio

No trabalho de Santoro & Ronconi (2005) descreve-se a aplicação de programação linear inteira multiperíodos a um sistema de distribuição de cloreto de potássio. O modelo foi desenvolvido para a CVRD - Companhia Vale do Rio Doce. Criada em julho de 1942, a CVRD é uma das maiores empresas de produção e exportação de minério de ferro *in natura* e pelotas de minério de ferro em âmbito mundial. Em janeiro de 1992, a CVRD, por meio de contrato de arrendamento com a Petrobrás, assumiu o gerenciamento da mina de Potássio Taquari-Vassouras, no município de Rosário do Catete/SE. A unidade é a única produtora de cloreto de potássio (KCl) em atividade no Brasil. O cloreto de potássio é um importante componente para a produção de fertilizantes, sendo produzido através de lavra subterrânea de silvinita e posteriormente beneficiado por meio do processo de flotação. As reservas são estimadas em 13 milhões de toneladas (informações disponíveis na página *http://www.cvrd.com.br*). Na época do estudo toda a produção - 506 mil toneladas de cloreto de potássio granulado e 114 mil toneladas de *standard* - estava sendo comercializada no mercado interno. Esta produção responde por cerca de 15% da demanda brasileira do produto.

A indústria de fertilizante tem como matéria-prima fontes de *NPK* (Nitrogênio-Fósforo-Potássio) que são processadas para a obtenção dos produtos finais: fertilizantes mistos. A porcentagem de cada componente, Nitrogênio, Fósforo, Potássio, na composição do produto final é variável pois dependendo da carência do solo a ser fertilizado, esta composição pode ser alterada. A extração do cloreto de potássio tem como característica um alto custo de variação do nível de produção dado o alto investimento em equipamentos e arrendamento. A demanda, por outro lado, é sazonal, dado que é um insumo agrícola. Essa característica faz com que exista a necessidade de manter um sistema eficiente de planejamento de armazenagem e distribuição de modo a atender o mercado durante todo o ano minimizando os custos de transporte, armazenagem e atrasos na entrega.

Atualmente, planeja-se um volume razoavelmente estável de extração de cloreto de potássio capaz de atender a demanda, distribuída em diferentes estados. O atendimento é feito diretamente da mina. Este planejamento é realizado sem auxílio de técnicas de otimização. A administração central tem a função de comercialização e distribuição, funções essas muito importantes, uma vez que não existem armazéns reguladores de porte nos pontos de demanda, e deve garantir o máximo escoamento da extração da mina. Como o frete tem custos sazonais, em certas épocas do ano o custo de transporte aumenta significativamente e, pela dificuldade de estocagem nos pontos de demanda, o mesmo não pode ser evitado. Esses fatos sugeriram o desenvolvimento de um modelo que pudesse amenizar os problemas encontrados. O sistema proposto por Santoro & Ronconi (2005) é baseado num modelo de otimização multiperíodos que visa a minimização dos custos com transporte, armazenagem e atrasos na entrega.

Aplicações de programação matemática em problemas de distribuição da produção e localização de centros de armazenagem são encontradas em abundância na literatura. A seguir citamos algumas destas aplicações. Gutierrez (1996) desenvolveu um modelo de programação linear (LP) multiperíodos e multiprodutos para otimizar a produção, a armazenagem de matérias-primas e produtos acabados, e a distribuição de uma empresa de insumos agrícolas com elevada sazonalidade no consumo. Schuster & Allen (1998) tratam do problema de planejamento agregado de um fabricante de derivados de frutas, minimizando custos de transporte, fabricação e estocagem também através de um modelo de LP. Na indústria sucroalcooleira, Yoshizaki *et al.* (1996) aplicaram um modelo de transbordo na distribuição de álcool combustível e Colin *et al.* (1999) tratam da distribuição de açúcar de uma usina em um modelo linear multiperíodos. Rabinowitz & Mehrez (2001) apresentam um modelo não linear multiperíodos para a reestruturação do sistema logístico de uma empresa do ramo de extração de potássio. Os autores analisam e sugerem modificações na capacidade de armazenamento de cada subproduto nos pontos de distribuição. Jolayemi & Olorunniwo (2004) propõem um modelo de programação linear inteira mista (MILP) multiperíodos para o planejamento da produção e transporte de mercadorias em um ambiente com múltiplas origens e armazéns com capacidades extensíveis.

Preocupados com a localização de estruturas, Hindi *et al.* (1998) abordam o problema da distribuição de diferentes "commodities" através de um modelo monoperíodo MILP. Neste problema os clientes devem ser abastecidos através de diferentes origens, passando por centros de distribuição que podem ser ativados ou não. Jayaraman (1998) trata simultaneamente, através de um modelo MILP monoperíodo, o problema de gerenciamento de estoques, localização de instalações e determinação da política de transportes.

Os modelos de planejamento e distribuição da produção são tipicamente multiperíodos e tratam com sistemas estabelecidos onde existem demanda sazonal e custos variáveis nos períodos. Por outro lado os modelos de localização de estruturas geralmente são monoperíodo, minimizam custos de transporte e pressupõem o sistema em regime estável, veja por exemplo os recentes trabalhos de Resende & Werneck (2005) e Michel & Van Hentenryck (2004). Conforme a revisão da literatura apresentada por Owen & Daskin (1998), o estudo do aspecto dinâmico da localização de estruturas teve início há muitos anos (veja por exemplo Tapiero (1971) e Van Roy & Erlenkotter (1982)), porém, ainda poucos trabalhos abordam este tema.

Para o problema tratado foi desenvolvido um modelo multiperíodos que combina características dos modelos de localização com os de planejamento e distribuição da produção. É um modelo de maximização da margem de lucro que decide se é interessante ou não existir armazenagem em certos pontos do sistema e, além disso, considera a possibilidade de atrasos no atendimento da demanda. Diferentemente das abordagens normalmente utilizadas, o modelo não incorpora o custo fixo de instalação de armazéns, permitindo a decisão em separado de instalação ou não de um ou mais armazéns, considerando a margem obtida comparativamente aos custos de instalação dos mesmos. Esta abordagem foi escolhida para flexibilizar a ação da empresa que pode decidir construir ou não armazéns com instalações mais ou menos sofisticadas nos pontos selecionados ou analisar formas alternativas de armazenagem de menor custo, como por exemplo, a locação temporária de armazéns de terceiros ou de clientes e a antecipação de entregas com eventual redução de preços.

A idéia básica do modelo é verificar se é interessante aproveitar a forte variação do preço do frete para compensar os custos de implantação e operação de um ou mais pontos de armazenagens próximos aos pontos de maior demanda. O modelo integrado proposto utiliza conceitos de programação linear inteira mista e foi desenvolvido para auxiliar o processo decisório de planejamento da distribuição do cloreto de potássio dentro do sistema gerenciado pela CVRD. O objetivo é a maximização da margem bruta de lucro do sistema - diferença entre a receita das vendas e os custos de produção, de estocagem e de operação nos armazéns, de transporte na rede e de atrasos. As restrições tratam das limitações de capacidade de escoamento das origens, de fluxo de materiais, de armazenagem e do número máximo de armazéns. As decisões principais são a alocação ótima de pontos de armazenagem e a movimentação do cloreto de potássio no sistema.

Na modelagem apresentada consideram-se somente o produto granulado e as rotas usuais de transporte. A capacidade dos armazéns é limitada, o custo de operação é proporcional à quantidade movimentada e não se considera o custo fixo de instalação dos mesmos. Para cada período serão obtidos os seguintes valores: a quantidade de produtos produzidos em cada origem; a quantidade estocada em cada armazém; o volume transportado de cada origem para cada armazém e de cada origem ou armazém para cada destino; a quantidade de unidades atrasadas e em qual destino; e todos os custos envolvidos nas operações de transporte e armazenagem. Além disso, como resultado final, é possível determinar a margem de lucro esperada em cada período estudado e as localizações ideais dos armazéns e suas capacidades mínimas. Maiores detalhes sobre o modelo proposto podem ser encontrados em Santoro & Ronconi (2005).

O modelo foi implementado no software "What's Best!" 5.0 em um microcomputador Pentium III 750 MHz com 128 Mb de memória RAM. A entrada de dados, nesse software, assim como a saída dos resultados, é feita através de planilhas eletrônicas (no caso o Microsoft Excel), fazendo com que a interface com o operador seja bastante amigável.

Foram considerados: 12 pontos de consumo; uma única origem, a mina de potássio Taquari-Vassouras; 4 cidades como possíveis pontos de instalação dos armazéns; armazéns sem estoques inicias; destinos sem pendências de entrega no início do horizonte de

planejamento, e 4 trimestres. A produção da mina foi considerada conhecida durante os períodos analisados e com valor constante igual a 212.500 ton. A demanda de cada ponto de consumo está apresentada na Tabela 30. Note que apesar da demanda total no horizonte considerado, 850.000 ton, ser igual a produção total, nos primeiros trimestres ela é inferior, tornando obrigatória a existência de armazenagem de produto.

Tabela 30. Demandas (ton.) agregadas por período.

Período	Destino												Total
	1	2	3	4	5	6	7	8	9	10	11	12	
1	633	5553	16506	2769	28831	8626	477	56839	7387	15896	23772	141	167430
2	575	6059	11317	2769	34686	7667	488	83087	6364	14061	19739	162	186974
3	846	6724	16085	2769	50835	12902	905	114185	10573	20368	27212	207	263611
4	487	6183	12157	2769	39694	7608	1113	102480	10397	18471	30442	184	231985

O modelo final, composto de 240 variáveis reais, 4 variáveis binárias e 94 restrições, foi processado dentro da capacidade do software. Esclarece-se que algumas variáveis foram eliminadas pela ausência de rotas viáveis de transporte entre alguns pontos. A solução foi obtida após poucos segundos de processamento.

Apresenta-se a seguir a solução ótima obtida pelo modelo parametrizado para NA=4, ou seja permitiu-se até 4 armazéns. O modelo decidiu por 2 armazéns nos locais candidatos a=1 e a=3, o que significa que a solução obtida é a mesma para NA igual a 2, 3 ou 4. A Tabela 31 fornece as quantidades a serem transportadas da origem e dos armazéns para cada ponto de consumo, além das quantidades que serão enviadas da origem para cada armazém.

Tabela 31. Quantidades (ton.) transportadas em cada período.

Período (t)	Ponto de partida		Armazém				Destino											
			1	2	3	4	1	2	3	4	5	6	7	8	9	10	11	12
1	origem	1	45070				633	5553	16506	2769	28831	8626	477	56839	7387	15896	23772	141
	armazém	1																
		2																
		3																
		4																
2	origem	1	6041		19485		575	6059	11317	2769	34686	7667	488	83087	6364	14061	19739	162
	armazém	1																
		2																
		3																
		4																
3	origem	1					846	6724	8762		50835			114185	10573	20368		207
	armazém	1							7323	2769		12902	905				27212	
		2																
		3																
		4																
4	origem	1					487	6183	12157	2769	39694	7608	1113	82995	10397	18471	30442	184
	armazém	1																
		2																
		3												19485				
		4																

Como a produção acumulada é superior à demanda nos 3 primeiros trimestres, é imprescindível estocar. A Tabela 32 fornece as decisões tomadas pelo modelo em relação às quantidades acumuladas de produtos nos armazéns no final de cada período e indica que os armazéns a=1 e a=3 deverão ter capacidades mínimas de aproximadamente 51.200 ton e 19.500 ton, respectivamente.

Tabela 32. Quantidades estocadas (ton.) nos finais de cada período.

Período	Armazéns (a)			
(t)	1	2	3	4
0				
1	45070			
2	51111		19485	
3			19485	
4				

Os resultados obtidos podem ser analisados em conjunto com os custos médios de frete entre a origem (mina Taquari-Vassouras) e os armazéns (Figura 20). No primeiro período, conforme era esperado, a inevitável criação de estoque ocorreu no armazém localizado na origem (Rosário do Catete/SE), evitando assim os custos com o transporte.

Porém, observa-se que no segundo período, a redução de custos obtida pelos baixos valores dos fretes compensou os custos com a armazenagem de produtos em pontos distantes da origem (a=3, Estado de Goiás).

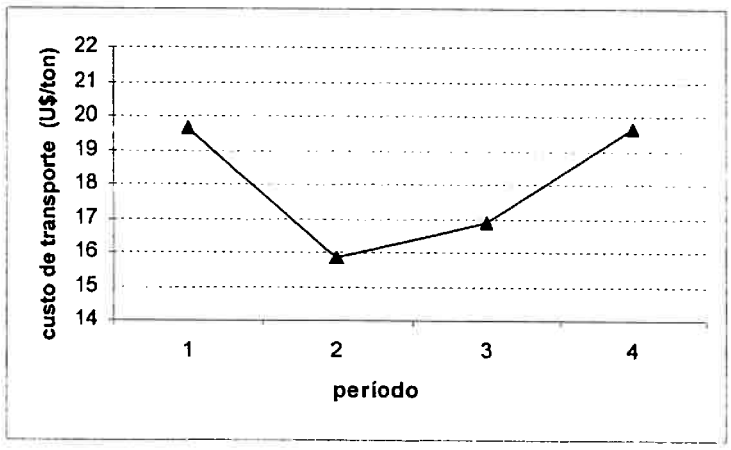

Figura 20. Custos médios de transporte entre a origem e os armazéns.

A solução ótima obtida pelo modelo, com dois pontos de armazenagem, não apresenta atrasos e a margem de lucro obtida é de US$ 50.295.971. Esta margem não considera o custo fixo de instalação de armazéns, o modelo delega para a empresa a função de analisar a margem de lucro obtida e compará-la com os custos necessários para a realização da armazenagem.

Buscando fornecer informações úteis para a empresa, o número máximo de armazéns permitidos (*NA*) foi alterado para avaliar o efeito na margem de lucro. A Tabela 33 permite comparar as soluções ótimas obtidas com a variável *NA* tomando os valores entre 1 e 4.

Tabela 33. Comparação de soluções com número diferente de armazéns permitidos (NA).

		Capacidade mínima dos armazéns (ton)				Margem de lucro (US)
		1	2	3	4	
NA	1	70596				50.263.522
	2, 3 e 4	45070		19485		50.295.971

O pequeno ganho na margem de lucro com dois armazéns (U$ 32.449) aliado à necessidade de menor capacidade de armazenagem no armazém da localização 1 não compensa o custo fixo relativo à construção de um armazém na localização 3. Dessa forma concluímos que a melhor decisão é manter somente um armazém na origem. Porém, como

se mostrou vantajosa a armazenagem de produtos em outros pontos diferentes da origem, o resultado sugere a análise de formas alternativas de armazenagem de menor custo, como por exemplo, a locação temporária de armazéns de terceiros ou de clientes, ou mesmo a antecipação de entregas com eventual redução de preços. O estudo fornece todas as informações necessárias para uma negociação de preços ou análise de custos.

Com base nos resultados computacionais apresentados, é possível afirmar que o modelo desenvolvido foi capaz de fornecer soluções viáveis operacionalmente que atendem às necessidades da empresa em resolver o problema descrito, podendo ser utilizado tanto para a decisão sobre investimentos em armazéns como para as de planejamento do fluxo de cloreto de potássio, considerados os armazéns definidos, e mesmo para as decisões de preço do produto nos mercados. Vale ressaltar que o modelo apresentado foi inicialmente aplicado considerando somente a produção nacional de cloreto de potássio granulado, no entanto, ele possui flexibilidade para considerar outros produtos e outros pontos de origem, como por exemplo portos, no caso de abastecimento por meio de importação.

CAPÍTULO 5. CONCLUSÕES

Neste trabalho descrevemos uma seleção de artigos nos quais várias técnicas de otimização são aplicadas a problemas de engenharia de produção, especialmente na área de Gestão de Operações e Logística. Estes trabalhos abrangem diversos tipos de problemas abordados pela candidata durante o seu trabalho de pesquisa. Dentre estes destaca-se a resolução de problemas de programação de tarefas em diferentes ambientes. Além disso, foram considerados problemas de empacotamento e aplicações práticas.

As técnicas de otimização mais utilizadas foram: modelagem de problemas com programação linear e programação inteira, desenvolvimento de novos limitantes inferiores combinados com o algoritmo *branch-and-bound*, elaboração de heurísticas construtivas e de busca local e aplicação de meta-heurísticas puras e híbridas. Além disso, estratégias mistas envolvendo métodos de programação não linear também foram elaboradas.

Cada técnica foi selecionada de acordo com as propriedades e complexidade do problema tratado, além do objetivo do estudo. Para as aplicações práticas, desenvolvemos modelos possíveis de serem solucionados através de pacotes comerciais disponíveis para as empresas interessadas. Para os problemas de alta complexidade, foram desenvolvidos tanto métodos heurísticos como exatos. O intuito da elaboração de novos métodos heurísticos é a rápida obtenção de soluções de boa qualidade. Estes métodos, além de relevantes na prática, procuraram contribuir com a literatura existente fornecendo métodos competitivos. A motivação para o desenvolvimento de métodos exatos, que possuem restrições quanto à dimensão das instâncias resolvidas, é a possibilidade de fornecer parâmetros de comparação para avaliar a qualidade de métodos heurísticos, além de encontrar soluções ótimas para problemas de pequeno porte.

Vale ressaltar que o desenvolvimento dos métodos propostos sempre partiu de um estudo aprofundado das características de cada problema abordado. Acreditamos que, embora seja verdade que existem métodos heurísticos e exatos que podem ser aplicados a uma vasta gama de problemas, é a exploração de propriedades específicas de cada problema que leva a uma resolução eficiente do mesmo. Desta forma, independentemente

do método utilizado, a tentativa de resolver um problema deve começar pela busca de características específicas de cada problema estudado.

Os estudos descritos apresentaram bons resultados e os correspondentes artigos estão publicados ou submetidos em revistas internacionais reconhecidas. Este texto também apresentou trabalhos inéditos recentemente desenvolvidos. Além disso, novos estudos, que dão continuidade a alguns tópicos anteriormente mencionados, estão em andamento. Dentre estes podemos destacar o empacotamento de cilindros de diferentes diâmetros e o desenvolvimento de limitantes inferiores considerando a minimização da soma ponderada dos adiantamentos e atrasos das tarefas.

Birgin *et al.* (2005a e 2005b) lidam com o problema de empacotar itens dentro de objetos utilizando heurísticas baseadas em subproblemas de programação não linear. Basicamente, a resolução dos subproblemas corresponde a responder a pergunta de se um determinado subconjunto dos itens pode ou não ser empacotado no objeto. A contribuição dos métodos numéricos de programação não linear consiste em tentar resolver os subproblemas. Só que muitas vezes os subproblemas são problemas complexos de programação não linear com muitos minimizadores locais não globais. Portanto, a resposta à pergunta de se um determinado subconjunto de itens pode ou não ser empacotado pode ser "não sei", no lugar dos desejados "sim" ou "não". Lidar com esse tipo de respostas e decidir quais são as perguntas que devem ser feitas (subproblemas que deve se tentar resolver) são as questões importantes que a parte heurística do método deve resolver.

Quando os itens que se deseja empacotar são idênticos, o problema de decidir que perguntas devem ser feitas é simples. Começamos perguntando se pode ser empacotado um item. Enquanto a resposta for sim, tentamos colocar mais um. Esta foi a abordagem utilizada em Birgin *et al.* (2005a), onde é apresentado um método para o empacotamento de cilindros idênticos num contêiner retangular. O modelo introduzido neste trabalho também pode ser estendido para empacotar círculos diferentes dentro de retângulos e círculos (veja Seção 3.1). Para estes casos, o esquema para resolver os problemas de decisão é aplicável. Por outro lado, não é claro que problemas de decisão devem ser resolvidos de forma a maximizar a área utilizada. Como um trabalho futuro, pretendemos desenvolver heurísticas adequadas para a seleção dos cilindros diferentes que devem ser empacotados. Os

resultados obtidos serão comparados com resultados recentes obtidos por Hifi & Hallah (2004).

Outro trabalho em andamento aborda o problema de programação de tarefas em uma única máquina. As tarefas têm uma data de entrega comum restrita e conhecida. A medida de desempenho considerada é a minimização da soma ponderada dos adiantamentos e atrasos das tarefas. Devido à complexidade do problema, *NP-hard* (Hall *et al.*, 1991), diversos autores abordaram este problema com técnicas heurísticas (Lee & Kim, 1995; James, 1997; Biskup & Feldmann, 2001; Feldman & Biskup, 2003; Hino *et al.*, 2005). Nos últimos anos, o interesse e os esforços em algoritmos exatos têm crescido devido ao aumento significativo da capacidade de processamento dos equipamentos atuais. No entanto, para o caso em questão, não há na literatura existente, modelos desenvolvidos que obtenham soluções comprovadamente ótimas com tempos de processamento razoáveis.

Assim sendo, o trabalho proposto busca elaborar um algoritmo exato para o problema considerado baseado no método *branch-and-bound*. O algoritmo *branch-and-bound* utilizará duas abordagens diferentes dependentes do instante de início de processamento das tarefas: no instante zero ou em um instante diferente de zero. Esta divisão baseia-se na Propriedade 3 descrita na Seção 2.3.1 que afirma que existe uma solução ótima onde ou uma tarefa é completada na data de entrega ou a primeira tarefa inicia o seu processamento no instante zero. Essa divisão busca explorar as características do problema e as propriedades conhecidas para minimizar o esforço computacional e tornar o algoritmo mais eficiente. Para ambas estratégias limitantes inferiores serão propostos.

BIBLIOGRAFIA

Abadi, I.N.K., Hall, N.G. & Sriskandarajah, C., 2000, Minimizing Cycle Time in a Blocking Flowshop, *Operations Research*, 48, pp. 177-180.

Adenso-Díaz, B., 1996, An SA/TS Mixture Algorithm for the Scheduling Tardiness Problem, *European Journal of Operational Research*, 88, pp. 516-524.

Adenso-Díaz, F., Glover, F.W., Ghaziri, H.M., González, J.L., Laguna, M., Moscato, P. & Tseng, F.T., 1996, *Optimización Heurística y Redes Neuronales*, Editorial Paraninfo, Madrid.

Aiex, R.M., Binato, S. & Resende, M.G.C., 2003, Parallel Grasp with Path-Relinking for Job Shop Scheduling, *Parallel Computing*, 29, pp. 393- 430.

Armentano, V.A. & Ronconi, D.P., 1999, Tabu Search for Total Tardiness Minimization in Flowshop Scheduling Problems, *Computers & Operations Research*, 26, pp. 219-235.

Armentano, V.A. & Ronconi, D.P., 2000, Minimização do Tempo Total de Atraso no Problema de *Flowshop* com Buffer Zero Utilizando Busca Tabu, *Gestão & Produção*, 7, pp. 353-363.

Akturk, M.S. & Ozdemir, D., 2001, A New Dominance Rule to Minimize Total Weighted Tardiness with Unequal Release Dates, *European Journal of Operational Research*, 135, pp. 394-412.

Bagchi, U.; Chang, Y. & Sullivan, R. S., 1986, Minimizing Mean Absolute Deviation of Completion Times about a Common Due Date, *Naval Research Logistics Quarterly*, 33, pp. 227-240.

Baker, K.R., 1974, *Introduction to sequencing and scheduling*. John Wiley, New York.

Baker, K. R. & Scudder, G. D., 1990, Sequencing with Earliness and Tardiness Penalties: A Review, *Operations Research*, 38, pp. 22-36.

Beasley, J.E., 2004, A Population Heuristic For Constrained Two-Dimensional Non-Guillotine Cutting, *European Journal of Operational Research*, 156, pp. 601-627.

Berkley, B.J., 1993, Effect of buffer capacity and sequencing rules on single-card kanban system performance, *International Journal of Production Research*, 31, pp. 2875-2893.

Birgin, E.G. & Martínez, J.M., 2002, Large-Scale Active-Set Box-Constrained Optimization Method with Spectral Projected Gradients, *Computational Optimization and Applications*, 23, pp. 101-125.

Birgin, E.G., Martínez, J.M. & Raydan, M., 2001, SPG: Software for Convex-Constrained Optimization, *ACM Transactions on Mathematical Software*, 27, pp. 340-349.

Birgin, E.G., Martínez, J.M. & Ronconi, D.P., 2003, Minimization Subproblems and Heuristics for an Applied Clustering Problem, *European Journal of Operational Research*, 146, pp. 19-34.

Birgin, E.G., Martínez, J.M. & Ronconi, D.P., 2005a, Optimizing the Packing of Cylinders into a Rectangular Container: A Nonlinear Approach, *European Journal of Operational Research*, 160, pp. 19-33.

Birgin, E.G., Martínez, J.M., Nishihara, F.H. & Ronconi, D.P., 2005b, Orthogonal Packing of Rectangular Items within Arbitrary Convex Regions by Nonlinear Optimization, aparecerá em *Computers & Operations Research*.

Birgin, E.G., Martínez J.M., Mascarenhas, W.F. & Ronconi D.P., 2005c, Method of Sentinels for Packing Objects within Arbitrary Regions, aparecerá em *Journal of the Operational Research Society*.

Birgin, E.G., Morabito, R. & Nishihara F.H., 2005d, A Note on an L-approach for Solving the Manufacturer's Pallet Loading Problem, *Journal of the Operational Research Society*, 56, pp. 1448-1451.

Biskup, D. & Feldmann, M., 2001, Benchmarks for Scheduling on Single/Machine against Restrictive and Unrestrictive Common Due Date, *Computers & Operations Research*, 28, pp. 787-801.

Buzzo, W.R. & Moccellin, J.V., 2000, Programação Da Produção Em Sistemas Flow Shop Utilizando Um Método Heurístico Híbrido Algoritmo Genético/Simulated Anneling, *Gestão & Produção*, 7, pp. 364-377.

Caixeta-Filho, J.V., 1999, Losses in the Transportation of Fruits and Vegetables: A Brazilian Case, *International Journal of Logistics*, 2, pp. 325-342.

Canuto, S.A., Resende, M.G.C. & Ribeiro, C.C., 2001, Local Search with Perturbations for the Prize-Collecting Steiner Tree Problem In Graphs, *Networks*, 38, pp. 50-58.

Caraffa, V., Ianes, S, Bagchi, T.P. & Sriskandarajah, C., 2001, Minimizing Makespan in a Flowshop Using Genetic Algorithms, *International Journal of Production Economics*, 70, pp. 101-115.

Cheng, T.C.E. & Kahlbacher H.G., 1991, A Proof for the Longest/Job/First Policy in One/Machine Scheduling, *Naval Research Logistics*, 38, pp. 715-720.

Colin, E.C., Cipparrone, F.A.M. & Shimizu, T., 1999, Otimização do Custo de Transporte na Distribuição – Armazenagem de Açúcar, *Produção*, 9, pp. 23-30.

Conway, R.W., Maxwell, W.L. & Miller, L.W., 1967, *Theory of Scheduling*, Dover Publications, New York.

Correia, M.H., Oliveira, J.F., Ferreira J.S., 2000, Cylinder Packing by Simulated Annealing, *Pesquisa Operacional*, 20, pp. 269-284.

Correia. M.H., Oliveira, J.F., FerreiraJ.S., 2001, A New Upper Bound for the Cylinder Packing Problem, *International Transactions in Operational Research*, 8, pp. 571-583.

Dantzig, G.B. & Thapa, M.N., 1997, *Linear Programming*, Springer, New York.

Dowsland, K.A., 1987, An Exact Algorithm for the Pallet Loading Problem, *European Journal of Operational Research*, 84, pp. 78-84.

Dowsland, K.A., 1996, Genetic algorithms. a tool for OR?, *Journal of the Operational Research Society*, 47, pp. 550-561.

Du, J., & Leung, J.Y.-T., 1990, Minimizing Total Tardiness on one Machine is NP-hard, *Mathematics of Operations Research*, 15, pp. 483-495.

Dudek, R.A., Panwalkar, S.S., & Smith, M.L., 1992, The Lessons of Flowshop Scheduling Research, *Operations Research*, 40, pp. 7-13.

Dueck, G. & Scheuer T., 1990, Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Anneling, *Journal of Computational Physics*, 90, pp.161-175.

Dutta, S.K., & Cunningham, A.A., 1975, Sequencing Two-Machine Flow-Shops with Finite Intermediate Storage, *Management Science*, 21, pp. 989-996.

Dyckhoff, H., Scheithauer, G. & Terno, J., 1997, Cutting and packing: An Annotated Bibliography, em *Annotated Bibliographies in Combinatorial Optimization*, Eds. Dell'Amico, F., Maffioli, F. & Martello, S., 44, pp. 145-159.

Feldmann, M. & Biskup D., 2003, Single-Machine Scheduling for Minimizing Earliness and Tardiness Penalties By Meta-Heuristic Approaches, *Computers & Industrial Engineering*, 44, pp. 307-323.

Feo, T. & Resende, M.G.C., 1989, A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem, *Operations Research Letters*, 8, pp. 67-71.

Feo, T. & Resende, M.G.C., 1995, Greedy Randomized Adaptative Search Procedures, *Journal of Global Optimization*, 6, pp. 109-133.

Festa, P., Pardalos, P.M., Pitsoulis, L.S. & Resende, M.G.C., 2005, Grasp with Path-Relinking for the Weighted Maximum Satisfiability Problem, em *Proceedings of IV Workshop on Efficient and Experimental Algorithms (WEA2005)*, Lecture Notes in Computer Science, 3503, pp. 367-379.

Fraser, H.J. & George J.A., 1994, Integrated Container Loading Software for Pulp and Paper Industry, *European Journal of Operational Research*, 77, pp. 466-474.

Furlan, C.F. & Ronconi, D.P., 2005, Programação de Ordens de Produção com Diferentes Instantes de Liberação para uma Data Única de Entrega, *GEPROS – Gestão da Produção e Sistemas*, 1, pp. 19-28.

George, J.A., George, J.M. & Lamar B.W., 1995, Packing Different-sized Circles into a Rectangular Container, *European Journal of Operational Research*, 84, pp. 693-712.

Gilmore, P.C. & Gomory, R.E., 1964, Sequencing a One State Variable Machine: A Solvable Case of the Traveling Salesman Problem, *Operational Research*, 12, pp. 655-665.

Glover, F., 1996, Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges, em *Interfaces in Computer Science and Operations Research*, Eds. Barr, R., Helgason, R. & Kennington, J., Kluwer Academic Publishers, pp. 1-75.

Glover, F. & Laguna M., 1997, *Tabu Search*, Kluwer Academic Publishers, Boston.

Gonçalves, J.F., Mendes J.J.M. & Resende M.G.C., 2005, A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem, *European Journal of Operational Research*, 167, pp. 77-95.

Gordon, V.; Proth, J. & Chu, C., 2002, A Survey of the State-Of-The-Art of Common Due Date Assignment and Scheduling Research, *European Journal of Operation Research*, 139, pp. 1-25.

Grabowski, J., & Pempera J., 2000, Sequencing of Jobs in Some Production System, *European Journal of Operational Research*, 125, pp. 535-550.

Graham, R.L., Lubachevsky, B.D., Nurmela, K.J. & Ostergard, P.R.J., 1998, Dense Packing of Congruent Circles in a Circle, *Discrete Mathematics*, 181, 139-154.

Gutierrez, C.J., 1996, Development and Application of a Linear Programming Model to Optimize Production and Distribution of a Manufacturing Company, Dissertação de Mestrado, Massachusetts Institute of Technology - MIT.

Hall, N.G., Kubiak, W. & Sethi S.P., 1991, Earliness/Tardiness Scheduling Problems, II: Deviation of Completion Times About a Restrictive Common Due Date, *Operations Research*, 39, pp. 847-856.

Hall, N.G., & Sriskandarajah, C., 1996, A Survey of Machine Scheduling Problems with Blocking and No-wait in Process, *Operations Research*, 44, pp. 510-525.

Hifi, M. & M'Hallah, R., 2004, Approximate Algorithms for Constrained Circular Cutting Problems, *Computers & Operations Research*, 31, pp. 675-694.

Hindi, K.S., Basta, T. & Pienkosz, K., 1998, Efficient Solution of a Multi-commodity, Two-stage Distribution Problem with Constraints on Assignment of Customers to Distribution Centers, *International Transactions in Operational Research*, 5, pp. 519-527.

Hino, C.M., Ronconi, D.P. & Mendes, A.B., 2005, Minimizing Earliness and Tardiness Penalties in a Single-Machine Problem with a Common Due Date, *European Journal of Operational Research*, 160, pp. 190-201.

Holland, J.H., 1975, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor.

Isermann, H., 1991, Heuristiken zur Lösung des zweidimensionalen Packproblem für Rundgefäße, *OR Spectrum*, 54, pp. 213-223.

James, R.J.W., 1997, Using Tabu Search to Solve the Common Due Date Early/Tardy Machine Scheduling Problem, *Computers & Operations Research*, 24, pp. 199-208.

Jayaraman, V., 1998, Transportation, Facility Location and Inventory Issues in Distribution Network Design, *International Journal of Operations & Production Management*, 18, pp. 471-493.

Johnson, S.M., 1954, Optimal Two- and Three-Stage Production Schedules with Setup Times Included, *Naval Research Logistics Quartely*, 1, pp. 61-67.

Jolayemi, J.K. & Olorunniwo, F.O., 2004, A Deterministic Model for Planning Production Quantities in a Multi-plant, Multi-warehouse Environment with Extensible Capacities, *International Journal of Production Economics*, 87, pp. 99-113.

Kanet, J.J., 1981, Minimizing the Average Deviation of Job Completion Times about a Common Due Date, *Naval Research Logistics Quarterly*, 28, pp. 643-651.

Kawamura, M.S., Ronconi D.P. & Yoshizaki H., Optimizing Transportation and Storage of Final Products in the Sugar and Ethanol Industry, submetido para *International Transactions in Operational Research.*

Kim, Y-D., 1993, Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness, *Journal of the Operational Research Society*, 44, pp. 19-28.

Kim, Y-D., 1995, Minimizing Total Tardiness in Permutation Flowshops, *European Journal of Operational Research*, 85, pp. 541-555.

Kiyuzato, L., Ronconi D.P., Salamoni R., Tsai C.K. & Yoshizaki H., 2002, Minimizing Total Tardiness: a case study in an autoparts industry, *International Transactions in Operational Research*, 9, pp. 371-379.

Kozan, E. & Preston, P., 1999, Genetic Algorithms to Schedule Containers Transfers at Multimodal Terminals, *International Transactions in Operational Research*, 6, pp. 311-329.

Laguna, M. & Martí, R., 1999, GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization, *INFORMS Journal on Computing*, 11, pp. 44-52.

Laguna, M. & Velarde J.L.G., 1991, A Search Heuristic for Just-In-Time Scheduling in Parallel Machines, *Journal of Intelligent Manufacturing*, 2, pp. 253-260.

Lee, C.Y. & Kim S.J., 1995, Parallel Genetic Algorithms for the Earliness/Tardiness Job Scheduling Problem with General Penalty Weights, *Computers & Industrial Engineering*, 28, pp. 231-243.

Leisten, R., 1990, Flowshop Sequencing Problems with Limited Buffer Storage, *International Journal of Prodution Research*, 28, pp. 2085-2100.

Levner, E.M., 1969, Optimal Planning of Parts Machining on a Number of Machines, *Automation and Remote Control*, 12, pp. 1972-1978.

Lins, L., Lins, S. & Morabito R., 2003, An L-Approach for Packing (L,W)- Rectangles into Rectangular and L-Shaped Pieces, *Journal of Operational Research Society*, 54, pp. 777-789.

Lodi, A., Martello, S. & Monaci, M., 2003, Two-dimensional Packing Problems: a survey, *European Journal of Operational Research*, 141, pp. 241-253.

Manne, A.S., 1960, On the Job-shop Scheduling Problem, *Operations Research*, 8, pp. 219-223.

Martínez, J.M. & Martínez, L., 2003, Packing Optimization for Automated Generation of Complex System's Initial Configurations for Molecular Dynamics and Docking, *Journal of Computational Chemistry*, 24, pp. 829-825.

McCormick, S.T., Pinedo, M.L., Shenker, S., & Wolf, B., 1989, Sequencing in an Assembly Line with Blocking to Minimize Cycle Time, *Operations Research*, 37, pp. 925-936.

Michael, L. & Van Hentenryck, P., 2004, A Simple Tabu Search for Warehouse Location, *European Journal of Operational Research*, 157, 3, pp. 576-591.

Mladenovic, N., Plastria, F. & Urosevic, D., 2005, Reformulation Descent Applied to Circle Packing Problems, *Computers & Operations Research*, 32, pp. 2419-2434.

Morabito, R. & Morales, S., 1998, A Simple and Effective Recursive Procedure for the Manufacturer's Pallet Loading Problem, *Journal of Operational Research Society*, 49, pp. 819-828.

Nagano, M.S. & Moccellin, J.V., 2002, A Hight Quality Solution Constructive Heuristic for Flow Shop Scheduling, *Journal of the Operational Research Society*, 12, pp. 1374-1379.

Nawaz, M., Enscore Jr, E. & Ham, I., 1983, A Heuristic Algorithm for the m-machine, n-job Flow Shop Sequencing Problem, *OMEGA*, 11, pp. 91-95.

Nemhauser, G.L., & Wosley, L.A., 1988, *Integer and Combinatorial Optimization*, Wiley, New York.

Nowicki, E., 1999, The Permutation Flow Shop with Buffers: A Tabu Search Approach, *European Journal of Operational Research*, 116, pp. 205-219.

Owen, S.H. & Daskin, M.S., 1998, Strategic Facility Location: A Review, *European Journal of* Operational *Research*, 111, pp. 423-447.

Pinedo, M., 1995, *Scheduling: theory, algorithms and systems*, Prentice-Hall, New Jersey.

Pitsoulis, L.S. & Resende, M.G.C., 2002, Greedy Randomized Adaptative Search Procedures, em *Handbook of Applied Optimization*, Eds. Pardalos, P.M. & Resende, M.G.C., Oxford University Press, pp. 168-183.

Potts, C.N., & Van Wassenhove, L.N., 1982, A Decomposition Algorithm for the Single Machine Total Tardiness Problem, *Operations Research Letters*, 1, pp. 177-181.

Rabinowitz, G. & Mehrez, A., 2001, A Multi-echelon Multi-commodity, Logistic System Design at the Dead Sea Works Ltd, *Computers & Industrial Engineering*, 39, pp. 65-79.

Raman, N., 1995, Minimum tardiness scheduling in flow shops: Construction and evaluation of alternative solution approaches, *Journal of Operations Management*, 12, pp. 131-151.

Rayward-Smith, V.J., Osman, I.H., Reeves C.R. & Smith, G.D., 1996, *Modern Heuristic Search Methods*, Wiley, New York.

Reddi, S.S., 1976, Sequencing with Finite Intermediate Storage , *Management Science*, 23, pp. 216-217.

Reddi, S.S., & Ramamoorthy, C.V., 1972, Flow-Shop Sequencing Problem with No Wait in Process, *Operational Research Quarterly*, 23, pp. 323-330.

Reklaitis, G.V., 1982, Review of Scheduling of Process Operations, *AIChE Symposium Series*, 78, pp. 119-133.

Resende, M.G.C. & Ribeiro C.C., 2003, Greedy Randomized Adaptive Search Procedures, em *Handbook of Metaheuristics*, Eds. Glover F. & Kochenberger G., Kluwer Academic Publishers, pp. 219-249.

Resende, M.G.C. & Ribeiro C.C., 2005, Grasp with Path-Relinking: Recent Advances and Applications, em *Metaheuristics: Progress as Real Problem Solvers*, Eds. Ibaraki, T., Nonobe, K. & Yagiura, M., Springer, pp. 29-63.

Resende, M.G.C. & Werneck, R.F., 2005, A Hybrid Multistart Heuristic for the Uncapacitated Facility Location Problem, aparecerá em *European Journal of Operational Research*.

Rios-Mercado, R.Z. & Bard J.F., 1998, Heuristics for the Flow Line Problem with Setup Costs, *European Journal of Operational Research*, 110, pp. 76-98.

Ronconi, D.P., 1997, Uma Contribuição para o estudo do Problema de *Flowshop* para Minimizar a Soma dos Atrasos, Tese de Doutorado, FEEC-Universidade Estadual de Campinas, Campinas/SP.

Ronconi, D.P., 2004, A Note on Constructive Heuristics for the Flowshop Problem with Blocking In-Process, *International Journal of Production Economics*, 87, pp. 39-48.

Ronconi, D.P., 2005, A Branch-and-Bound Algorithm to Minimize the Makespan in a Flowshop with Blocking, *Annals of Operations Research*, 138, pp. 53-65.

Ronconi, D.P. & Armentano V.A., 1993, Um Método Heurístico Baseado em Grafos para Formação de Células de Manufatura, *Caderno de Engenharia de Produção - DEP* , 21, pp. 24-48.

Ronconi, D.P., & Armentano, V.A., 1995 (publicado em 1997), Busca Tabu para a Minimização do Tempo Total de Atraso no Problema de *Flowshop*, *Pesquisa Operacional*, 15, pp. 23-34.

Ronconi, D.P. & Armentano V.A., 2001, Lower Bounding Schemes for Flowshops with Blocking In-Process, *Journal of the Operational Research Society*, 52, pp. 1289-1297.

Ronconi, D.P. & Henriques L.S., 2005, Heuristic Approaches For Total Tardiness Minimization In A Flowshop With Blocking, submetido para *Omega, The International Journal of Management Science*.

Ronconi, D.P. & Togashi, S.T., 2002, Um Estudo sobre Formulações Inteiras Mistas para Problemas de Programação no Ambiente *Flowshop* com Datas de Entrega, Anais do XI CLAIO - Congreso Latinoiberoamericano de Informática e Investigación Operativa, Concepción/Chile.

Sakuraba, C.S. & Ronconi D.P., 2006, Minimizing the sum of absolute deviations about a common due date in a two-machine flowshop, *working paper*.

Santoro, M. & Ronconi D.P., Modeling the Transportation and Warehousing of Potassium on the Fertilizers Industry: a Brazilian Case Study, submetido para *International Journal of Logistics*.

Sarper, H., 1995, Minimizing the Sum of Absolute Deviations about a Common Due Date for the Two-Machine Flow Shop Problem, *Applied Mathematical Modelling*, 19, pp. 153-161.

Sen, T. & Gupta, S. K., 1984, A State-of-Art Survey of Scheduling Research Involving Due Dates, *OMEGA*, 12, pp. 63-76.

Sharadapriyadarshini, B., & Rajendran, C., 1996, Formulations and heuristics for scheduling in buffer-constrained flowshop and flowline-based manufacturing cell with different buffer-space requirements for jobs: Part 1, *International Journal of Production Research*, 34, pp. 3465-3485.

Schuster, E.W. & Allen, S.J., 1998, Raw Material Management at Welch's, Inc., *Interfaces*, 28, pp. 13-24.

Stafford, E.F., 1988, On the Development of a Mixed-Integer Linear Programming Model for the Flowshop Sequencing Problem, *Journal of Operational Research Society*, 39, pp. 1163-1174.

Stafford Jr, E.F., Tseng, F.T. & Gupta, J.N.D., 2005, Comparative Evaluation of MILP Flowshop Models, *Journal of the Operational Research Society*, 56, pp. 88-101.

Stoyan, Y.G. & Yaskov, G., 2004, A Mathematical Model and a Solution Method for the Problem of Placing Various-Sized Circles into a Strip, *European Journal of Operational Research*, 156, pp. 590-600.

Suhami, I., & Mah, R.S.H., 1981, An Implicit Enumeration Scheme for the Flowshop Problem with No Intermediate Storage, *Computers and Chem. Eng.*, 5, pp. 83-91.

Suliman, S.M.A., 2000, A Two-Phase Heuristic Approach to the Permutation Flow-Shop Scheduling Problem. *International Journal of Production Economics*, 64, pp. 143-152.

Sung, C.S. & Min, J.I., 2001, Scheduling in a Two-Machine Flowshop with Batch Processing Machine(s) for Earliness/Tardiness Measure under a Common Due Date, *European Journal of Operational Research*, 131, pp. 95-106.

Taillard, E.,1993, Benchmarks for basic scheduling problems, *European Journal of Operational Research*, 64, pp. 278-285.

Talbi, E.-G., 2002, A Taxonomy of Hybrid Metaheuristics, *Journal of Heuristics*, 8, pp. 541-564.

Tapiero, C.S., 1971, Transportation-Location-Allocation Problems Over Time, *Journal of Regional Science*, 11, pp. 377-384.

Tarnowski, A., Terno, J. & Scheithauer, G., 1994, A Polynomial-Time Algorithm For The Guilhotine Pallet Loading Problem, *INFOR*, 32, pp. 275-287.

Taube-Netto, M., 1996, Integrated Planning for Poultry Production at Sadia, *Interfaces*, 26, pp. 38-53.

Van Roy, T.J. & Erlenkotter, D., 1982, A Dual-Based Procedure for Dynamic Facility Location, *Management Science*, 28, pp. 1091-1105.

Wagner, H.M., 1959, An Integer Linear-programming Model for Machine Scheduling, *Naval Research Logistics*, 6, pp. 131-140.

Wang, H., Huang, W., Zhang, Q. & Xu, D., 2002, An Improved Algorithm for the Packing of Unequal Circles within a Larger Containing Circle, *European Journal of Operational Research*, 141, pp. 440-453.

Williams, H.P., 1999, *Model Building in Mathematical Programming*, John Wiley & Sons, New York.

Yeung, W.K.; Oguz, C. & Cheng, T.C.E., 2003, Two-Stage Flowshop Earliness and Tardiness Machine Scheduling Involving a Common Due Window, *International Journal of Production Economics*, 90, pp. 421-434.

Yoshizaki, H.T.Y., Muscat, A.R.N. & Biazzi, J.L., 1996, Decentralizing Ethanol Distribution in South Eastern Brazil, *Interfaces*, 26, pp. 24-34.

Zanakis, S.H., Evans, J.R. & Vazacopoulos, A.A., 1989, Heuristic Methods and Applications: A Categorized Survey, *European Journal of Operational Research*, 43, pp. 88-110.