CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Diretoria de Pesquisa e Pós-Graduação

Programa de Mestrado em Modelagem Matemática e Computacional

HEURÍSTICAS PARA O PROBLEMA DE SEQUENCIAMENTO EM UMA MÁQUINA COM PENALIDADES POR ANTECIPAÇÃO E ATRASO DA PRODUÇÃO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, como parte dos requisitos necessários para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

Aluno: Bruno Ferreira Rosa

Orientador: Prof. Dr. Marcone Jamilson Freitas Souza (UFOP)

Co-Orientador: Prof. Dr. Sérgio Ricardo de Souza (CEFET-MG)

Belo Horizonte, Dezembro de 2009.

Dedico este trabalho aos meus pais José Lázaro e Isilene, e à minha namorada Fernanda, que estão ao meu lado em todos os momentos e foram os principais motivadores deste trabalho. Amo muito vocês!

Agradecimentos

Agradeço a Deus pela vida, por me guardar nas muitas viagens necessárias para a execução deste trabalho.

Aos meus pais pelo exemplo de amor, união, honestidade e determinação.

Aos meus irmãos pela amizade e por estarem sempre ao meu lado, principalmente nos momentos de descontração.

Ao grande amor da minha vida Fernanda por ser minha confidente, minha companheira, pela compreensão e pelo seu amor incondicional.

A todos os meus familiares que sempre me apoiaram em minhas escolhas.

Aos professores Marcone e Sérgio por me aceitarem, por acreditarem na minha capacidade, pelas essenciais orientações, pela paciência e disposição em me ajudar.

Aos professores e funcionários do Programa de Pós-Graduação do CEFET-MG pela oportunidade de buscar e aprimorar meus conhecimentos.

Aos amigos do MMC que estiveram presentes na execução deste trabalho. A amizade e colaboração de vocês foram fundamentais.

Aos colegas que fiz na "estrada" pelas caronas, pela confiança depositada em mim e por compartilharem suas histórias de vida.

Ao CEFET-MG pelo apoio e incentivo.

Enfim, os meus sinceros agradecimentos a todos aqueles que colaboraram, mesmo que indiretamente, para a realização deste trabalho.

Resumo

Este trabalho trata do problema de sequenciamento de tarefas em uma máquina. No problema abordado, os tempos de preparação da máquina são dependentes da sequência de produção e cada tarefa está associada a um tempo de processamento e a uma janela de tempo, dentro da qual ela deve ser preferencialmente concluída. O objetivo é minimizar a soma ponderada dos atrasos e das antecipações na execução de tais tarefas. Em termos práticos, as penalidades por antecipação são devidas aos custos gerados pela necessidade de estocagem, enquanto as penalidades por atraso são consequências de multas contratuais. São propostas duas novas formulações de programação matemática para representar o problema, sendo a primeira delas um aperfeiçoamento de uma formulação da literatura e, a outra, uma formulação indexada no tempo. Propôs-se, também, um algoritmo heurístico para determinar a melhor sequência de produção. Tal algoritmo é composto de duas fases. Na primeira, gera-se uma solução com base na metaheurística GRASP, no Princípio da Otimalidade Próxima e na Descida em Vizinhança Variável. Na segunda, faz-se o pós-refinamento da solução proveniente da fase anterior por meio de outra Descida em Vizinhança Variável. Com o objetivo de reduzir o custo computacional, adicionalmente é proposta uma estratégia de redução do espaço de busca. Devido à particularidade dos problemas-teste da única base de dados existente na literatura referente ao problema tratado, foram geradas outras duas novas bases de dados, uma das quais mais genérica. Experimentos computacionais mostram que a formulação indexada no tempo possibilita resolver problemas maiores e com maior eficiência, quando comparada com as formulações de programação matemática da literatura. O algoritmo heurístico proposto mostrou-se competitivo com os demais algoritmos existentes na literatura, tendo o baixo custo computacional como o seu grande aliado. Desta forma, foi possível resolver, em tempo computacional viável, problemas de dimensões maiores que os resolvidos até então.

Palavras-Chave: Sequenciamento em uma máquina, Formulação indexada no tempo, GRASP, Princípio da Otimalidade Próxima, Descida em Vizinhança Variável.

Abstract

This work deals with the problem of job scheduling in a single-machine. The matter approached is that the machine's setup time depends on the production sequence and each job has a processing time and a time window within which it should preferably be completed. The aim is to minimize the weighted sum of the tardiness and earliness as they are working. In practical terms, the advance penalties are due to the costs incurred by the need for storage, whereas late-payment penalties are the contractual penalties result. It was proposed two new mathematical programming formulations in order to represent the problem, firstly an improvement of the literature formulation, and then a time-indexed formulation. It was also proposed an heuristic algorithm to determine the best production sequence. This algorithm is composed of two phases. First, it would generate a solution based on GRASP, Proximate Optimality Principle and Variable Neighborhood Descent. Second, post-refinement of the solution would be made from the previous phase through another Variable Neighborhood Descent. In order to reduce the computational cost it was also proposed a strategy to reduce the search space. Due to the distinctive characteristics of the test-problem existent in the database of the current literature, we generated two other new databases, being one of them more generic. Computational experiments show that time-indexed formulation allows us to solve larger problems in a more efficient way than other mathematical programming formulations do. The heuristic algorithm proposed here proved to be as good as the algorithms in the literature, having the advantage of the low computational costs as its ally. Thus, it was possible to resolve, in feasible computing time, problems of greater dimensions than the ones solved until now.

Keyword: Single-machine scheduling, Time-indexed formulation, GRASP, Proximate Optimality Principle, Variable Neighborhood Descent.

Sumário

Lista de Abreviaturas e Siglas

ADDOIPT Algoritmo de Determinação das Datas Ótimas de Início de Processamento das Tarefas

AE Algoritmos Evolutivos

AG Algoritmos Genéticos

BB *Branch-and-Bound*

BDDT Base de dados em que os problemas satisfazem a desigualdade triangular

BDS Base de dados em que os tempos de *setup* são simétricos

BDNS Base de dados em que os problemas não satisfazem, necessariamente, à uma determinada propriedade

BL Busca Local

EDD *Earliest Due Date* - data de entrega mais cedo

FIFO *First-In-First-Out* - o primeiro que entrar é o primeiro a sair

GPV Algoritmo baseado no GRASP, no POP e no VND

GPV-REB Algoritmo GPV que faz uso da estratégia de redução do espaço de busca

GRASP *Greedy Randomized Adaptive Search Procedure* - Procedimento de busca adaptativa gulosa e randomizada

HC Heurísticas Construtivas

ILS *Iterated Local Search*

JIT *Just in Time*

LRC Lista Restrita de Candidatos

MDR Método de descida randômica

PD Programação Dinâmica

PLIM Programação Linear Inteira Mista

POP Princípio da Otimalidade Próxima

PR *Path Relinking* - Reconexão por Caminhos

PSUMAA Problema de sequenciamento em uma máquina com penalidades por antecipação e atraso na produção

PSUMAA-JE-TPD PSUMAA com janelas de entrega distintas, tempos de *setup* dependentes da sequência de produção e tempos ociosos entre as execuções de tarefas consecutivas permitidos

SA *Simulated Annealing*

SS *Scatter Search*

TS *Tabu Search* - Busca Tabu

VND *Variable Neighborhood Descent* - Descida em Vizinhança Variável

Lista de Figuras

Lista de Algoritmos

Lista de Tabelas

Capítulo 1

Introdução

O surgimento do sistema de administração *Just in Time* (JIT), que ocorreu nos meados da década de 70, evidenciou a importância de um planejamento criterioso das atividades produtivas. Visando a redução dos custos provenientes de processos produtivos, a filosofia JIT desencoraja, além dos atrasos, também as antecipações das tarefas. Há antecipação de uma tarefa quando ela é concluída antes da data desejada para sua entrega e há atraso quando ela é concluída após tal data. Deste modo, ambas as situações acarretam penalidades.

De acordo com Liaw (1999), concluir uma tarefa com atraso pode resultar em multas contratuais, perda de credibilidade da empresa e redução de vendas. Do mesmo modo, concluir uma tarefa antecipadamente pode resultar em custos financeiros extras pela necessidade de disponibilização antecipada de capital, necessidade de espaço para armazenamento ou necessidade de outros recursos para manter e gerenciar o estoque (França Filho, 2007).

O problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção, de agora em diante denotado por PSUMAA, consiste em sequenciar e determinar o momento em que as tarefas devem ser executadas em uma máquina, com o objetivo de minimizar a soma ponderada das antecipações e dos atrasos na produção de tais tarefas. Tal problema, segundo Baker & Scudder (1990), reflete melhor ambientes de produção administrados conforme a filosofia JIT.

Outros objetivos relacionados a essa filosofia de produção também são encontrados na literatura, entre eles: minimizar a soma das penalidades por antecipações e atrasos juntamente com os tempos ociosos gerados durante a produção das tarefas (Sourd, 2005), minimizar a soma da maior antecipação com o maior atraso ocorridos na sequência de execução das tarefas (Tavakkoli-Moghaddam *et al.*, 2006) e minimizar a soma dos quadrados das antecipações e dos atrasos ocorridos no decorrer do processo de produção (Weng & Ventura, 1996; Mondal, 2002).

Com relação às datas de entrega das tarefas, o PSUMAA pode ser dividido em três variantes:

(i) datas de entrega comuns (*common due date*): todas as tarefas devem ser, preferencialmente, concluídas em uma única data pré-determinada;

(ii) datas de entrega distintas (*distinct due dates*): existe uma data de entrega específica associada a cada tarefa, na qual a tarefa deve ser preferencialmente concluída;

(*iii*) janelas de entrega distintas (*distinct due windows*): há um determinado período de tempo associado a cada tarefa, dentro do qual a tarefa deve ser preferencialmente concluída.

O caso de janelas de entrega distintas ocorre quando existem tolerâncias em torno das datas desejadas para a entrega de cada tarefa (Koulamas, 1996). Estas tolerâncias estão relacionadas às características individuais das tarefas e influenciam nos tamanhos das janelas de entrega. As tarefas concluídas dentro de suas respectivas janelas de entrega não incorrem em nenhuma penalidade. Já aquelas concluídas fora de suas janelas de entrega, são penalizadas. A produção de bens perecíveis é um exemplo relacionado. Assuma que um fabricante de produtos químicos combina uma certa substância A, que deteriora rapidamente, com uma segunda substância B para produzir um produto C. Se A for produzida suficientemente antes de B, ela deteriorará. Por outro lado, se A for produzida muito depois de B, o custo da produção de C será maior.

Nas indústrias em que são produzidos diferentes tipos de produtos e existe uma troca frequente do tipo de tarefa executada em uma máquina, após a conclusão de uma tarefa, geralmente é necessário preparar a máquina antes do início da execução da tarefa seguinte (Allahverdi *et al.*, 1999). Este tempo de preparação, chamado tempo de *setup*, inclui os tempos gastos para trocar as ferramentas, preparar o material, limpar a máquina etc. A maioria dos trabalhos em problemas de sequenciamento assume que os tempos de *setup* são independentes da sequência de produção, isto é, que eles são desprezíveis ou podem ser acrescentados aos tempos de processamento das tarefas (Gupta & Smith, 2006). No entanto, de acordo com Panwalkar *et al.* (1973), *apud* Gupta & Smith (2006), em grande parte das situações práticas, tais tempos são dependentes da sequência de produção. Christofoletti (2002) cita o exemplo de uma fábrica de papel que produz diversos tipos de folhas com diferentes cores, espessuras e texturas e que realiza frequentemente a preparação de certas máquinas para obter os diferentes tipos de produtos. Em problemas como este, os tempos de *setup* variam de acordo com sequência de produção e representam uma parcela de tempo considerável em relação ao tempo total de processamento. Portanto, eles não podem ser desconsiderados.

Considerações sobre a continuidade do funcionamento da máquina também podem ser impostas ao PSUMAA. Parte dos trabalhos da literatura não permite a inserção de tempos ociosos na sequência de produção (Chang, 1999). Conforme Li (1997), existem casos em que o custo por manter a máquina inativa é maior que o preço pago pela antecipação de uma tarefa e, neste caso, vale a pena antecipar a produção. Outra situação em que isto ocorre é quando a capacidade da máquina é inferior à demanda. No entanto, há casos em que vale a pena manter a máquina parada, mesmo que exista uma tarefa disponível para ser processada (Souza *et al.*, 2008). Para ilustrar esta última situação, considere duas tarefas i e j, que devem ser consecutivas em uma sequência de produção. À tais tarefas estão associados, respectivamente, custos unitários por atraso de 100 e 20, custos unitários por antecipação de 22 e 2, datas de entrega iguais a 50 e 55, e 15 e 25 unidades de tempo de processamento. Suponha que a máquina já esteja disponível desde o instante 20. Se a tarefa i começar no instante 20, será concluída no instante 35 e, portanto, haverá uma penalização igual a 330 ($= 15 \times 22$) pela sua antecipação. Nesse caso, a tarefa j poderá começar somente no instante 35. Considerando seu tempo de processamento, ela será concluída no instante 60, portanto 5 unidades de tempo atrasada em relação à data de entrega, resultando em um custo por atraso igual a 100 ($= 5 \times 20$), e um custo total

de 430 (=330 + 100). Considere, agora, que o processamento da tarefa i seja iniciado no instante 35, mesmo sabendo que a máquina está disponível desde o instante 20. Neste caso, a tarefa i seria concluída no instante 50, sem multa por antecipação ou por atraso. A tarefa j, então, já poderia começar, terminando no instante 75, com 20 unidades de tempo de atraso, resultando em uma multa por atraso de 400 ($= 20 \times 20$). O custo total nesta última situação seria de 400 unidades, menor que aquele no qual a tarefa i começava seu processamento no instante de liberação da máquina. Assim, não havendo restrições à ociosidade das máquinas, determinar a melhor data para iniciar o processamento de cada tarefa ou, equivalentemente, inserir tempos ociosos entre tarefas, poderá produzir melhores soluções.

Este trabalho trata o PSUMAA com janelas de entrega distintas, tempos de preparação dependentes da sequência de produção e são permitidos tempos ociosos entre as execuções de tarefas consecutivas. Para clareza de entendimento e evitar dúvidas, o PSUMAA com tais características será doravante denotado por PSUMAA-JE-TPD. Apesar de este problema ser uma generalização do PSUMAA, ele não tem recebido a atenção merecida na literatura.

1.1 Objetivos

Nesta Seção são apresentados os objetivos deste trabalho.

1.1.1 Objetivo Geral

Este trabalho trata o problema de sequenciamento de tarefas em uma máquina com penalidades por antecipação e atraso da produção na sua forma mais genérica. O objetivo principal é desenvolver um algoritmo heurístico robusto e capaz de encontrar soluções de boa qualidade para tal problema em tempo de execução viável.

1.1.2 Objetivos Específicos

Com a finalidade de alcançar o objetivo geral, os seguintes objetivos específicos foram traçados:

- Fazer um levantamento dos trabalhos da literatura que tratam o problema abordado e problemas relacionados, bem como os métodos utilizados para resolvê-los;
- Gerar novos conjuntos de problemas-teste do problema em estudo;
- Desenvolver um algoritmo heurístico para resolver o problema;
- Desenvolver novos modelos de programação matemática para representar o problema abordado;
- Resolver os problemas-teste de dimensões menores por meio dos modelos de programação matemática desenvolvidos e disponíveis na literatura;
- Resolver os problemas-teste disponíveis por meio do algoritmo heurístico implementado e, em seguida, analisar e discutir os resultados encontrados.

- Divulgar os resultados do trabalho em eventos científicos, bem como em periódicos especializados.

1.2 Motivação

O PSUMAA com janelas de entregas distintas e tempos de *setup* dependentes da sequência de produção possui muitas aplicações práticas em indústrias metalúrgicas, têxteis, de tintas, entre outras. O caso real de uma indústria siderúrgica é tratado em Bustamante (2007). Em tal trabalho, uma máquina é considerada como sendo uma sequência de laminadores e cada tarefa representa a produção de um determinado produto (barra chata, cantoneira, vergalhão etc.). Antes da fabricação de cada produto, é necessário realizar um conjunto de ajustes de mesma natureza na sequência de laminadores. Estes ajustes dependem do produto a ser fabricado e do produto fabricado anteriormente. Como cada ajuste exige um tempo de execução, a soma destes tempos configura o tempo de *setup*. Esta situação é ilustrada na Figura 1.1.

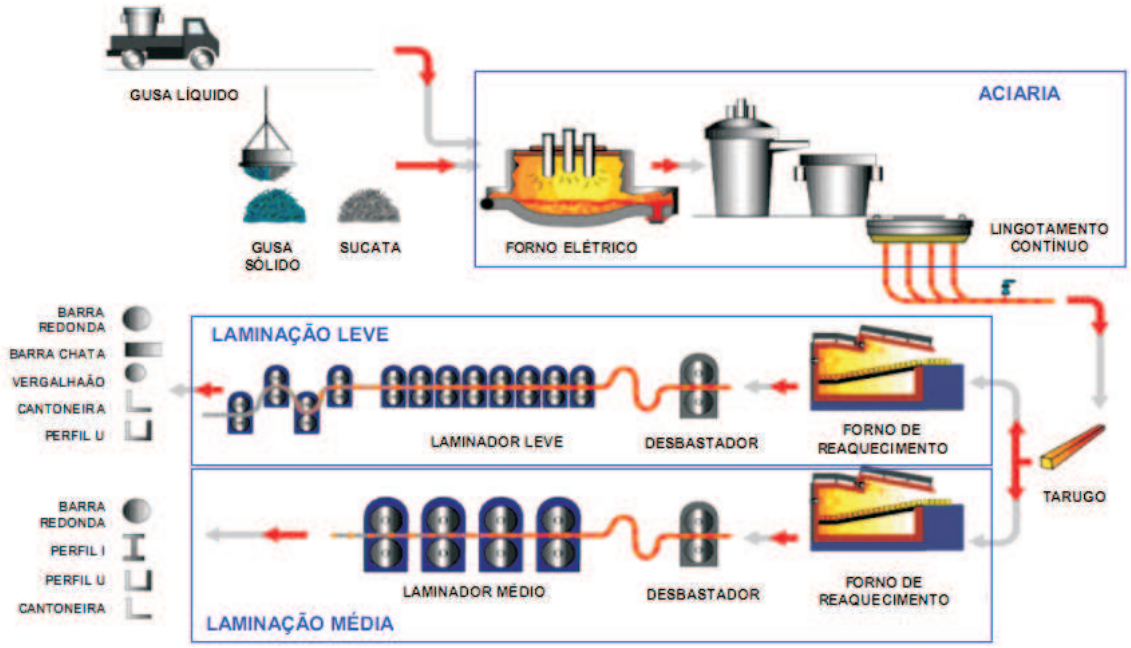

Figura 1.1: Aplicação em uma indústria siderúrgica.

Além do grande número de aplicações, associado ao PSUMAA está a dificuldade de resolvê-lo de forma ótima. Tal dificuldade cresce fatorialmente à medida em que se aumenta a quantidade de tarefas a serem sequenciadas. Esta união entre aplicabilidade e dificuldade de resolução motiva o desenvolvimento de algoritmos eficientes para resolvê-lo.

A utilização de métodos exatos pode demandar um tempo proibitivo para encontrar a solução ótima do PSUMAA, mesmo para problemas de médio porte. Isso justifica a

utilização de algoritmos heurísticos, que são procedimentos capazes de encontrar uma "boa solução", não necessariamente a solução ótima, em tempo computacional viável.

Neste trabalho são utilizados o GRASP (Feo & Resende, 1995) e o Princípio da Otimalidade Próxima (Glover & Laguna, 1997) como base para o algoritmo proposto devido aos seguintes motivos: o primeiro por ser uma metaheurística que possui poucos parâmetros para serem ajustados e que se mostrou eficiente na resolução de diversos problemas combinatórios e o segundo, para refinar as soluções parciais e, assim, reduzir o custo computacional da fase de busca local do GRASP.

O desenvolvimento de novas formulações de programação matemática para o problema é motivada pelo fato de existir na literatura apenas um modelo que representa o PSUMAA-JE-TPD. A utilização de um modelo indexado no tempo é motivada pelo trabalho de de Paula (2008), o qual utilizou tal modelagem num problema relacionado e obteve bons resultados. Uma formulação de programação matemática similar à de Gomes Júnior (2007), porém com menor número de variáveis e de restrições, também é proposta na esperança de que ela possibilite resolver problemas de maiores dimensões, quando comparada à formulação de tal autor.

1.3 Estrutura do Trabalho

O restante deste trabalho está organizado como segue.

No Capítulo 2 é feita uma descrição detalhada do PSUMAA-JE-TPD. Um resumo das principais técnicas pertinentes à resolução desse problema também é apresentado. O capítulo é encerrado com a descrição dos trabalhos mais relevantes da literatura em relação ao PSUMAA-JE-TPD.

A formulação de programação matemática proposta por Gomes Júnior (2007) para representar o problema, uma nova formulação baseada nesta última e um modelo de programação matemático indexado no tempo são apresentados no Capítulo 3.

No Capítulo 4 é apresentado o algoritmo heurístico proposto para resolver o PSUMAA-JE-TPD.

Os resultados computacionais obtidos com as formulações de programação matemática propostas e com o algoritmo heurístico desenvolvido são apresentados e discutidos no Capítulo 5.

O Capítulo 6 conclui o trabalho e aponta perspectivas para aperfeiçoamento do algoritmo proposto.

Capítulo 2

Problema Abordado e Métodos de Resolução

Neste capítulo é feita uma apresentação das características do problema abordado, seguida pela apresentação dos métodos existentes para resolver problemas similares. Na Seção 2.3 é feita uma revisão da literatura relacionada ao PSUMAA-JE-TPD e problemas correlatos.

2.1 Características do PSUMAA-JE-TPD

O problema de sequenciamento de tarefas em uma máquina tratado neste trabalho (PSUMAA-JE-TPD) possui as seguintes características:

- Uma máquina deve processar um conjunto I de n tarefas;
- Associado a cada tarefa $i \in I$ está:
  - um tempo de processamento P_i;
  - uma janela de entrega $[E_i, T_i]$ dentro da qual a tarefa i deve ser preferencialmente concluída;
  - um custo α_i por unidade de tempo de antecipação;
  - um custo β_i por unidade de tempo de atraso;
- Há antecipação de uma tarefa $i \in I$ quando seu processamento é concluído antes de E_i. O tempo de antecipação de i é dado por $e_i = \max(0,\ E_i - C_i)$, onde C_i representa a data de conclusão de i;
- Há atraso de uma tarefa $i \in I$ quando seu processamento é concluído depois de T_i. O tempo de atraso de i é dado por $t_i = \max(0,\ C_i - T_i)$, onde C_i representa a data de conclusão de i;
- As tarefas que forem concluídas dentro de suas respectivas janelas de entrega não geram penalidades;

- A máquina executa no máximo uma tarefa por vez e uma vez iniciado o processamento de uma tarefa, não é permitida a sua interrupção;
- Todas as tarefas estão disponíveis para processamento na data 0;
- Entre duas tarefas i e $j \in I$ consecutivas é necessário um tempo S_{ij} de preparação da máquina, chamado tempo de *setup*. Assume-se que o tempo de preparação da máquina para o processamento da primeira tarefa na sequência é igual a 0;
- É permitido tempo ocioso entre a execução de duas tarefas consecutivas.

O objetivo é determinar uma sequência de produção e as datas de início de produção das tarefas de forma a minimizar a soma ponderada dos tempos de antecipações e de atrasos, ou seja, minimizar o valor de Z na equação (2.1):

$$Z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{2.1}$$

É interessante observar que no objetivo do PSUMAA-JE-TPD está embutido dois subproblemas que devem ser resolvidos simultaneamente. Um deles é determinar a sequência de produção, isto é, determinar a ordem em que as tarefas serão realizadas. O outro subproblema consiste em programar a data de início de produção de cada tarefa de forma que a soma ponderada das antecipações e atrasos seja a menor possível.

A Figura 2.1 ilustra dois posicionamentos possíveis para as tarefas de uma mesma sequência de produção envolvendo três tarefas. No primeiro posicionamento, as datas de início de processamento das tarefas foram programadas de modo que a tarefa 3 é concluída antecipadamente, acarretando em penalidade por antecipação, e a tarefa 2 dentro de sua janela de entrega, não gerando penalidades. Já no segundo posicionamento, as datas de início de processamento das tarefas foram programadas de forma que a tarefa 2 é concluída com atraso, gerando penalidade, e a tarefa 3 é concluída dentro de sua janela de entrega, não agregando penalidade. Nesta figura, também são ilustrados os tempos de *setup* e de ociosidade de máquina. De acordo com os valores de penalidade incorridos, um destes posicionamentos será o mais adequado para as tarefas desta sequência. Por exemplo, considerando que o tempo de antecipação da tarefa 3 no posicionamento 1 é igual ao tempo de atraso da tarefa 2 no posicionamento 2, o posicionamento 2 será mais vantajoso que o posicionamento 1, caso $\alpha_3 > \beta_2$, e, caso contrário, o posicionamento 1 será mais vantajoso que o posicionamento 2.

2.2 Métodos de Resolução

O problema de sequenciamento de tarefas é, em seu caso geral, de natureza combinatória. Apesar de possuir domínio finito (para um problema com n tarefas existem $n!$ sequências possíveis) e de simples enumeração, a ideia de testar todas as possíveis sequências é computacionalmente inviável, mesmo para uma quantidade moderada de tarefas. Por exemplo, para o problema com 16 tarefas, um computador que avalia uma sequência em aproximadamente 10^{-6} segundos, gastaria mais de 8 meses para encontrar a melhor sequência.

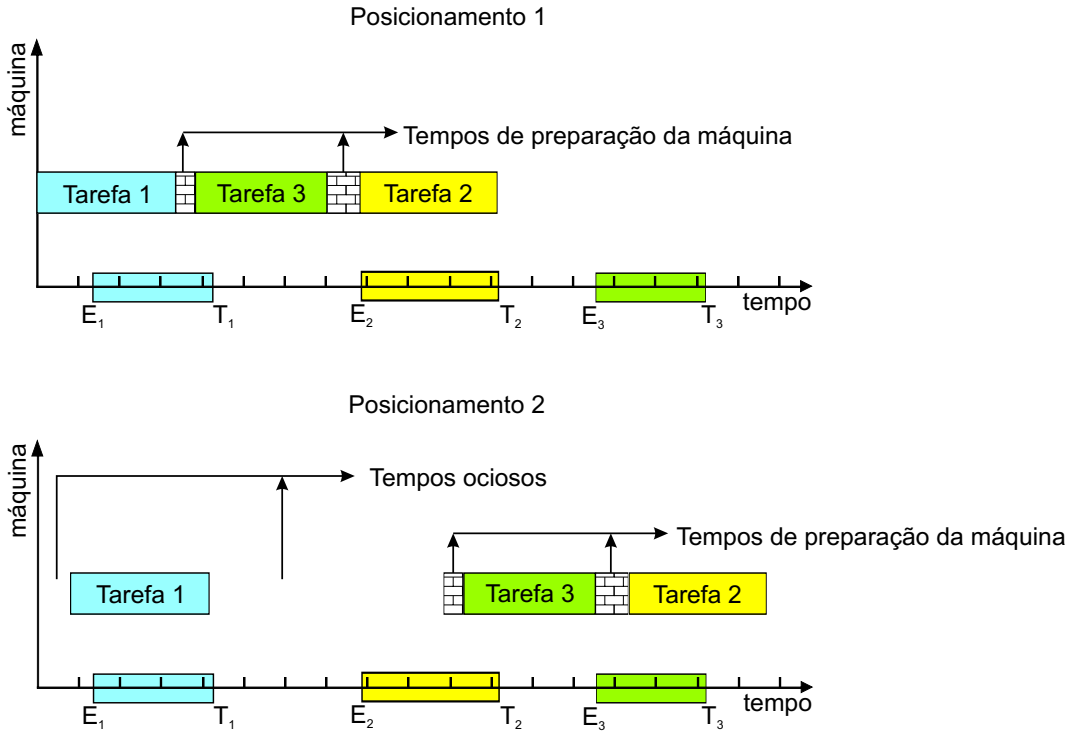

Figura 2.1: Exemplo de dois posicionamentos distintos para as tarefas de uma mesma sequência de produção.

O PSUMAA com datas de entrega comum é NP-difícil (Baker & Scudder, 1990), ou seja, ainda não é conhecido nenhum algoritmo de complexidade polinomial que o resolva. Como o caso de datas de entrega comum pode ser visto como um caso particular do problema abordado neste projeto, tem-se que o PSUMAA-JE-TPD também é NP-difícil.

Os métodos utilizados para resolver problemas combinatórios, e consequentemente o PSUMAA, podem ser divididos em dois grupos distintos: métodos exatos e métodos heurísticos. Estes métodos são descritos nas seções seguintes.

2.2.1 Métodos Exatos

Os métodos exatos são algoritmos determinísticos, fundamentados em programação matemática e capazes de encontrar a solução ótima de um problema. Uma solução é dita globalmente ótima, ou simplesmente ótima, se, sob um certo critério de otimização, não existir outra solução para o problema com o valor melhor que o dela. Os algoritmos exatos varrem, implicitamente ou não, o conjunto das soluções possíveis para um problema e armazenam a melhor solução encontrada. Por isso, tem-se a garantia de que a solução retornada é, ou não é, uma solução ótima do problema tratado. Além disso, é possível mensurar o quanto a melhor solução encontrada até um dado instante é inferior à possível solução ótima.

Apesar de serem capazes de encontrar a solução ótima de um problema, devido ao elevado custo computacional, esses métodos são limitados à resolução de problemas de pequeno porte. Como exemplificado anteriormente, para problemas combinatórios, o número de soluções possíveis cresce abruptamente à medida em que se aumenta a dimensão

do problema. Deste modo, a simples enumeração das soluções, ainda que implicitamente, se torna inviável para problemas de dimensão mediana.

Os métodos exatos ainda possuem como desvantagens: modelagem mais complexa e nem sempre conseguem encontrar uma solução viável rapidamente. Apesar de suas limitações, os algoritmos exatos são de grande importância na validação de algoritmos não exatos.

A seguir é feita uma breve apresentação de alguns dos métodos exatos já utilizados para resolver o PSUMAA.

Enumeração Total

O método de enumeração total é o mais intuitivo e simples de se implementar entre os algoritmos exatos. Ele consiste em enumerar todas as soluções possíveis e compará-las entre si. A melhor solução é retornada. Este método é fortemente limitado aos problemas com poucas tarefas.

Branch-and-Bound

O método *Branch-and-Bound* (BB) é um algoritmo de enumeração implícita, ou seja, um método dotado de uma certa "sabedoria", que retorna a solução ótima de um problema de otimização mesmo enumerando apenas parte do conjunto de soluções possíveis. O BB é baseado no paradigma de divisão e conquista. Tal paradigma consiste em dividir o problema a ser resolvido em partes menores, encontrar soluções para as partes, e então combinar as soluções obtidas em uma solução global (Ziviane, 2004).

De acordo com Gray *et al.* (1997), o método inicia-se considerando o problema original com todas as possíveis soluções, dito problema raiz. Em seguida, são aplicados procedimentos que calculam limitantes superior e inferior para o problema raiz. Se estes limites forem iguais, então uma solução ótima foi encontrada e o procedimento termina. Caso contrário, o problema é dividido em subproblemas menores. Estes subproblemas se tornam filhos do problema raiz. O algoritmo é aplicado recursivamente, gerando assim uma árvore de subproblemas. Quando a solução ótima de um subproblema é encontrada, os demais subproblemas deixam de ser considerados, ou seja, eles são eliminados da árvore de subproblemas. A busca continua até que todos os subproblemas sejam resolvidos ou eliminados, ou até atingir um dado limiar entre a melhor solução encontrada e os limitantes inferiores de todos os subproblemas não resolvidos.

Um limitante superior para um problema de minimização é um valor superior ou igual ao valor ótimo deste problema. O valor da função objetivo de qualquer solução factível, por exemplo, é um limitante superior para o problema. Analogamente, um limitante inferior para este problema é um valor para o qual tem-se a garantia de que a solução ótima é superior ou igual a ele.

Para o cálculo de limites superiores podem ser utilizadas heurísticas, enquanto os limites inferiores são geralmente obtidos por relaxação do problema original. Relaxar um problema consiste em considerar apenas parte de suas restrições. Deste modo, tem-se que a solução do problema original também é uma solução para o problema relaxado. Esta técnica parte do princípio de que o problema relaxado é mais fácil de ser resolvido quando comparado com o problema original. Dentre as técnicas de relaxação existentes,

a mais utilizada na resolução do PSUMAA é a relaxação lagrangeana. A técnica de relaxação lagrangeana consiste em adicionar um conjunto de restrições à função objetivo do problema original e resolver o problema relaxado iterativamente. Cada restrição é adicionada com um determinado peso, denominados *multiplicadores de Lagrange*.

Programação Dinâmica

A técnica de programação dinâmica (PD) consiste em dividir o problema em subproblemas. O cálculo das soluções dos subproblemas é feito a partir dos subproblemas menores para os maiores e os resultados são armazenados em uma tabela. Deste modo, uma vez que um subproblema é resolvido, a resposta é armazenada em uma tabela e nunca mais é recalculada. De acordo com Ziviane (2004), a PD é indicada quando a divisão de um problema de tamanho n resulta em n subproblemas de tamanho $n - 1$ facilmente resolvíveis.

2.2.2 Métodos Heurísticos

Como dito anteriormente, o PSUMAA pertence ao conjunto dos problemas de otimização combinatória, para os quais encontrar uma solução ótima de forma exata é uma tarefa difícil. Mesmo com a utilização de estratégias que reduzem o espaço de busca (ver seção 2.2.1), haverá sempre uma quantidade de tarefas a partir da qual a utilização de métodos exatos necessita de um tempo de processamento inviável. Além disso, na maioria das situações práticas, é mais interessante encontrar rapidamente uma "boa solução", ao invés de perder tempo à procura da solução ótima. Por isso, os métodos heurísticos têm sido muito utilizados na resolução de problemas de natureza combinatória.

Souza (2008) define heurística como sendo uma técnica inspirada em processos intuitivos que procura uma boa solução a um custo computacional aceitável, sem, no entanto, estar capacitada a garantir sua otimalidade, bem como garantir quão próximo está da solução ótima.

Segundo Weise (2008), uma heurística é uma parte de um algoritmo de otimização que usa informações conhecidas para ajudar a decidir qual solução candidata será a próxima a ser testada ou qual a próxima solução a ser produzida. Normalmente, as heurísticas são dependentes das classes de problemas.

Ainda de acordo com Weise (2008), normalmente empregam-se heurísticas em algoritmos determinísticos para definir a ordem de processo das soluções candidatas. Já algoritmos probabilísticos podem considerar somente os elementos do espaço de busca que foram selecionados por uma heurística. Isto se deve ao fato de que, apesar de não garantirem a otimalidade da solução obtida, os métodos heurísticos são de fácil implementação e podem gerar boas soluções rapidamente.

Deste modo, uma heurística pode ser vista como uma técnica que se utiliza de informações particulares dos problemas para encontrar boas soluções a baixo custo computacional.

As heurísticas podem ser divididas em duas classes diferentes: heurísticas construtivas e heurísticas de refinamento. As seções seguintes são destinadas a descrever tais classes.

Heurísticas Construtivas

Heurísticas construtivas (HC) são algoritmos que têm por objetivo a construção de uma solução de forma incremental. A cada passo um elemento é inserido na solução parcial de acordo com uma função de avaliação, que por sua vez depende do problema em questão, até gerar uma solução completa. Nas heurísticas clássicas, os elementos candidatos são geralmente ordenados segundo uma função gulosa, que estima o benefício da inserção de cada elemento, e somente o "melhor" elemento é inserido a cada passo (Souza, 2008). As HC gulosas (heurísticas que utilizam uma função gulosa) são determinísticas e muito utilizadas para gerar soluções iniciais que, possivelmente, serão melhoradas por outros procedimentos heurísticos, ou seja, as soluções gulosas podem ser utilizadas como um limiar superior da solução.

Uma heurística construtiva frequentemente encontrada na literatura para o PSUMAA com datas de entrega distintas é a EDD (*Earliest Due Date* - data de entrega mais cedo). A EDD avalia uma tarefa de acordo com a data de início da sua respectiva janela de entrega. A cada passo, a tarefa com menor data de entrega e ainda não sequenciada é inserida no final da subsequência corrente. O algoritmo é interrompido quando não houver mais tarefa a ser sequenciada.

Outra maneira de se construir uma solução é de forma aleatória, ou seja, em cada passo, o elemento que será inserido na solução parcial é escolhido aleatoriamente dentre os elementos que ainda estão fora dela. Este método é bastante utilizado devido sua facilidade de implementação e por gerar soluções diferentes, isto é, não é determinístico. Porém, testes empíricos mostram que as soluções obtidas geralmente são de baixa qualidade e, por isso, necessitam de um esforço maior na fase de refinamento (Souza, 2008).

Heurísticas de Refinamento

As heurísticas de refinamento são algoritmos que têm por objetivo melhorar uma dada solução inicial por meio de modificações em suas componentes. Uma solução inicial pode ser obtida, por exemplo, através de uma HC. As heurísticas pertencentes a esta classe também são conhecidas como técnicas de busca local (BL) e são baseadas na noção de vizinhança.

Seja S o conjunto de todas as soluções possíveis para um determinado problema de otimização e $f : S \to \mathbb{R}$ a função objetivo a ser minimizada[1]. Uma vizinhança é qualquer função $V : S \to P(S)$, onde $P(S)$ representa o conjunto das partes de S. Isto é, vizinhança é uma função que leva cada solução $s \in S$ em um subconjunto $V(s) \subseteq S$. Um elemento $s' \in V(s)$ é dito um vizinho de s. Todo vizinho $s' \in V(s)$ é obtido de s por meio de uma operação m, denominada movimento. Esta operação é representada por $s' \leftarrow s \oplus m$.

A Figura 2.2 ilustra a noção de vizinhança. Nela, $V(s) = \{s^1, s^2, s^3, s^4, s^5, s^6\}$ representa a vizinhança da solução $s \in S$. Os movimentos que levam s a cada um dos seus vizinhos são representados pelas setas.

O movimento de troca da ordem de processamento de duas tarefas é, por exemplo, um movimento que define uma possível vizinhança para o PSUMAA. Segundo este movimento, a vizinhança de uma solução $s \in S$ é o conjunto das soluções $s' \in S$ obtidas de s por meio

[1]Caso o problema seja de maximização, ele pode ser facilmente transformado em um problema equivalente de minimização.

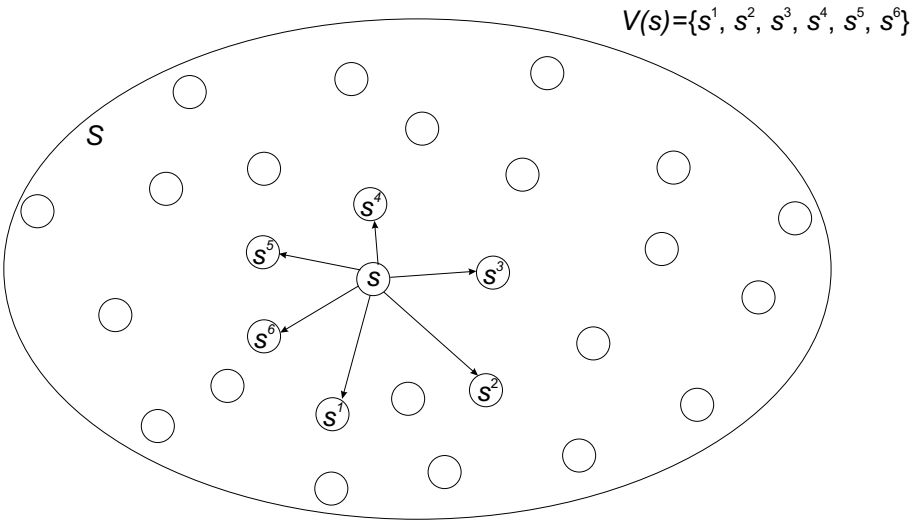

Figura 2.2: Exemplo de vizinhança de uma solução.

da troca da ordem de processamento entre duas tarefas.

Dada uma vizinhança, as técnicas de BL clássicas consistem em caminhar iterativamente de vizinho para vizinho, tentando melhorar a solução construída. Claramente, uma BL é sensível à vizinhança adotada. Deste modo, a escolha de uma vizinhança que permite chegar em qualquer solução do espaço de busca, a partir de qualquer outra solução deste espaço e em um número finito de iterações, é de fundamental importância para o sucesso da técnica.

Outros conceitos importantes surgem com a noção de vizinhança.

Sejam f, V e S como definidos anteriormente. Dizemos que uma solução $s \in S$ é um mínimo local com relação à vizinhança V se $f(s) \leq f(s')$ para todo $s' \in V(s)$. Se uma solução $s^\star \in S$ é tal que $f(s^\star) \leq f(s')$, para todo $s' \in S$, $s^\star$ é dito um mínimo global.

As heurísticas de refinamento clássicas são: método da descida, método de primeira melhora e método de descida randômica.

Dadas uma vizinhança e uma solução inicial, iterativamente, o método da descida analisa todos os vizinhos da solução corrente e o melhor vizinho é selecionado. Se o melhor vizinho representar uma solução melhor que a corrente, ele passa a ser a solução corrente. Caso contrário, o método pára. Portanto, a solução retornada é um ótimo local em relação a vizinhança adotada.

O método da descida é descrito no Algoritmo 1, no qual f é a função a ser minimizada, V é uma vizinhança e s uma solução inicial.

Como o método da descida analisa toda a vizinhança em cada iteração, ele demanda muito tempo computacional para resolver problemas de maior dimensão. Técnicas alternativas tornam-se necessárias para estes casos.

O método de primeira melhora (*first improvement method*) é uma variante do método da descida e evita a busca exaustiva pelo melhor vizinho. Neste método, a exploração da vizinhança é interrompida quando um vizinho melhor é encontrado. Quando isto acontece, a solução corrente é atualizada e a exploração da nova vizinhança é inicializada. Deste modo, toda a vizinhança é explorada apenas no pior caso, ou seja, quando somente o último vizinho é de melhora ou todos os vizinhos não são de melhora. O método é inter-

```
   procedimento Descida(f(.), V(.), s)
1  V ← {s' ∈ V(s) | f(s') < f(s)};
2  enquanto (|V| > 0) faça
3      Selecione s' ∈ V, onde s' = arg min{f(s) | s ∈ V};
4      s ← s';
5      V ← {s' ∈ V(s) | f(s') < f(s)};
6  fim-enquanto;
7  Retorne s;
   fim Descida;
```

Algoritmo 1: Método da descida.

rompido quando um mínimo local é encontrado. É desejável que a ordem de exploração das soluções vizinhas seja alterada a cada passo. Do contrário, privilegia-se apenas um caminho determinístico no espaço de soluções (Souza, 2008).

O Algoritmo 2 representa o método primeiro de melhora. A função a ser minimizada f, a vizinhança V e a solução inicial s são os dados de entrada do método.

```
   procedimento PrimeiraMelhora(f(.), V(.), s)
1  continua ← nao ; /* Verifica se a busca na vizinhança corrente pode ser
   interrompida */
2  enquanto (continua = nao) faça
3      continua ← sim;
4      para (s' ∈ V(s)) faça
5          se (f(s') < f(s)) então
6              s ← s';
7              continua ← nao;
8              Sai do para; /* A exploração da vizinhança é interrompida para
               explorar a vizinhança da nova solução corrente */
9          fim-se;
10     fim-para;
11 fim-enquanto;
12 Retorne s;
   fim PrimeiraMelhora;
```

Algoritmo 2: Método primeira melhora.

Uma outra alternativa para se evitar a busca exaustiva de toda a vizinhança é o método de descida randômica. Este método também é uma variante do método da descida. Iterativamente, um vizinho é escolhido de forma aleatória e analisado. Se este vizinho for de melhora, ele passa a ser a solução corrente. Caso contrário, a solução corrente permanece a mesma e passa-se para a próxima iteração. O algoritmo é interrompido após um número pré-determinado de iterações sem melhora na solução corrente.

A solução retornada pelo método randômico de descida não é necessariamente um ótimo local, pois nesta técnica nem todos os vizinhos da solução corrente são explorados.

O refinamento de uma solução s por meio do método randômico de descida, num

problema de minimização de uma função f, é representado pelo Algoritmo 3. A vizinhança V e o número máximo de iterações sem melhora $IterMax$ também são dados de entrada do método.

```
   procedimento DescidaRandomica(f(.), V(.), IterMax, s)
1  Iter ← 0 ;                          /* Contador de iterações sem melhora */
2  enquanto (Iter < IterMax) faça
3      Iter ← Iter + 1;
4      Selecione aleatoriamente s' ∈ V(s);
5      se (f(s') < f(s)) então
6          Iter ← 0 ;
7          s ← s' ;
8      fim-se;
9  fim-enquanto;
10 Retorne s;
   fim DescidaRandomica;
```

Algoritmo 3: Método descida randômica.

2.2.3 Metaheurísticas

Como visto na Seção 2.2.2, as heurísticas de refinamento clássicas são fortemente dependentes da solução inicial e sempre convergem para o primeiro ótimo local em relação à vizinhança adotada. Isto é um grande problema, pois um ótimo local em relação a uma determinada vizinhança pode não ser o ótimo global do problema.

A Figura 2.3 ilustra o comportamento das heurísticas de refinamento para o problema de minimização de uma função f. Observe que os ótimos locais encontrados a partir de s_1 e s_3 não correspondem ao ótimo global, ao contrário do ótimo local relativo à s_2.

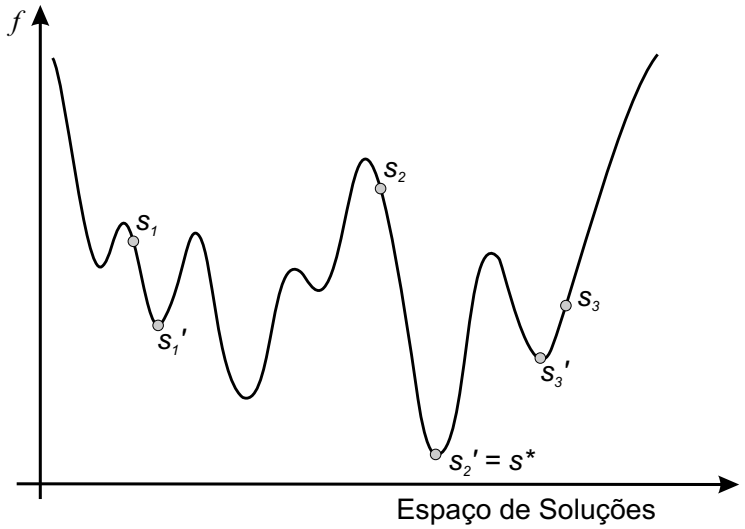

Figura 2.3: Comportamento das heurísticas de refinamento.

Como o objetivo é encontrar soluções o mais próximo possível da ótima e em tempo reduzido, torna-se necessária a utilização de algoritmos dotados de certa "inteligência", que sejam capazes de encontrar ótimos locais, mas escapar destes em busca de um ótimo global. Tais algoritmos são comumente chamados de metaheurísticas.

De acordo com Weise (2008), uma metaheurística é um método heurístico para resolver uma classe mais geral de problemas. Ela combina funções objetivo ou heurísticas de um modo abstrato e esperançosamente eficiente, normalmente sem utilizar conhecimentos mais profundos do problema, isto é, tratando-o como uma "*caixa-preta*".

Para Ribeiro (1996), as metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico.

Portanto, neste trabalho considera-se que uma metaheurística é um algoritmo de caráter geral, ao contrário das heurísticas convencionais, com capacidade de escapar de mínimos locais e encontrar o mínimo global de um problema.

Uma metaheurística se diferencia de outra basicamente pela técnica utilizada para fugir de um ótimo local. A forma com que exploram o espaço de busca divide as metaheurísticas em duas categorias distintas: uma baseada em busca local e a outra baseada em busca populacional.

As metaheurísticas fundamentadas em busca local exploram o espaço de busca elemento a elemento, utilizando a noção de vizinhança, a qual varia de acordo com problema abordado (ver Seção 2.2.2). Já os métodos baseados em busca populacional partem de um conjunto de soluções e aplicam, nestas soluções, operadores que visam a melhoria de todo o conjunto.

A seguir serão apresentadas algumas metaheurísticas clássicas.

Descida em Vizinhança Variável

Proposto por Mladenović & Hansen (1997), a Descida em Vizinhança Variável (*Variable Neighborhood Descent*, VND) é um método de busca local que consiste em explorar o espaço de busca por meio de trocas sistemáticas de vizinhanças.

Conforme esses autores, o VND é baseado nos três princípios básicos apresentados a seguir, sendo que o último deles é de natureza empírica e sugere que um ótimo local fornece informações importantes sobre o ótimo global.

- Um ótimo local em relação à uma dada vizinhança pode não ser um ótimo local relativo à outra vizinhança;
- Um ótimo global é um ótimo local em relação a todas as possíveis vizinhanças;
- Para muitos problemas, ótimos locais com relação a uma ou mais vizinhanças são relativamente próximos.

O método VND clássico recebe como entrada uma solução inicial e um conjunto finito e ordenado de vizinhanças. A cada iteração, uma vizinhança é explorada até que se encontre um ótimo local em relação à ela. Se este ótimo local representar uma melhora na solução corrente, ele é aceito e retorna-se à primeira vizinhança. Caso contrário, passa-se a explorar a vizinhança seguinte. O método pára quando um ótimo local em relação a todas as vizinhanças é encontrado.

O Algoritmo 4 representa o método VND básico. Nele é considerado o problema de minimização de uma função f, dados um conjunto $V = \{V^{(1)}, V^{(2)}, \ldots, V^{(r)}\}$, composto de r vizinhanças diferentes, e uma solução inicial s.

```
   procedimento VND(f(.), V(.), r, s)
1  k ← 1 ;                                          /* Vizinhança corrente */
2  enquanto (k ≤ r) faça
3      Encontre o melhor vizinho s' ∈ V^(k)(s);
4      se (f(s') < f(s)) então
5          s ← s';
6          k ← 1
7      senão
8          k ← k + 1
9      fim-se
10 fim-enquanto;
11 Retorne s;
   fim VND;
```

Algoritmo 4: Descida em Vizinhança Variável.

Segundo Souza (2008), dependendo do problema abordado, a busca pelo ótimo local (linha 3 do Algoritmo 4) pode ser custosa computacionalmente. Nesta situação, é comum fazer a busca pela primeira solução de melhora (ver Seção 2.2.2). Outra alternativa é considerar a exploração apenas em um certo percentual da vizinhança, sendo este percentual um parâmetro do método. Uma terceira alternativa, bastante utilizada nestas situações, é aplicar o método randômico de descida (também descrito na Seção 2.2.2) em cada vizinhança explorada.

Mais informações sobre o VND podem ser encontradas em Mladenović & Hansen (1997) e Hansen & Mladenović (2003), dentre outros trabalhos.

GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure* - Procedimento de busca adaptativa gulosa e aleatória) é um método iterativo, proposto por Feo & Resende (1995), constituído basicamente de duas fases: uma fase de construção e outra de busca local. Na fase de construção, uma solução é gerada, elemento a elemento, de forma parcialmente gulosa. Já a fase de busca local consiste em refinar a solução construída na primeira fase. Este procedimento é repetido *GRASPmax* vezes e a melhor solução encontrada é retornada como resultado.

O Algoritmo 5 descreve o procedimento básico do GRASP aplicado ao problema de minimização de uma função f. A função adaptativa gulosa g, a vizinhança V, *GRASPmax* e γ são os parâmetros do método.

Como mencionado na Seção 2.2.2, os procedimentos gulosos são determinísticos e, assim, retornam sempre a mesma solução para um determinado problema. Além disso, a solução obtida geralmente converge rapidamente para um ótimo local. Por outro lado, as construções aleatórias geram soluções diversas e, em geral, de baixa qualidade; ou seja,

```
   procedimento GRASP(f(.), g(.), V(.), GRASPmax, γ)
1  f* ← ∞ ;                                   /* Menor valor de f obtido */
2  para (Iter = 1, 2, ... , GRASPmax) faça
3      s ← ConstrucaoGRASP(g(.), γ);
4      BuscaLocal(f(.), V(.), s);
5      se (f(s) < f*) então
6          s* ← s;
7          f* ← f(s);
8      fim-se
9  fim-para;
10 Retorne s*;
   fim GRASP;
```

Algoritmo 5: GRASP.

uma construção aleatória gera uma solução diferente em cada execução e uma busca local nestas soluções normalmente demoram para encontrar um ótimo local.

A fase de construção do GRASP visa conciliar as qualidades de uma construção gulosa com as qualidades de uma construção aleatória, isto é, ser capaz de gerar soluções diferentes e que convergem rapidamente para um ótimo local. A cada iteração desta fase, os elementos que ainda estão fora da solução são ordenados por meio de uma função que estima o benefício associado a inclusão de cada elemento na solução parcial. Os melhores elementos, segundo esta função, são inseridos em uma lista restrita de candidatos (LRC). Desta lista, aleatoriamente escolhe-se o elemento que será incrementado à solução parcial naquela iteração.

O tamanho da LRC e o nível de gulosidade e de aleatoriedade da fase de construção do GRASP são determinados por um parâmetro $\gamma \in [0, 1]$. É interessante observar que $\gamma = 0$ implica em uma solução completamente gulosa e $\gamma = 1$ implica em uma solução totalmente aleatória.

O Algoritmo 6 descreve a fase de construção de um algoritmo GRASP. A função adaptativa gulosa g é quem estima o benefício da seleção de cada um dos elementos e o parâmetro γ determina a LRC em cada iteração.

Na fase de busca local do GRASP pode-se, por exemplo, utilizar uma das técnicas apresentadas na Seção 2.2.2.

Pode-se notar que um ótimo local é encontrado em cada iteração do GRASP. Como os ótimos locais são encontrados a partir de soluções iniciais distintas, eles podem ser diferentes. O melhor ótimo local encontrado é retornado pelo método como a solução para o problema.

Feo & Resende (1995), Resende & Ribeiro (2003) e Souza (2008) apresentam mais informações sobre o GRASP.

Busca Tabu

Proposta por Glover (1986), a Busca Tabu (*Tabu Search*, TS) é uma metaheurística baseada em busca local.

procedimento *ConstrucaoGRASP*$(g(.), \gamma)$
1 $s \leftarrow \emptyset$;
2 Inicialize o conjunto C de candidatos;
3 enquanto $(C \neq \emptyset)$ faça
4 $g(t_{min}) \leftarrow \min\{g(t) \mid t \in C\}$;
5 $g(t_{max}) \leftarrow \max\{g(t) \mid t \in C\}$;
6 $LRC \leftarrow \{t \in C \mid g(t) \leq g(t_{min}) + \gamma(g(t_{max}) - g(t_{min}))\}$;
7 Aleatoriamente, selecione um elemento $t \in LRC$ com probabilidade uniforme;
8 $s \leftarrow s \cup \{t\}$;
9 Atualize o conjunto C de candidatos;
10 fim-enquanto;
11 Retorne s;
fim *ConstrucaoGRASP*;

Algoritmo 6: Fase de construção de um algoritmo GRASP.

Dada uma vizinhança e uma solução inicial, a TS consiste basicamente em, iterativamente, explorar a vizinhança atual e mudar da solução corrente para o melhor vizinho encontrado, mesmo que o melhor vizinho não represente uma solução de melhora. Isto é feito até que um critério de parada seja satisfeito.

A estratégia de aceitar soluções de piora é utilizada para escapar das armadilhas dos ótimos locais; porém, esta tática pode fazer com que o algoritmo caia em ciclos, isto é, o algoritmo pode voltar a uma solução gerada anteriormente. Por exemplo, quando a solução corrente é um ótimo local, seu melhor vizinho não será de melhora e será aceito. Na iteração seguinte, o ótimo local poderá ser novamente gerado por ser o melhor vizinho da solução corrente e o algoritmo ficará cíclico.

Para evitar ciclagem, o ideal seria armazenar todas as soluções geradas pelo método em uma lista, denominada lista tabu. No entanto, isto é computacionalmente inviável. Uma alternativa é armazenar apenas as últimas $|LT|$ soluções geradas. Mas uma lista deste tamanho evitaria ciclos de até $|LT|$ iterações, além de poder ser inviável armazenar $|LT|$ soluções e testar se uma solução está ou não na lista tabu.

Na prática, o mais comum é criar uma lista tabu de movimentos proibidos. A lista tabu clássica contém os movimentos reversos aos últimos $|LT|$ movimentos realizados (onde $|LT|$ é um parâmetro do método) e funciona como uma fila FIFO (*First-In-First-Out*).

Apesar de reduzir o risco de ciclagem, uma lista tabu de movimentos pode ser muito restritiva (impede o retorno a uma solução gerada anteriormente e também que novas soluções sejam geradas). Para tentar corrigir este problema, usa-se um critério de aspiração, que retira, sob certas condições, o *status* tabu de um movimento. Um exemplo de critério de aspiração é aceitar um movimento, mesmo que tabu, se ele melhorar o valor global da função objetivo. Esse é o chamado critério de aspiração por objetivo, sendo o mais usado em implementações Busca Tabu. Outra possibilidade é realizar o movimento tabu mais antigo se todos os possíveis movimentos forem tabus (aspiração por *default*). Uma relação completa de outros critérios de aspiração usualmente utilizados em Busca Tabu é encontrada em Glover & Laguna (1997).

Geralmente, o método é interrompido após $BTmax$ iterações sem melhora na melhor solução encontrada, sendo $BTmax$ um parâmetro do método.

O Algoritmo 7 descreve a metaheurística TS básica aplicada ao problema de minimização de uma função f. A vizinhança V, a função de aspiração A, o conjunto de soluções vizinhas testadas em cada iteração (SVT), o tamanho da lista tabu $|LT|$, o critério de construção e atualização da lista tabu, o número máximo de iterações sem melhora $BTmax$ e a solução inicial s são as entradas do método.

```
   procedimento BuscaTabu(f(.), V(.), A(.), SVT, |LT|, BTmax, s)
 1 s* ← s ;                                          /* Melhor solução obtida */
 2 Iter ← 0 ;       /* Contador do número de iterações sem melhora de s* */
 3 LT ← ∅ ;                                                  /* Lista tabu */
 4 Inicialize a função de aspiração A;
 5 enquanto (Iter ≤ BTmax) faça
 6     Iter ← Iter + 1;
 7     Seja s' ← s ⊕ m o melhor elemento de SVT ⊂ V(s) tal que o movimento m não
       é tabu (m ∉ LT) ou a condição de aspiração de s' (A(s')) é atendida;
 8     Atualize a LT;
 9     s ← s';
10     se (f(s) < f(s*)) então
11         s* ← s;
12         Iter ← 0;
13     fim-se
14     Atualize A;
15 fim-enquanto;
16 Retorne s*;
   fim BuscaTabu;
```

Algoritmo 7: Busca Tabu.

Mais informações a respeito da TS podem ser encontradas em Glover (1986), Glover (1996), Glover & Laguna (1997), Gendreau (2003) e Souza (2008), entre outros trabalhos.

Simulated Annealing

O *Simulated Annealing* (SA) é uma técnica de busca local probabilística proposta por Kirkpatrick *et al.* (1983) e fundamentada na termodinâmica, ao simular o processo de recozimento (nome dado à operação de resfriamento de um conjunto de átomos aquecidos).

Para se obter metais mais estáveis, estruturalmente fortes e de menor energia interna, eles são aquecidos a uma temperatura suficientemente alta e depois são lentamente resfriados. Durante o recozimento, o material passa por vários estados possíveis.

Como o SA simula o processo de recozimento dos metais, são feitas as seguintes correspondências com um problema de otimização. As soluções do espaço de busca correspondem aos possíveis estados de um metal. O valor da função objetivo corresponde à energia interna em cada estado. O valor de um ótimo local (possivelmente global) corresponde à energia mínima.

O método parte de uma solução inicial qualquer e a cada iteração é gerado, aleatoriamente, um único vizinho s' da solução corrente s. Seja $\Delta = f(s') - f(s)$ e considere que o problema seja de minimização. Se $\Delta < 0$ então a solução vizinha passa a ser a solução

corrente. Caso contrário ($\Delta \geq 0$), o vizinho pode ser aceito com uma probabilidade $e^{\frac{-\Delta}{kT}}$, onde k é a constante de Boltzmann e T representa a temperatura corrente.

Inicialmente, a temperatura T assume um valor suficientemente alto T_0. A cada iteração i do método, a temperatura é diminuída, ou seja, $T_i \leftarrow \alpha \cdot T_{i-1}$, onde $0 < \alpha < 1$. Os valores de T_0 e α são parâmetros do método.

Observe que quanto maior a temperatura, maior a probabilidade de se aceitar uma solução de piora. Portanto, a cada iteração do algoritmo a chance de uma solução de piora ser aceita é reduzida. As soluções de piora são aceitas para que se escape das armadilhas dos ótimos locais.

O método é interrompido quando a temperatura atinge um valor próximo de zero e somente vizinhos de melhora são aceitos, evidenciando o encontro de um ótimo local.

O Algoritmo 8 descreve o método SA básico para o problema de minimização de uma função f. A estrutura de vizinhança V, a razão de resfriamento α, o número de iterações em cada temperatura ($SAmax$), a temperatura inicial T_0, um número real próximo de zero (ϵ) e uma solução inicial s são as entradas do método.

```
   procedimento SA(f(.), V(.), α, SAmax, T0, ε, s)
 1 s* ← s ;                                          /* Melhor solução obtida */
 2 IterT ← 0 ;                    /* Número de iterações na temperatura T */
 3 T ← T0 ;                                          /* Temperatura corrente */
 4 enquanto (T > ε) faça
 5     enquanto (IterT < SAmax) faça
 6         IterT ← IterT + 1;
 7         Aleatoriamente, escolha s' ∈ V(s);
 8         Δ ← f(s') − f(s);
 9         se (Δ < 0) então
10             s ← s';
11             se (f(s') < f(s*)) então s* ← s';
12         senão
13             Aleatoriamente, escolha x ∈ [0, 1];
14             se (x < e^(−Δ/kT)) então s ← s';
15         fim-se
16     fim-enquanto;
17     T ← α · T;
18     IterT ← 0
19 fim-enquanto;
20 Retorne s*;
   fim SA;
```

Algoritmo 8: *Simulated Annealing.*

Mais detalhes do método podem ser encontrados em Kirkpatrick *et al.* (1983), Ribeiro (1996), Henderson *et al.* (2003) e Souza (2008), entre outros trabalhos.

Iterated Local Search

O *Iterated Local Search* (ILS) é um método de busca local que faz uso de perturbações para escapar das armadilhas dos ótimos locais e, possivelmente, encontrar um ótimo global. Tais pertubações devem ser fortes o suficiente para permitir a fuga do ótimo local corrente e que a busca local explore diferentes regiões do espaço de busca. Ao mesmo tempo, elas dever ser suficientemente fracas para preservar características do ótimo local corrente e evitar reinícios aleatórios.

De acordo com Souza (2008), é necessário especificar quatro componentes para a aplicação do ILS:

1. Procedimento *GeraSolucaoInicial*, que gera uma solução inicial. Pode ser utilizado, por exemplo, uma HC (ver Seção 2.2.2);
2. Procedimento *BuscaLocal*, que retorna uma solução possivelmente melhorada. Por exemplo, pode ser utilizada uma heurística de refinamento (ver seção 2.2.2);
3. Procedimento *Perturbacao*, que modifica a solução corrente, guiando-a a uma solução intermediária;
4. Procedimento *CriterioAceitacao*, que decide em qual solução a próxima perturbação será aplicada.

Um critério de aceitação muito utilizado consiste em aceitar um ótimo local s" somente se ele for melhor que o ótimo local corrente s. O Algoritmo 9 ilustra a aplicação do ILS básico a um problema de minimização de uma função f.

```
   procedimento ILS(f(.))
 1 s_0 ← GeraSolucaoInicial();
 2 s ← BuscaLocal(s_0);
 3 s* ← s;                                   /* Melhor solução obtida */
 4 enquanto (critério de parada não é satisfeito) faça
 5     s' ← Perturbacao(s);
 6     s" ← BuscaLocal(s');
 7     se (f(s") < f(s*)) então
 8         s* ← s";
 9     fim-se
10     s ← CriterioAceitacao(s, s");
11 fim-enquanto;
12 Retorne s*;
   fim ILS;
```

Algoritmo 9: *Iterated Local Search.*

Geralmente, utiliza-se um número máximo de iterações sem melhora na melhor solução encontrada como critério de parada do algoritmo.

Mais detalhes do método podem ser encontrados em Souza (2008) e Lourenço *et al.* (2003).

Algoritmos Evolutivos

Algoritmos evolutivos (AE's) são metaheurísticas baseadas em busca populacional e que utilizam mecanismos inspirados na biologia, como mutação, *crossover*, seleção natural e condições de sobrevivência, para refinar, iterativamente, um conjunto de soluções candidatas (Weise, 2008).

Essencialmente, um AE parte de um conjunto de soluções candidatas, denominado população inicial, e, em cada iteração, todos os elementos (indivíduos) da população atual são avaliados pela função de avaliação. Caso o critério de parada seja satisfeito, o método é interrompido e os melhores elementos são listados. Caso contrário, com o auxilio de uma função de aptidão, que pode não depender somente da função de avaliação, os indivíduos são novamente avaliados. A função de aptidão tem o objetivo de auxiliar na seleção dos indivíduos da população que darão origem a novos indivíduos. A geração de novos indivíduos pode ser feita por meio de *crossover* ou mutação. No *crossover*, dois indivíduos pais dão origem a filhos que possuem características de ambos os pais. A mutação consiste em alterar uma característica do indivíduo. Terminada a etapa de geração de novos indivíduos, também com o auxilio da função de aptidão, é determinada a população sobrevivente, isto é, os indivíduos que farão parte da população da próxima iteração.

A Figura 2.4 ilustra a estrutura básica de um AE.

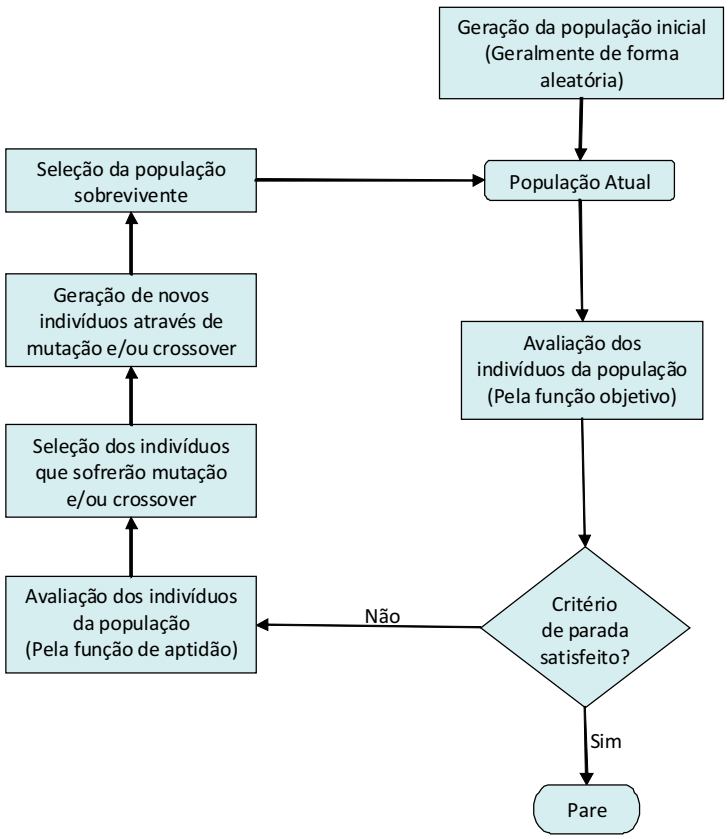

Figura 2.4: Estrutura básica de um Algoritmo Evolutivo.

De acordo com Souza (2008), os critérios de parada mais comuns são: quando um certo

número de gerações é atingido, quando não há melhora na melhor solução encontrada após um certo número de iterações ou quando o desvio padrão da população é inferior a um certo valor (situação que evidencia uma homogeneidade da população).

Mais informações sobre os AE's podem ser encontradas, por exemplo, em Weise (2008).

Algoritmos Genéticos

Segundo Weise (2008), os Algoritmos Genéticos (AG's) compõem uma subclasse dos AE's, na qual os elementos do espaço de busca são vetores de componentes binários ou arranjos de tipos elementares.

É muito comum encontrar na literatura trabalhos que, por conveniência, consideram os espaços de soluções dos AG's formados por vetores de números inteiros. Neste caso, a definição de AG coincide com a definição de AE.

De qualquer forma, pode-se dizer que a Figura 2.4 ilustra a estrutura de um Algoritmo Genético básico.

Reeves (2003), Weise (2008) e Souza (2008) apresentam mais detalhes a respeito dos AG's.

Scatter Search

O *Scatter Search* (SS) é um método baseado em busca populacional que apresenta similaridades com os Algoritmos Genéticos; difere no tamanho da população e no uso de estratégias determinísticas, ao invés de probabilísticas, para atingir os objetivos de diversificação e intensificação (Belfiore *et al.*, 2006).

Ainda conforme Belfiore *et al.* (2006), o SS opera sobre um conjunto de soluções, chamado conjunto de referência, combinando estas soluções para criar novas soluções de modo a melhorar as soluções originais. É, portanto, um método evolutivo. O tamanho do conjunto de referência em *scatter search* é pequeno comparado à população nos Algoritmos Genéticos.

Segundo (Glover & Laguna, 2003), o processo do SS é organizado para:

1. Capturar informações não contidas separadamente nas soluções originais;

2. Tirar proveito de métodos de soluções heurísticas auxiliares (para avaliar as combinações produzidas e gerar ativamente novas soluções);

3. Utilizar estratégias específicas no lugar de métodos aleatórios de busca.

Uma abordagem mais detalhada do método é apresentada em Glover & Laguna (2003) e Belfiore *et al.* (2006).

2.2.4 Estratégias de Intensificação

As técnicas de intensificação são métodos que exploram com maior intensidade as soluções próximas de uma dada solução. A seguir serão apresentadas duas destas técnicas.

Princípio da Otimalidade Próxima

O Princípio da Otimalidade Próxima (POP) é uma técnica que refina uma parte da solução, por meio de uma busca local, com o objetivo de melhorar toda a solução. Este método é baseado na idéia de que boas soluções em um nível estão próximas de boas soluções em um nível adjacente (Glover & Laguna, 1997).

Na prática, o POP é muito utilizado em métodos construtivos. Periodicamente, antes de se inserir um novo elemento na solução em construção, esta é submetida à uma busca local.

Reconexão por Caminhos

A Reconexão por Caminhos (*Path Relinking*, PR) é uma técnica de intensificação proposta em Glover (1996). O método parte de duas soluções dadas, sendo uma chamada de solução base e a outra de solução guia. Iterativamente, um atributo da solução guia é adicionado na solução base. A solução base sofre, então, uma busca local em que se fixam todos os atributos da solução guia que já foram incrementados à solução base. O método pára quando a solução base possui todos os atributos da solução guia.

O objetivo do método é, portanto, encontrar, durante as iterações, uma solução que possua os atributos bons contidos nas duas soluções dadas, isto é, uma solução melhor que as duas soluções iniciais.

O PR pode ser utilizado como um método de pós-otimização ou então para intensificar a busca em torno de ótimos locais. No primeiro caso, as melhores soluções distintas encontradas por algum procedimento heurístico são armazenadas em um conjunto e posteriormente aplica-se a PR em pares de soluções deste conjunto. Tal conjunto é chamado de conjunto elite e seu tamanho é um parâmetro do método. No segundo caso, sempre que um ótimo local é encontrado por um determinado procedimento, aplica-se a PR no par de soluções formado pelo ótimo local e a melhor solução até então encontrada pelo algoritmo.

Mais detalhes desta técnica podem ser encontrados em Glover (1996), Glover & Laguna (2003) e Souza (2008).

2.3 Trabalhos relacionados

Muitos trabalhos são encontrados na literatura com o objetivo de resolver o PSUMAA em suas diversas variantes. Uma breve descrição dos trabalhos mais relevantes é apresentada a seguir. Por questões didáticas, os trabalhos foram separados de acordo com as datas de entrega. Primeiramente são apresentados os trabalhos que tratam do PSUMAA com datas de entrega comum, em seguida os com datas de entrega distintas e, por último, os com janelas de entrega distintas.

2.3.1 PSUMAA com Datas de Entrega Comuns

Esta seção é destinada à apresentação de alguns trabalhos que tratam do PSUMAA com datas de entrega comuns.

Dentre as versões do PSUMAA, a mais estudada é a com datas de entrega comuns e tempos de *setup* independentes da sequência de produção. Para este caso, existem propriedades que podem ser utilizadas nos algoritmos de resolução, de modo a reduzir o espaço de busca e consequentemente o tempo computacional. As principais propriedades que são utilizadas por todos os autores são:

Propriedade 1 *Em uma sequência ótima não há tempo ocioso entre duas tarefas consecutivas.*

Propriedade 2 *Em uma sequência ótima a conclusão de uma tarefa coincide com a data de entrega ou o processamento da primeira tarefa inicia no instante zero.*

Propriedade 3 *Em uma sequência ótima, as tarefas antecipadas estão ordenadas de forma não-crescente pela razão entre o tempo de processamento e a penalidade por unidade de tempo de antecipação, enquanto as tarefas atrasadas estão ordenadas de forma não-decrescente pela razão entre o tempo de processamento e a penalidade por unidade de tempo de atraso.*

Se uma sequência satisfaz a Propriedade 3, ela é dita *V-shaped* (Biskup & Feldmann, 2001).

As justificativas para estas propriedades podem ser encontradas em Szwarc (1996). Ainda em Szwarc (1996), foram estudadas, também, propriedades da otimalidade no caso em que as penalidades são dependentes das tarefas e as razões entre os tempos de processamento e as penalidades satisfazem certas condições. As propriedades estudadas foram utilizadas em três algoritmos baseados em programação dinâmica e de complexidade pseudo-polinomial, sendo que estes algoritmos refletiam o estado da arte até então. Experimentos em problemas-teste com até 100 tarefas evidenciaram a melhoria proporcionada pela utilização das propriedades apresentadas.

Lee & Kim (1995) resolveram o problema por meio de um algoritmo genético paralelo com representação binária, sendo este uma variante do AG convencional. Uma condição necessária para que um arranjo em série de genes seja *V-shaped* dentro de uma sequência é apresentada. Foram resolvidos problemas-teste com até 100 tarefas.

Alidaee & Dragan (1997) apresentaram um algoritmo de complexidade $O(n \log n)$ que resolve de forma ótima os casos em que as penalidades por antecipação e atraso são iguais para a mesma tarefa e proporcionais aos tempos de processamento das tarefas.

Biskup & Feldmann (2001) propuseram um gerador de problemas-teste. Foram utilizadas duas novas heurísticas para resolver e obter limites superiores para os problemas-teste, com até 1000 tarefas. As heurísticas se mostraram eficientes, pois encontraram resultados próximos dos ótimos conhecidos de problemas de menor porte. O gerador, junto com os limites superiores, são disponibilizados pelos autores para servirem como referências de trabalhos futuros sobre o PSUMAA.

Feldmann & Biskup (2003) apresentam uma nova e apropriada representação para o problema, baseada nas propriedades deste caso do PSUMAA. Foram utilizadas três metaheurísticas para resolver o problema: um Algoritmo Evolutivo, um *Simulated Annealing* e um *Threshold Accepting*, sendo este último uma variante do SA. Estas metaheurísticas foram aplicadas em problemas-teste propostos por Biskup & Feldmann (2001).

Uma construção gulosa e heurísticas baseadas em Busca Tabu e Algoritmos Genéticos foram propostas por Hino *et al.* (2005). Estratégias híbridas também foram analisadas com o objetivo de melhorar o desempenho destes métodos. Foram resolvidos problemas-teste com até 1000 tarefas.

Belfiore *et al.* (2006) aplicaram a metaheurística *Scatter Search* nesta variante do PSUMAA. O algoritmo proposto foi utilizado para resolver todos os problemas-teste propostos por Biskup & Feldmann (2001).

Ying (2008) resolveu o problema por meio de um procedimento *Branch-and-Bound* que faz uso de um algoritmo *Recovering Beam Search* para caminhar na árvore de busca. O autor também utilizou a base de dados proposta por Biskup & Feldmann (2001) para realizar experimentos computacionais.

O PSUMAA com datas de entrega comum pode ser classificado em restrito ou irrestrito. Quando a soma dos tempos de processamento de todas as tarefas é maior que a data de entrega, diz-se que o problema é restrito. Caso contrário, o problema é dito irrestrito.

James (1997) testou diferentes formas de Busca Tabu para resolver o caso restrito do problema, inclusive uma que aplica as propriedades apresentadas anteriormente em uma nova técnica que reduz o tamanho do espaço de busca de $n!$ para 2^n. Foram resolvidos problemas-teste com até 250 tarefas e a TS que faz uso da nova técnica mostrou-se mais eficiente que as demais.

Mondal & Sen (2001) consideraram que as penalidades por antecipação e atraso são independentes das tarefas. Foi apresentado um algoritmo com espaço de busca em grafo para encontrar a solução ótima de problemas-teste maiores do caso restrito. Ao contrário da programação dinâmica, o algoritmo encontrou soluções ótimas mesmo quando a memória era limitada. O algoritmo se mostrou mais rápido que a programação dinâmica e *depth-first-branch-bound* (*depth-first* é uma heurística utilizada para caminhar na árvore de busca do método BB) e consegue resolver muitos problemas-teste maiores em tempo razoável. Também foram propostos e utilizados novos limites inferiores e superiores para o problema. Os problemas-teste resolvidos são com até 2000 tarefas.

Rabadi *et al.* (2004) consideraram o PSUMAA com datas de entrega comuns, penalidades iguais (o objetivo é minimizar a soma simples das antecipações e dos atrasos) e tempos de *setup* dependentes da sequência de produção. Foi proposto um algoritmo *Branch-and-Bound* para obter soluções ótimas do caso irrestrito. Os autores resolveram problemas-teste com até 25 tarefas em tempo computacional aceitável, o que representou um avanço para a época, já que os algoritmos exatos de então resolviam, na otimalidade, apenas problemas desta classe com até 8 tarefas.

2.3.2 PSUMAA com Datas de Entrega Distintas

Esta seção é destinada à apresentação de alguns trabalhos que tratam do PSUMAA com datas de entrega distintas.

M'Hallah (2007) considerou o problema com as penalidades iguais (o objetivo é minimizar a soma simples das antecipações e dos atrasos) e os tempos de *setup* independentes da sequência de produção, não sendo permitida a existência de tempo ocioso entre a execução de tarefas consecutivas. Foi proposto um método heurístico híbrido que combina heurísticas de busca local (regras de despacho, métodos de descida e SA) e um algoritmo evolutivo baseado em AG. Os efeitos de cada heurística no método são mostrados via

experimentos com problemas-teste com até 4000 tarefas.

Chang (1999) tratou do problema com penalidades iguais e tempos de *setup* independentes da sequência de produção, sendo permitida a ociosidade de máquina. O autor utilizou um algoritmo *branch-and-bound* para resolver o problema. Um esquema para calcular diferentes limites inferiores baseados no procedimento de eliminação de sobreposição de uma sequência *Just-In-Time* também foi apresentado. Propriedades e teoremas deste procedimento de eliminação foram demonstradas. O algoritmo foi utilizado para resolver problemas-teste com até 45 tarefas. Ventura & Radhakrishnan (2003) abordaram o mesmo problema, para o qual também propuseram um algoritmo BB. Neste trabalho foram derivadas propriedades de uma sequência ótima. Um procedimento de relaxação lagrangeana que utiliza o algoritmo de subgradiente foi implementado para atingir soluções próximas da otimidade e ajustar os limites inferiores. O algoritmo foi testado em problemas-teste com até 100 tarefas e apresentou uma divergência relativa menor que 3% das soluções ótimas.

Li (1997) estudou o caso com tempos de *setup* independentes da sequência de produção, não sendo permitida a existência de tempos ociosos entre a execução de tarefas consecutivas. Neste trabalho, o problema foi decomposto em dois subproblemas com estruturas mais simples. O limite inferior do problema foi, então, dado pela soma dos limites inferiores dos dois subproblemas. O limite inferior de cada subproblema foi obtido por relaxação lagrangeana. Também foram desenvolvidos dois procedimentos de ajuste multiplicador, de complexidade $O(n \log n)$, para resolver os duais dos subproblemas. Um algoritmo *branch-and-bound* baseado nestes procedimentos foi apresentado e utilizado para resolver problemas-teste com até 50 tarefas, dobrando a dimensão dos problemas resolvidos, na otimidade, por algoritmos exatos até aquela data. Li (1997) propôs, também, uma heurística baseada em busca local e resolveu problemas com até 3000 tarefas. Almeida & Centeno (1998) aplicaram um método heurístico composto, que combina TS, SA e técnicas de descida, em problemas-teste com até 50 tarefas. Liaw (1999), Valente & Alves (2005a) e Valente & Alves (2005b) também trataram este mesmo problema. O primeiro autor utilizou um algoritmo BB baseado em procedimentos que determinam limites inferiores e superiores eficientes. Foram derivadas regras de dominância para ajudar a eliminar nós não promissores da árvore de busca. O desempenho do algoritmo foi testado em problemas-teste com até 50 tarefas. No segundo trabalho foram introduzidas duas novas heurísticas, sendo uma regra de despacho e um procedimento guloso. Também foram utilizadas regras de dominância, visto que elas melhoram os resultados das heurísticas e demandam pouco tempo adicional. Os experimentos computacionais foram realizados em problemas-teste com até 500 tarefas. O terceiro trabalho propõe dois algoritmos BB. Cada BB proposto se utiliza de uma variante do algoritmo *beam-search* para caminhar na árvore de busca, sendo que um deles se utiliza de um *Filtered Beam Search* e o outro de um *Recovering Beam Search*. Os dois métodos foram testados em problemas-teste com até 500 tarefas.

Lee & Choi (1995) trataram do caso com tempos de *setup* independentes da sequência de produção, sendo permitida a ociosidade de máquina. Foi apresentado um algoritmo de complexidade polinomial que determina a data ótima de inicio de uma tarefa em uma dada sequência de produção. Também aplicou-se ao problema um algoritmo genético que faz uso deste algoritmo de data ótima. Foram resolvidos problemas-teste com até 80 tarefas. Mazzini & Armentano (2001) estudaram este mesmo problema. Em tal trabalho, uma

heurística construtiva que determina a sequência de produção e simultaneamente insere tempos ociosos foi proposta e utilizada para resolver problemas-teste com até 80 tarefas.

No trabalho de Bustamante (2007) foi estudado o problema com tempos de *setup* dependentes da sequência de produção, sendo permitida a existência de tempos ociosos de máquina. São desenvolvidos dois modelos de programação linear inteira mista, sendo que tais modelos exigem que os tempos de *setup* satisfaçam a desigualdade triangular. O autor também sugeriu alterações nos modelos que permitem resolver o caso com janelas de entrega distintas, mas os testes computacionais foram realizados apenas em problemas com datas de entrega distintas. Foram resolvidos, na otimalidade, problemas com até 10 tarefas por meio do software de otimização GLPK 4.8.

2.3.3 PSUMAA com Janelas de Entrega Distintas

Alguns trabalhos da literatura que tratam o PSUMAA com janelas de entrega distintas são apresentados a seguir.

Wan & Yen (2002) trataram o problema com tempos de *setup* independentes da sequência de produção, permitindo a existência de tempos ociosos de máquina. Primeiramente, foi apresentada uma formulação matemática do problema junto com várias propriedades importantes para sua resolução. Em seguida, foi proposto um procedimento de complexidade polinomial para determinar a data de conclusão ótima de processamento de cada tarefa em uma dada sequência, sendo este procedimento uma extensão do algoritmo proposto por Lee & Choi (1995). Por fim, uma Busca Tabu, que faz uso do procedimento de datas ótimas, foi proposta para resolver o problema. Foram realizados testes computacionais em problemas envolvendo até 80 tarefas.

Koulamas (1996) focou no problema com janelas de entrega distintas, porém com uma sensível distinção dos demais trabalhos da literatura em relação à antecipação. Em seu trabalho, há antecipação de uma tarefa quando seu processamento é iniciado antes do início de sua janela de entrega. O restante da literatura considera que há antecipação quando uma tarefa é concluída antes do início de tal janela. Koulamas (1996) considerou, também, tempos de *setup* independentes da sequência de produção, sendo permitida a ociosidade de máquina. O autor adaptou heurísticas já utilizadas em outros casos do PSUMAA e as aplicou nessa versão. Também foi proposto um algoritmo que insere tempos ociosos, de modo ótimo, em uma dada sequência de produção. Foram realizados experimentos computacionais em problemas-teste com até 200 tarefas.

Gomes Júnior (2007), Penna (2009) e Ribeiro (2009) correspondem ao estado da arte em relação ao PSUMAA-JE-TPD. No primeiro trabalho foi proposto um modelo de programação matemática para representar o problema, no qual deixa de ser necessário que o problema satisfaça a desigualdade triangular. Também foi proposto um método heurístico baseado em GRASP, ILS e VND. Para cada sequência de tarefas gerada pela heurística proposta, é utilizado um algoritmo de complexidade polinomial que determina a data ótima de início de processamento das tarefas na sequência dada, sendo este algoritmo uma extensão daquele proposto por Wan & Yen (2002). Foram realizados experimentos computacionais em problemas-teste com os tempos de *setup* simétricos e com até 75 tarefas, utilizando a heurística, e até 12 tarefas, utilizando o modelo matemático. O segundo trabalho propõe uma heurística baseada em GRASP, VND, TS e PR, enquanto no terceiro trabalho é proposto um Algoritmo Genético Adaptativo para resolver o problema. Nos

dois últimos trabalhos foram realizados experimentos nos mesmos problemas-teste utilizados por Gomes Júnior (2007), sendo mostrada a superioridade dos algoritmos propostos em relação a este último. Ribeiro (2009) também realizou testes em outras duas bases de dados em que os tempos de *setup* não são necessariamente simétricos. Foram resolvidos problemas-teste com até 100 tarefas por meio do AG proposto.

Capítulo 3

Formulações de Programação Matemática

Como foi mencionado na Seção 2.2.1, os métodos exatos são de grande importância por serem capazes de encontrar a solução ótima de problemas de pequena dimensão e auxiliarem na validação dos procedimentos heurísticos. Com esta finalidade, nesta seção são apresentadas três formulações de programação matemática para representar o PSUMAA-JE-TPD. A primeira delas foi proposta por Gomes Júnior (2007), enquanto as outras duas são propostas neste trabalho. A segunda formulação é um aperfeiçoamento da primeira e a última é indexada no tempo.

3.1 Modelo de Gomes Júnior (2007) - MPLIM-G

A seguir é reproduzido o modelo de programação linear inteira mista (PLIM) proposto por Gomes Júnior (2007) para representar o PSUMAA-JE-TPD. Doravante esse modelo passa a ser denotado por MPLIM-G.

Para auxiliar na modelagem, são utilizadas duas tarefas fictícias, 0 (zero) e $n + 1$, que devem ser sequenciadas necessariamente na primeira e na última posição, respectivamente. Admite-se que P_0 e P_{n+1} são iguais a zero e que $S_{0i} = S_{i0} = 0$ e $S_{i,n+1} = S_{n+1,i} = 0$, $\forall\ i \in I$. A data de início do processamento da tarefa $j \in I \cup \{0, n+1\}$ e os tempos de antecipação e atraso da tarefa $i \in I$ são representados por s_j, e_i e t_i, respectivamente.

Sejam y_{ij} variáveis que determinam a sequência de produção, sendo $y_{ij} = 1$ se a tarefa j for sequenciada imediatamente após a tarefa i e $y_{ij} = 0$, caso contrário, $\forall\ i, j \in I \cup \{0, n+1\}$. Considere, ainda, uma constante M de valor suficientemente grande. O modelo MPLIM-G é representado pelas equações (3.1)-(3.10):

$$\min \quad Z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{3.1}$$

$$\text{s.a.} \quad s_i + P_i + y_{ij}(M + S_{ij}) - M \leq s_j \quad \forall\, i \in I \cup \{0\}, \quad \forall j \in I \cup \{n+1\} \text{ e } i \neq j \tag{3.2}$$

$$\sum_{j=1,\, j\neq i}^{n+1} y_{ij} = 1 \quad \forall\, i \in I \cup \{0\} \tag{3.3}$$

$$\sum_{i=0,\, i\neq j}^{n} y_{ij} = 1 \quad \forall\, j \in I \cup \{n+1\} \tag{3.4}$$

$$s_i + P_i + e_i \geq E_i \quad \forall\, i \in I \tag{3.5}$$

$$s_i + P_i - t_i \leq T_i \quad \forall\, i \in I \tag{3.6}$$

$$s_i \geq 0 \quad \forall\, i \in I \cup \{0, n+1\} \tag{3.7}$$

$$e_i \geq 0 \quad \forall\, i \in I \tag{3.8}$$

$$t_i \geq 0 \quad \forall\, i \in I \tag{3.9}$$

$$y_{ij} \in \{0, 1\} \quad \forall\, i, j \in I \cup \{0, n+1\} \tag{3.10}$$

A função objetivo, representada pela equação (3.1), busca a minimização da soma ponderada das antecipações e atrasos. As restrições (3.2) garantem que existe tempo suficiente para executar uma tarefa i e preparar a máquina antes do início do processamento da tarefa seguinte j, ou seja, elas garantem que o processamento da tarefa j não será iniciado antes do término do processamento da tarefa i. As restrições (3.3) e (3.4) garantem que cada tarefa terá apenas uma tarefa imediatamente sucessora e uma tarefa imediatamente antecessora, respectivamente. As restrições (3.5) e (3.6) definem as antecipações e os atrasos de acordo com as respectivas janelas de entrega de cada tarefa. As restrições (3.7), (3.8), (3.9) e (3.10) dizem respeito ao tipo de variáveis.

3.2 Primeiro Modelo Proposto - MPLIM-BG

O modelo de programação linear inteira mista (PLIM) apresentado a seguir é baseado naquele proposto por Gomes Júnior (2007) (ver Seção 3.1, página 30) e será denotado por MPLIM-BG. Diferentemente de tais autores, é utilizada apenas uma tarefa fictícia, denominada 0 (zero), para auxiliar na modelagem. Esta tarefa é sequenciada duas vezes, sendo uma na primeira posição e outra na última. Considera-se que $P_0 = 0$ e que $S_{0i} = S_{i0} = 0$, $\forall\, i \in I$.

Considerando s_i, e_i, t_i e y_{ij} tais como no MPLIM-G, tem-se que as variáveis $y_{n+1,i}$ e $y_{i,n+1}$, $\forall\, i \in I \cup \{0\}$, e s_{n+1} deixam de ser necessárias no MPLIM-BG. Sendo assim, se comparado ao MPLIM-G, no MPLIM-BG há redução de $2n+3$ variáveis e $n+1$ restrições, com n representando o número de tarefas a serem sequenciadas.

Finalmente, o MPLIM-BG é obtido do MPLIM-G substituindo-se as restrições (3.2), (3.3), (3.4), (3.7) e (3.10) pelas restrições (3.11), (3.12), (3.13), (3.14) e (3.15), respectivamente, apresentadas a seguir:

$$s_i + P_i + y_{ij}(M + S_{ij}) - M \leq s_j \qquad \forall\, i \in I \cup \{0\},\ \forall\, j \in I \ \text{ e } \ i \neq j \tag{3.11}$$

$$\sum_{j=0,\, j\neq i}^{n} y_{ij} = 1 \qquad \forall\, i \in I \cup \{0\} \tag{3.12}$$

$$\sum_{i=0,\, i\neq j}^{n} y_{ij} = 1 \qquad \forall\, j \in I \cup \{0\} \tag{3.13}$$

$$s_i \geq 0 \qquad \forall\, i \in I \cup \{0\} \tag{3.14}$$

$$y_{ij} \in \{0, 1\} \quad \forall\, i, j \in I \cup \{0\} \tag{3.15}$$

3.3 Segundo Modelo Proposto - MPLIM-IT

A seguir é proposto um modelo de programação linear inteira mista (PLIM) indexado no tempo para representar o PSUMAA abordado. Tal modelo, denotado por MPLIM-IT, é baseado no trabalho de de Paula (2008), que utilizou a discretização do tempo para modelar o problema de minimização dos atrasos em sequenciamento de máquinas paralelas com tempos de preparação dependentes da sequência de produção. O MPLIM-IT é descrito como segue.

Seja $H = \{h_0,\ h_1,\ h_2,\ \ldots,\ h_L\}$ o conjunto com as possíveis datas de início de processamento das tarefas de I e considere x_{ih} variáveis binárias, sendo $x_{ih} = 1$ se a tarefa i é programada para iniciar na data h e $x_{ih} = 0$, caso contrário, $\forall\ i \in I$ e $\forall\ h \in H$. Sendo assim, para que o problema tenha solução viável, é necessário que a cardinalidade de H seja maior ou igual ao número de tarefas a serem processadas.

Se e_i e t_i representam, respectivamente, as unidades de tempo de antecipação e tempo de atraso da tarefa $i \in I$, a formulação MPLIM-IT pode ser escrita assim:

$$\min \qquad Z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{3.16}$$

$$\text{s.a.} \quad x_{ih} + \sum_{u=h}^{\min(h+P_i+S_{ij}-1,h_L)} x_{ju} \leq 1 \qquad \forall\, i,j \in I, \quad \forall\, h \in H \text{ e } i \neq j \tag{3.17}$$

$$\sum_{h=h_0}^{h_L} x_{ih} = 1 \qquad \forall\, i \in I \tag{3.18}$$

$$\sum_{h=h_0}^{h_L} x_{ih} h + P_i + e_i \geq E_i \qquad \forall\, i \in I \tag{3.19}$$

$$\sum_{h=h_0}^{h_L} x_{ih} h + P_i - t_i \leq T_i \qquad \forall\, i \in I \tag{3.20}$$

$$e_i \geq 0 \qquad \forall\, i \in I \tag{3.21}$$

$$t_i \geq 0 \qquad \forall\, i \in I \tag{3.22}$$

$$x_{ih} \in \{0,\, 1\} \quad \forall\, i \in I \text{ e } \forall\, h \in H \tag{3.23}$$

A função objetivo representada pela equação (3.16) tem como critério de otimização a minimização da soma ponderada das antecipações e dos atrasos. As restrições (3.17) garantem que existe tempo suficiente para executar uma tarefa i e preparar a máquina antes do início do processamento da tarefa seguinte j, ou seja, elas garantem que o processamento da tarefa j não será iniciado antes do término do processamento da tarefa i. As restrições (3.18) garantem que cada tarefa seja executada uma única vez. As restrições (3.19) e (3.20) definem as antecipações e os atrasos de acordo com as respectivas janelas de entrega de cada tarefa. As restrições (3.21), (3.22) e (3.23) determinam os domínios das variáveis do problema.

As restrições (3.17) determinam que se uma tarefa i é iniciada no instante h, qualquer outra tarefa j, sucessora à tarefa i, poderá iniciar somente a partir da data $h + P_i + s_{ij}$, ou seja, depois do término da execução da tarefa i e da preparação da máquina para a tarefa j após a tarefa i. No entanto, caso exista uma tarefa k tal que $S_{ik} + P_k + S_{kj} < s_{ij}$, isto é, tal que o tempo de *setup* da tarefa i para a tarefa k adicionado ao tempo de execução da tarefa k e ao tempo de *setup* da tarefa k para a tarefa j seja menor que o tempo de *setup* da tarefa i para a tarefa j, o processamento da tarefa j poderá ser iniciado antes da data $h + P_i + s_{ij}$, desde que a tarefa k seja sequenciada exatamente entre as tarefas i e j. Neste último caso, o processamento da tarefa j poderá ser iniciado a partir do instante $h + P_i + S_{ik} + P_k + S_{kj}$, que antecede o instante $h + P_i + s_{ij}$, pois $S_{ik} + P_k + S_{kj} < s_{ij}$. Desta forma, diferentemente do MPLIM-G e do MPLIM-BG, o MPLIM-IT somente é válido se o problema satisfizer a desigualdade triangular para os tempos de *setup*, ou seja, se as condições (3.24) a seguir forem verificadas:

$$S_{ij} \leq S_{ik} + P_k + S_{kj} \qquad \forall\, i,\, j,\, k \in I,\; i \neq j,\; i \neq k \text{ e } j \neq k \tag{3.24}$$

No PSUMAA-JE-TPD abordado, a execução de qualquer tarefa pode ser iniciada em qualquer momento futuro ($H = [0,\ +\infty[$). Portanto, caso se tenha a garantia de que na sequência ótima o início do processamento da primeira tarefa não ocorre antes de h_{inf} e que o início do processamento da última tarefa é no máximo h_{sup}, pode-se discretizar o conjunto H no intervalo $[h_{inf},\ h_{sup}]$, isto é, adotar $H = \{h_{inf},\ h_{inf}+1,\ h_{inf}+2,\ \ldots\ , h_{sup}\}$. Sendo assim, tem-se que o MPLIM-IT é fortemente dependente do intervalo $[h_{inf},\ h_{sup}]$, visto que ele determina o horizonte de planejamento e, consequentemente, o número de variáveis e restrições deste modelo.

É de fundamental importância escolher o intervalo $[h_{inf},\ h_{sup}]$ de modo que ele garanta a existência de uma solução viável, contenha a solução ótima e facilite a obtenção de tal solução. Por outro lado, a amplitude desse intervalo não deve ser muito grande, para que não se tenha muitas restrições, variáveis e um modelo mais difícil de ser resolvido na otimalidade.

Os valores de h_{inf} e h_{sup} utilizados são apresentados na Seção 5.2.

Capítulo 4

Algoritmo Proposto

Na prática, a aplicação das formulações de programação matemática apresentadas na Seção 3 é limitada a problemas de pequeno porte, ou seja, a problemas com poucas tarefas. Com o objetivo de resolver casos do PSUMAA-JE-TPD de maior dimensão (com um número de tarefas mais elevado), neste capítulo são propostos algoritmos heurísticos que combinam os procedimentos GRASP (ver Seção 2.2.3), Princípio da Otimalidade Próxima - POP (ver Seção 2.2.4) e Descida em Vizinhança Variável - VND (ver Seção 2.2.3). Este método híbrido é denominado GPV.

4.1 GPV

Basicamente, a metodologia proposta – denotada por GPV – é composta de duas fases. Na primeira fase (linhas 3 a 12 do Algoritmo 10), repetidamente, constrói-se uma solução, baseada no procedimento de construção do GRASP e no POP, para ser refinada por um VND. Na segunda fase (linha 13 do Algoritmo 10), faz-se o pós-refinamento da solução proveniente da fase anterior por meio de outro VND. O GPV pode ser representado pelo Algoritmo 10. A função de avaliação f, a função adaptativa gulosa g, a vizinhança V, o número máximo de iterações sem melhora ($GRASPmax$) do GRASP, o número máximo de iterações sem melhora ($VNDmax$) do VND_1, o número máximo de iterações sem melhora ($MDRmax$) do método de descida randômica (MDR) e o parâmetro γ da fase de construção GRASP são as entradas do método.

4.2 Representação de uma Solução

Uma solução (sequência) para o PSUMAA-JE-TPD de n tarefas é representada por um vetor v de n posições, em que cada posição $i = 1,\ 2,\ \cdots,\ n$ indica a ordem de produção da tarefa v_i. Por exemplo, dada a sequência $v = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$, a tarefa 5 é a primeira a ser executada e a tarefa 6 a última.

4.3 Vizinhança de uma Solução

Para explorar o espaço de soluções são usados três tipos de movimentos: troca da ordem de processamento de duas tarefas da sequência de produção, realocação de uma

Algoritmo *GPV*$(f(.),\ g(.),\ V(.),\ GRASPmax,\ VNDmax,\ MDRmax,\ \gamma)$
1 $f^{\star} \leftarrow +\infty$;
2 $Iter \leftarrow 0$;
3 enquanto $(Iter < GRASPmax)$ faça
4 $\quad Iter \leftarrow Iter + 1$;
5 $\quad v_0 \leftarrow ConstroiSolucaocomPOP(f(.),\ g(.),\ N^R(.),\ MDRmax,\ \gamma)$;
6 $\quad v \leftarrow VND_1(v_0,\ V(.),\ VNDmax)$;
7 $\quad$ se $(f(v) < f^{\star})$ então
8 $\quad\quad v^{\star} \leftarrow v$;
9 $\quad\quad f^{\star} \leftarrow f(v)$;
10 $\quad\quad Iter \leftarrow 0$;
11 $\quad$ fim-se
12 fim-enquanto;
13 $v^{\star} \leftarrow VND_2(v^{\star},\ V(.))$;
14 Retorne $v^{\star}$;
fim *GPV*;

Algoritmo 10: Método GPV

tarefa para outra posição da sequência e realocação de um bloco de tarefas para outra posição da sequência. Esses movimentos definem, respectivamente, as vizinhanças N^T, N^R e N^{RB}.

Na vizinhança N^T, um exemplo de vizinho da solução $v = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$ é a solução $v' = (5,\ 4,\ 2,\ 1,\ 3,\ 6)$, pois v' é obtida por meio da troca da ordem de processamento da segunda com a quinta tarefa de v, como pode ser visto na Figura 4.1.

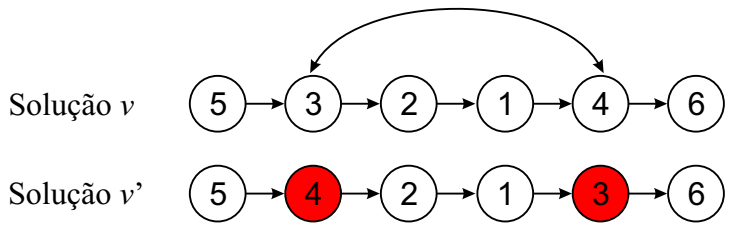

Figura 4.1: Movimento de troca - N^T.

Dada uma sequência de produção de n tarefas, cada tarefa na sequência pode trocar de posição com qualquer uma das outras $n-1$ tarefas. Ademais, trocar a ordem de processamento da i-ésima tarefa na sequência com a j-ésima tarefa é equivalente à troca da ordem de processamento da j-ésima tarefa na sequência com a i-ésima tarefa, para todo $i,\ j \in \{1,\ 2,\ \cdots,\ n\}$. Sendo assim, para uma solução do PSUMAA-JE-TPD com n tarefas, há $n(n-1)/2$ vizinhos distintos na vizinhança N^T.

A solução $v' = (5,\ 2,\ 1,\ 4,\ 3,\ 6)$ é um exemplo de vizinho da solução $v = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$ na vizinhança N^R, pois é obtida pela realocação da tarefa 3, que está na segunda posição em v, para a quinta posição na sequência de produção. Tal movimento é ilustrado na Figura 4.2.

Dada uma sequência de produção de n tarefas, cada tarefa na sequência pode ser

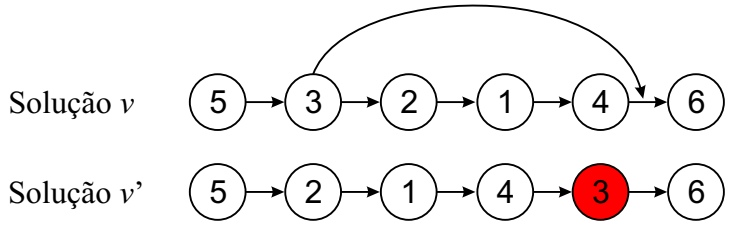

Figura 4.2: Movimento de realocação - N^R.

realocada para outras $n-1$ posições distintas. Além disso, realocar a tarefa da i-ésima posição para a posição $i+1$ na sequência é equivalente a realocar a tarefa da posição $i+1$ para a i-ésima posição na sequência, para todo $i \in \{1,\ 2,\ \cdots,\ n-1\}$. Sendo assim, para uma solução do PSUMAA-JE-TPD com n tarefas, há $n(n-1)-(n-1) = (n-1)^2$ vizinhos distintos na vizinhança N^R.

Já na vizinhança N^{RB}, um exemplo de vizinho da solução $v = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$ é a solução $v' = (1,\ 4,\ 5,\ 3,\ 2,\ 6)$, pois é obtida pela realocação do bloco de três tarefas $<5,\ 3,\ 2>$, que está sequenciado no início de v, para depois da tarefa 4 na sequência de produção, ou seja, para duas posições sucessoras. A Figura 4.3 ilustra tal movimento.

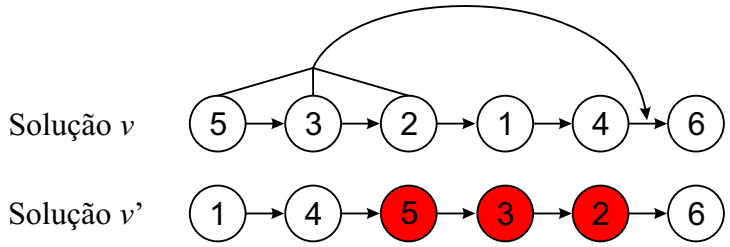

Figura 4.3: Movimento de realocação de um bloco de tarefas- N^{RB}.

Dada uma sequência de produção de n tarefas, existem $n-k+1$ blocos distintos de k tarefas, para todo $k \in \{1,\ 2,\ \cdots,\ n\}$, e cada um destes blocos podem ser realocados para $n-k$ posições distintas na sequência de produção. Além disso, realocar um bloco de i tarefas para j posições sucessoras é equivalente à realocação de um bloco de j tarefas para i posições antecessoras, para todo $i,\ j \in \{1,\ 2,\ \cdots,\ n\}$ e $i+j \leq n$. Sendo assim, para uma solução do PSUMAA-JE-TPD com n tarefas, há $\left(\sum_{k=1}^{n}(n-k+1)(n-k)\right)/2 = \left(\sum_{k=1}^{n}(n-k)^2 + \sum_{k=1}^{n}(n-k)\right)/2 = = \Big(n\,(2\,n-1)\,(n-1)\,/6 + n\,(n-1)\,/2\Big)/2 = n\,(n-1)\,(n+1)\,/6$ vizinhos distintos na vizinhança N^{RB}.

Com o objetivo de evitar a visita a vizinhos já visitados e consequentemente reduzir o custo computacional, todas as observações mencionadas nos parágrafos acima foram consideradas na implementação computacional do algoritmo proposto.

4.4 Função de Avaliação

Como os movimentos utilizados não geram soluções inviáveis, uma sequência v é avaliada pela função f dada pela equação (4.1), a qual deve ser minimizada:

$$f(v) = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{4.1}$$

Para determinar os valores de e_i e t_i para uma dada sequência de tarefas, é utilizado o algoritmo de determinação das datas ótimas de início de processamento das tarefas (ADDOIPT) apresentado a seguir.

4.4.1 Algoritmo de Determinação das Datas Ótimas de Início de Processamento das Tarefas

Dada uma sequência de produção, é utilizado o algoritmo de determinação das datas ótimas de início de processamento das tarefas (ADDOIPT) proposto por Gomes Júnior (2007) e baseado nos trabalhos de Wan & Yen (2002) e Lee & Choi (1995). O restante desta subseção é destinado à apresentação deste algoritmo.

Seja $V = (J_1, J_2, \ldots, J_n)$ uma sequência do conjunto de tarefas I. Logo $J_i \in I$ e $J_i \neq J_j$, para todo $i,\ j \in \{1, 2, \cdots,\ n\}$. Diz-se que uma sub-sequência de tarefas $B = (J_u, J_{u+1}, \ldots, J_v) \subseteq V$ forma um bloco na sequência V se as tarefas em B são sequenciadas consecutivamente sem tempo ocioso entre elas e existe ociosidade entre as tarefas J_{u-1} e J_u e as tarefas J_v e J_{v+1}.

Considere que há l blocos em V, ou seja, $V = B_1 \cup B_2 \cup \ldots \cup B_l$. Sejam $prim(B_j)$ e $ult(B_j)$ a primeira e a última tarefa no bloco B_j, respectivamente.

O processamento da primeira tarefa (J_1) da sequência V é programado para ser finalizado na data E_{J_1} se $P_{J_1} \leq E_{J_1}$ ou iniciado na data 0 (finalizando na data P_{J_1}) se $P_{J_1} > E_{J_1}$.

Seja C_i a data de conclusão da tarefa i.

As demais tarefas são programadas da seguinte forma:

- Se $C_{J_{i-1}} + S_{(J_{i-1})(J_i)} + P_{J_i} \geq E_{J_i}$, a tarefa J_i tem seu processamento programado para ser finalizado na data $C_{J_{i-1}} + S_{(J_{i-1})(J_i)} + P_{J_i}$ e passa a ser a última tarefa do bloco corrente.

- Caso contrário, se $C_{J_{i-1}} + S_{(J_{i-1})(J_i)} + P_{J_i} < E_{J_i}$, a tarefa J_i tem seu processamento programado para ser finalizado na data E_{J_i} e um novo bloco é iniciado.

Após a determinação da data de conclusão do processamento de cada tarefa, é necessário verificar o posicionamento dos blocos.

A antecipação e o atraso de uma tarefa i na sequência são dados, respectivamente por $\max\{0, E_i - C_i\}$ e $\max\{0, C_i - T_i\}$, respectivamente. Consequentemente, o custo gerado por antecipação e atraso de uma tarefa i concluída na data C_i pode ser dado pela função $g_i(C_i) = \alpha_i \cdot \max\{0, E_i - C_i\} + \beta_i \cdot \max\{0, C_i - T_i\}$. Logo a função custo para o bloco B_j da mudança de x unidades de tempo da data $E_{prim(B_j)}$ para a esquerda pode ser expressa pela equação:

$$Custo_j(x) = \sum_{i \in B_j} g_i(C_i - x), \tag{4.2}$$

em que $C_i - x$ é a nova data de conclusão do processamento da tarefa i.

Claramente a função $g_i(C_i - x)$ é uma função linear convexa por partes em relação a x para todo $i \in \{1,\ 2,\ \cdots,\ n\}$.

Lema 1 ***(Wan & Yen, 2002)*** *O somatório de duas funções lineares convexas por partes é também uma função linear convexa por partes.*

Baseado no Lema 1, tem-se:

Proposição 1 ***(Wan & Yen, 2002)*** *$Custo_j(x)$ é uma função linear convexa por partes em relação a x, para todo $j \in \{1,\ 2,\ \cdots,\ l\}$.*

Devido à natureza linear convexa por partes da função custo, o custo mínimo do bloco B_j ocorre nos pontos extremos de sua função $Custo_j$, isto é, no início ou no final da janela de entrega de uma das tarefas no bloco. Tal custo mínimo pode ser facilmente obtido pela comparação dos somatórios das penalidades das tarefas no bloco, onde cada um destes somatórios é obtido ao programar uma tarefa do bloco para ser concluída no início ou no final de sua janela de entrega. Todos os possíveis somatórios são calculados e o menor deles corresponde ao ponto de custo mínimo do bloco.

Quando o ponto mínimo do bloco B_j é encontrado, todo o bloco é movido em direção ao ponto mínimo até que um dos três casos seguintes aconteça:

(i) $s_{prim(B_j)} = 0$;

(ii) O ponto mínimo é alcançado;

(iii) $s_{prim(B_j)}$ torna-se igual a $C_{ult(B_{j-1})} + S_{ult(B_{j-1})\,(prim(B_j)}$.

Se ocorrer o caso (iii), o bloco B_j é unido ao bloco B_{j-1}, resultando em um novo bloco B_{j-1}. Logo o ponto mínimo do novo bloco B_{j-1} deve ser obtido.

O procedimento anterior deve ser realizado até que ocorra o caso (i) ou (ii) para cada bloco.

A seguinte proposição garante que o algoritmo apresentado determina as datas ótimas de início do processamento das tarefas de uma dada sequência.

Proposição 2 ***(Wan & Yen, 2002)*** *O custo total de uma dada sequência de tarefas alcança seu valor ótimo se cada bloco B_j na sequência alcançar seu ponto mínimo, exceto $s_{prim(B_1)}$, que pode ser igual a zero para o primeiro bloco.*

O Algoritmo 11, adaptado por Gomes Júnior (2007), descreve o procedimento AD-DOIPT. Nele, o procedimento *PontoMinimo*(B) calcula a data ótima de início de processamento do bloco B.

```
   procedimento DeterminaDatasOtimasInicio(n, s)
 1 B ← J_1;
 2 prim(B) ← ult(B) ← J_1;
 3 s_{J_1} ← max(0, E_{J_1} − P_{J_1});
 4 C_{J_1} ← max(E_{J_1}, P_{J_1});
 5 para (i = 2, ..., n) faça
 6     se (C_{J_{i−1}} + P_{J_i} + S_{(J_{i−1})(J_i)} < E_{J_i}) então
 7         B ← B + 1;                                  /* Cria um novo bloco */
 8         prim(B) ← ult(B) ← J_i;
 9         s_{J_i} ← E_{J_i} − P_{J_i} + S_{(J_{i−1})(J_i)};
10         C_{J_i} ← E_{J_i};
11     senão
12         ult(B) ← J_i;
13         s_{J_i} ← C_{J_{i−1}} + S_{(J_{i−1})(J_i)};
14         se (C_{J_{i−1}} + P_{J_i} + S_{(J_{i−1})(J_i)} = E_{J_i}) então
15             C_{J_i} ← E_{J_i};
16         senão
17             C_{J_i} ← s_{J_i} + P_{J_i};
18         fim-se
19     fim-se
20     repita ((até todos os blocos estiverem no ponto mínimo) ou (s_{J_1} = 0))
21         se (prim(B) = J_1) então
22             se (PontoMinimo(B) < 0) então
23                 s_{J_1} ← 0;
24             senão
25                 s_{J_1} ← PontoMinimo(B);
26             fim-se
27             C_{J_1} ← s_{J_1} + P_{J_1};
28         senão
29             se (PontoMinimo(B) > C_{ult(B−1)}) então
30                 s_{prim(B)} ← PontoMinimo(B);
31             senão
32                 s_{prim(B)} ← C_{ult(B−1)};
33                 B ← B − 1 ;          /* Une o bloco B ao bloco antecessor */
34             fim-se
35             C_{prim(B)} ← s_{prim(B)} + P_{prim(B)};
36         fim-se
37         para (cada tarefa J_k sucessora à tarefa prim(B)) faça
38             s_{J_k} ← C_{J_{k−1}} + S_{(J_{k−1})(J_k)};
39             C_{J_k} ← s_{J_k} + P_{J_k};
40         fim-para;
41     fim-repita;
42 fim-para;
```

$f(s) \leftarrow \sum_{i=1}^{n} \alpha_{J_i} \cdot \max\{0, E_{J_i} - C_{J_i}\} + \beta_{J_i} \cdot \max\{0, C_{J_i} - T_{J_i}\}$;
43
44 Retorne $f(s)$;
fim *DeterminaDatasOtimasInicio*;

Algoritmo 11: Algoritmo para determinar as datas ótimas de início de processamento das tarefas.

Para exemplificar a utilização do ADDOIPT, Gomes Júnior (2007) utilizou o problema com 4 tarefas apresentado na Tabela 4.1, em que se apresenta o tempo de processamento, o custo por antecipação, o custo por atraso, os tempos de *setup*, a data de início e de término da janela de entrega de cada tarefa.

Tabela 4.1: Problema para exemplo de aplicação do ADDOIPT.

Tarefas	**Dados**					***setup***			
	P	E	T	α	β	1	2	3	4
1	3	3	7	9	18	0	1	1	2
2	5	11	16	10	20	1	0	3	2
3	4	5	9	7	14	1	3	0	2
4	5	7	13	8	16	2	2	2	0

Dada a sequência $v = (3, 4, 1, 2)$, a primeira tarefa da sequência, tarefa $J_1 = 3$, tem seu processamento programado para ser finalizado na data $C_{J_1} = C_3 = E_3 = 5$, que representa o início da sua janela de entrega. A Figura 4.4 descreve o primeiro passo da programação de tarefas. Em tal figura, os valores de m representam as inclinações das respectivas retas.

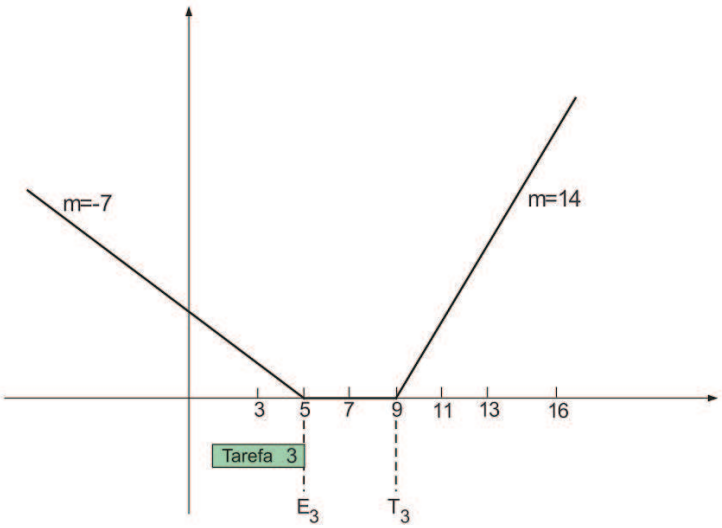

Figura 4.4: Inserção da primeira tarefa na sequência dada.

A Figura 4.5 exemplifica a programação da segunda tarefa da sequência, tarefa $J_2 = 4$, que tem sua conclusão programada para ser finalizada no instante $C_{J_2} = C_4 = 12$, pois $C_3 + S_{34} + P_4 \geq E_4$ $(12 > 7)$.

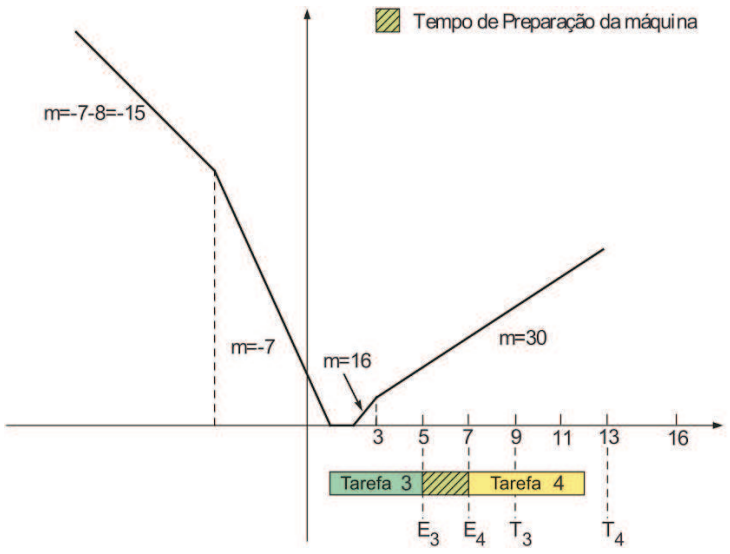

Figura 4.5: Inserção da segunda tarefa na sequência dada.

Neste momento, é necessário verificar a posição do bloco para verificar se ele está em seu ponto mínimo. Para isto, primeiramente, determina-se as possíveis datas de início do processamento do bloco. Essas datas são obtidas posicionando cada tarefa do bloco no início e no final de sua janela de entrega, pois, como foi visto, o custo mínimo do bloco se dá no início ou no final da janela de entrega de uma das tarefas no bloco. No exemplo mostrado, as possíveis datas de início do processamento do bloco são: -4, 1, 2 e 5. Determinadas as possíveis datas de início de processamento, calculam-se então as inclinações m de acordo com essas datas. Para calcular essas inclinações, as penalidades por antecipação assumem valores negativos e as penalidades por atraso assumem valores positivos. O ponto mínimo, o qual deseja-se encontrar, se dá no ponto aonde a inclinação m se torna maior ou igual a zero. No exemplo apresentado, se o processamento do bloco iniciar antes da data 4, ou seja, com todas as tarefa sendo antecipadas, então a inclinação m é dada pelo somatório das penalidades por antecipação das tarefas 3 e 4, portanto $m = -7 - 8 = -15$. Agora, se o processamento do bloco iniciar entre a data -4 e 1, a tarefa 4 deixa de estar antecipada, e a inclinação m passa a ser $m = -7$ (penalidade por antecipação da tarefa 3). Agora, se o processamento do bloco iniciar entre a data 1 e 2, a tarefa 3 deixa de estar antecipada, e a inclinação m passa a ser $m = 0$. Como m tornou-se maior ou igual a zero, este ponto é o ponto mínimo do bloco, ou seja, o custo mínimo do bloco se dá quando o processamento do bloco se inicia na data 1. Como o início do processamento bloco já é na data 1 então não é necessário movimentar o bloco até esta data. A Figura 4.5 mostra, também, tais valores de m.

A terceira tarefa da sequência, tarefa $J_3 = 1$, tem sua conclusão programada para ser finalizada na data $C_{J_3} = C_1 = 17$, pois $C_4 + S_{41} + P_1 \geq E_1$ $(17 > 3)$. Tal situação é ilustrada na Figura 4.6. Observe que, neste caso, a tarefa 1 é inserida como último elemento do bloco corrente.

Conhecida a data de conclusão da tarefa 1, deseja-se agora verificar a posição do bloco, para saber se ele está em seu ponto mínimo. Para isto, primeiramente, são determinadas as possíveis datas de início do processamento do bloco. Essas datas são obtidas posicionando cada tarefa do bloco para ser concluída no início e no final de sua janela de entrega. Sendo

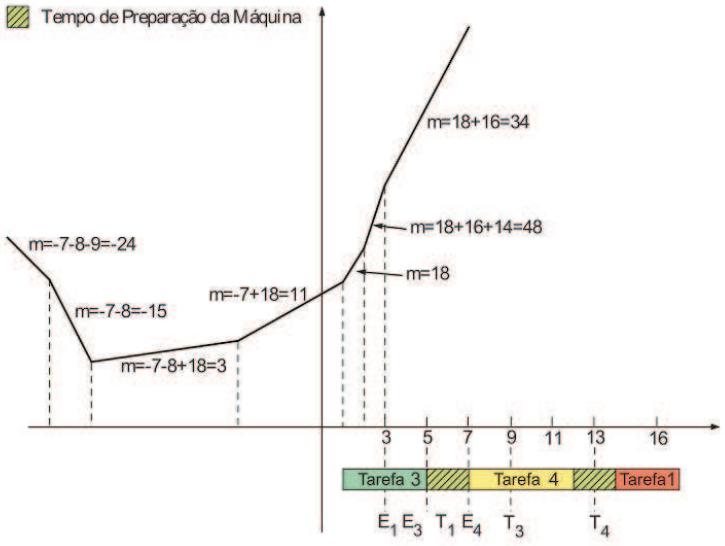

Figura 4.6: Inserção da terceira tarefa na sequência dada.

assim, as possíveis datas de início do processamento do bloco são: -13, -11, -4, 1, 2 e 5. As inclinações m calculadas de acordo com estas datas são mostradas, também, na Figura 4.6, onde verifica-se que a inclinação m torna-se maior ou igual a zero na data -11, sendo este, então, o ponto mínimo do bloco. Como pode ser visto também nesta figura, o início de processamento do bloco se encontra no instante 1, então o bloco deve ser movido até um dos três casos citados anteriormente acontecerem. No exemplo considerado, o bloco é empurrado até o caso (i) ocorrer, ou seja, $s_{prim(B_1)} = s_{J_1} = s_3 = 0$. Desta forma, tem-se: $s_3 = 0$ e $C_3 = 4$; $s_4 = 6$ e $C_4 = 11$; $s_1 = 13$ e $C_1 = 16$. A Figura 4.7 apresenta o bloco já movido e tendo seu início programado para iniciar no instante 0.

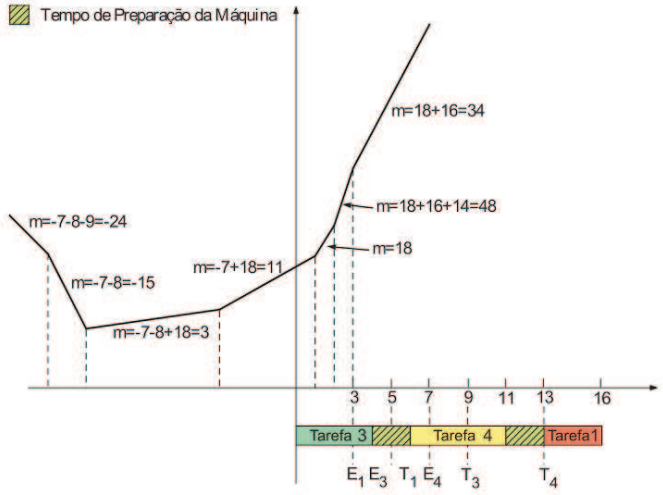

Figura 4.7: Bloco movido até o ponto de mínimo.

Este procedimento se repete até que todas as tarefas sejam sequenciadas.

4.5 Construção de uma Solução

Nesta etapa do GPV (linha 5 do Algoritmo 10), uma solução é formada, tarefa por tarefa, de forma parcialmente gulosa, seguindo as idéias da fase de construção do algoritmo GRASP (ver Seção 2.2.3) e do POP (ver Seção 2.2.4). Sendo assim, a cada iteração desta etapa, as tarefas que ainda não foram sequenciadas são ordenadas por meio de função adaptativa gulosa (g) que estima o benefício associado a inclusão de cada tarefa na subsequência corrente. As melhores tarefas, segundo a função g, são inseridas em uma lista restrita de candidatos (LRC). Desta lista, aleatoriamente e com probabilidade uniforme, escolhe-se a tarefa que será incrementada à subsequência corrente.

O tamanho da LRC e o nível de gulosidade e de aleatoriedade da solução construída são determinados por um parâmetro $\gamma \in [0, 1]$ tal que: quanto mais próximo de 0 estiver γ , maior o nível de gulosidade e menor o nível de aleatoriedade. Desta forma, $\gamma = 0$ implica em uma solução completamente gulosa e $\gamma = 1$ implica em uma solução totalmente aleatória.

Ainda na etapa de construção de uma solução, é aplicado o POP (linha 9 do Algoritmo 12). Assim, sempre que uma nova tarefa é inserida na solução em formação, esta é submetida a uma busca local. A busca local utilizada consiste em um método de descida randômica (MDR).

O Algoritmo 12 representa o procedimento proposto para construir uma solução. A função de avaliação de uma solução (f), a função adaptativa gulosa (g), a vizinhança (V), o número máximo de iterações sem melhora do MDR ($MDRmax$) e o parâmetro γ são as entradas do método.

procedimento *ConstroiSolucaocomPOP*($f(.)$, $g(.)$, $N^R(.)$, $MDRmax$, γ)
1 $s \leftarrow \emptyset$;
2 Inicialize o conjunto C de tarefas;
3 enquanto ($C \neq \emptyset$) faça
4 $g(t_{min}) \leftarrow \min\{g(t) \mid t \in C\}$;
5 $g(t_{max}) \leftarrow \max\{g(t) \mid t \in C\}$;
6 $LRC \leftarrow \{t \in C \mid g(t) \leq g(t_{min}) + \gamma(g(t_{max}) - g(t_{min}))\}$;
7 Aleatoriamente, selecione um elemento $t \in LRC$ com probabilidade uniforme;
8 $s \leftarrow s \cup \{t\}$;
9 $s \leftarrow MDR(s,\ N^R(.),\ MDRmax)$; `/* Aplica o POP */`
10 Atualize o conjunto C de candidatos;
11 fim-enquanto;
12 Retorne s;
fim *ConstroiSolucaocomPOP*;

Algoritmo 12: Construção de uma solução com POP.

Mais detalhes do procedimento *ConstroiSolucaocomPOP* são apresentados a seguir.

Foram experimentadas 11 funções adaptativas gulosas para estimar o benefício da inserção de cada tarefa j ainda não sequenciada na iteração i. Tais funções foram motivadas pelas características do problema e são apresentadas a seguir:

- $g_1(j) = E_j$;

- $g_2(j) = T_j$;
- $g_3(j) = E_j + T_j$;
- $g_4(j) = T_j - P_j$;
- $g_5(j) = E_j - P_j$;
- $g_6(j) = \dfrac{(E_j - P_j)\alpha_j + (T_j - P_j)\beta_j}{\alpha_j + \beta_j}$;
- $g_7(j) = E_j + T_j - 2P_j$;
- $g_8(j) = \dfrac{2E_j}{\max\{E_k \,|\, k \in F_i\}} + \dfrac{2T_j}{\max\{T_k \,|\, k \in F_i\}} + \dfrac{\alpha_j}{\max\{\alpha_k \,|\, k \in F_i\}} - \dfrac{\beta_j}{\max\{\beta_k \,|\, k \in F_i\}} + \dfrac{P_j}{\max\{P_k \,|\, k \in F_i\}} + \dfrac{S_{i-1,j}}{\max\{S_{i-1,k} \,|\, k \in F_i\}}$;
- $g_9(j) = \dfrac{E_j}{\max\{E_k \,|\, k \in F_i\}} + \dfrac{2T_j}{\max\{T_k \,|\, k \in F_i\}} + \dfrac{\alpha_j}{\max\{\alpha_k \,|\, k \in F_i\}} - \dfrac{\beta_j}{\max\{\beta_k \,|\, k \in F_i\}} + \dfrac{P_j}{\max\{P_k \,|\, k \in F_i\}} + \dfrac{S_{i-1,j}}{\max\{S_{i-1,k} \,|\, k \in F_i\}}$;
- $g_{10}(j) = \dfrac{E_j}{\max\{E_k \,|\, k \in F_i\}} + \dfrac{T_j}{\max\{T_k \,|\, k \in F_i\}} + \dfrac{\alpha_j}{\max\{\alpha_k \,|\, k \in F_i\}} - \dfrac{\beta_j}{\max\{\beta_k \,|\, k \in F_i\}} + \dfrac{P_j}{\max\{P_k \,|\, k \in F_i\}} + \dfrac{S_{i-1,j}}{\max\{S_{i-1,k} \,|\, k \in F_i\}}$;
- $g_{11}(j) = \dfrac{2E_j}{\max\{E_k \,|\, k \in F_i\}} + \dfrac{T_j}{\max\{T_k \,|\, k \in F_i\}} + \dfrac{\alpha_j}{\max\{\alpha_k \,|\, k \in F_i\}} - \dfrac{\beta_j}{\max\{\beta_k \,|\, k \in F_i\}} + \dfrac{P_j}{\max\{P_k \,|\, k \in F_i\}} + \dfrac{S_{i-1,j}}{\max\{S_{i-1,k} \,|\, k \in F_i\}}$.

Nas funções $g_8,\ g_9,\ g_{10},$ e g_{11}, F_i representa o conjunto das tarefas ainda não sequenciadas até a i-ésima iteração (para $i = 1$, no lugar de $\dfrac{S_{i-1,j}}{\max\{S_{i-1,k} \,|\, k \in F_i\}}$, é utilizado $- \dfrac{\text{média}\{S_{kj} \,|\, k \in I \text{ e } k \neq j\}}{\max\{S_{kl} \,|\, k, l \in I \text{ e } k \neq l\}}\Big)$.

Cada função g_k, $k = 1,\ 2,\ \cdots, 11$, foi utilizada para construir uma solução gulosa para cada um dos 608 problemas-teste com até 100 tarefas apresentados na Seção 5.1. A qualidade da solução encontrada por cada função adaptativa gulosa g_k, aplicada a um determinado problema-teste, foi mensurada pela equação (4.3), onde f_{g_k} representa o valor da função objetivo da solução encontrada pela função adaptativa gulosa g_k e $f_{g^\star}$ denota o valor da função objetivo da melhor solução encontrada pelas 11 funções gulosas.

$$Desvio_{g_k} = \frac{f_{g_k} - f_{g^\star}}{f_{g^\star}} \tag{4.3}$$

Os resultados encontrados estão sintetizados na Tabela 4.2. Nesta tabela, a primeira linha indica as funções adaptativas gulosas testadas. Já na segunda linha, são apresentas as médias dos desvios encontrados pelas respectivas funções aplicadas aos 608 problemas-teste.

Tabela 4.2: Resultados apresentados pelas funções adaptativas gulosas testadas.

Função	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_{11}
Desvio Médio	0,47	0,46	0,45	0,54	0,54	0,56	0,53	0,28	0,38	0,57	0,36

Na Tabela 4.2 é possível observar que as funções g_1, g_2, g_3, g_8, g_9 e g_{11} apresentaram desvios médios menores que 0,5. Sendo assim, optou-se por utilizar estas seis funções, uma em cada iteração em que o procedimento *ConstroiSolucaocomPOP* é utilizado. Tais funções são utilizadas segundo a ordem crescente do desvio médio encontrado, ou seja, na primeira iteração utiliza-se a função g_8, na segunda utiliza-se a função g_{11} e assim por diante, até que todas a seis funções sejam utilizadas; repetindo-se a aplicação desta sequência a seguir.

Para determinar a vizinhança a ser utilizada no MDR relativo ao POP, foi realizado o seguinte procedimento em um problema-teste com 30 tarefas. Cada uma das seis funções adaptativas gulosas determinadas anteriormente foi utilizada para construir uma solução para o problema. As soluções construídas foram, então, submetidas ao procedimento de Descida de Primeira Melhora com a vizinhança N^T. Este mesmo procedimento foi realizado com a vizinhança N^R. Como a ordem de visita aos vizinhos é determinante em um método de descida, o procedimento descrito foi realizado 20 vezes com cada vizinhança, sendo a ordem de visitas sempre escolhida aleatoriamente. No final de todas as execuções, as soluções obtidas com a vizinhança N^R apresentaram uma média 9,55% menor que a média das soluções obtidas com a vizinhança N^T. Além disso, a melhor solução obtida com a vizinhança N^R foi $7,63\%$ melhor que a melhor solução obtida com a N^T. Estes resultados sugerem que a vizinhança N^R é capaz de encontrar soluções de melhor qualidade e com melhor média que as soluções obtidas com a vizinhança N^T e, por isso, foi a escolhida para o método de descida usado no POP.

Como o POP é aplicado sempre que uma tarefa é inserida na solução em formação, utilizou-se uma descida randômica com vizinhança N^R, visto que este método de descida demanda menos tempo computacional que a Descida de Primeira Melhora. Sendo assim, dada uma solução, aleatoriamente escolhe-se uma tarefa na sequência e uma nova posição para ela. Se a nova sequência produzir uma solução com um valor menor para a função de avaliação, a nova sequência é aceita e passa a ser a solução corrente; caso contrário, é testada outra realocação aleatória. A busca é interrompida após *MDRMax* realocações consecutivas sem melhora na solução corrente, sendo *MDRMax* um parâmetro do método.

4.6 VND_1

Para refinar as soluções construídas na primeira fase do GPV (linha 6 do Algoritmo 10), utiliza-se um procedimento baseado em Descida em Vizinhança Variável (ver Seção 2.2.3), denotado por VND_1. Basicamente o VND_1 consiste em três passos:

***Passo 1*:** Descida randômica com a vizinhança N^T;

***Passo 2*:** Descida randômica com a vizinhança N^R;

***Passo 3*:** Descida randômica com a vizinhança N^{RB};

No *Passo 1*, duas tarefas são escolhidas de forma aleatória e a ordem de seus processamentos na sequência de produção é trocada. Se a nova sequência produzir uma solução com um valor menor para a função de avaliação, a nova sequência é aceita e passa a ser a solução corrente; caso contrário, é testada outra troca aleatória. O *Passo 1* termina quando ocorrer *VNDmax* trocas consecutivas sem melhora na solução corrente, sendo *VNDMax* um parâmetro do método. Neste último caso, passa-se para o *Passo 2*.

No *Passo 2*, aleatoriamente escolhe-se uma tarefa na sequência e uma nova posição para ela. Se a nova sequência produzir uma solução com um valor menor para a função de avaliação, a nova sequência passa a ser a solução corrente e volta-se ao *Passo 1*; caso contrário, é testada outra realocação aleatória. O *Passo 2* é interrompido após *VNDMax* realocações consecutivas sem melhora na solução corrente. Neste último caso, passa-se para o *Passo 3*.

No *Passo 3*, aleatoriamente escolhe-se um bloco de tarefas na sequência de produção (o tamanho do bloco também é escolhido de forma aleatória dentro do intervalo $[1, n-1]$) e uma nova posição para ele. Se a nova sequência produzir uma solução com um valor menor para a função de avaliação, a nova sequência passa a ser a solução corrente e volta-se ao *Passo 1*; caso contrário, é testada outra realocação de bloco aleatória. O *Passo 3* é interrompido após *VNDMax* realocações de blocos consecutivas sem melhora na solução corrente. Neste último caso, o VND_1 é interrompido e a melhor solução encontrada é retornada.

É interessante observar que a solução oriunda do VND_1 não é necessariamente um ótimo local em relação às vizinhanças adotadas, visto que nem toda a vizinhança é analisada em cada cada passo.

4.7 VND_2

Como a solução proveniente da primeira fase do GPV (linhas 1 a 12 do Algoritmo 10) não é necessariamente um ótimo local em relação às vizinhanças adotadas, ela é submetida a uma busca local mais efetiva, no caso, também baseada na Descida em Vizinhança Variável (VND_2). Nesta, a exploração do espaço de soluções é realizada de acordo com os seguintes passos:

***Passo 1*:** Descida Primeiro de Melhora com a vizinhança N^T;

***Passo 2*:** Descida completa com a vizinhança N^{RB};

Se no *Passo 1*, nenhum dos vizinhos com relação à vizinhança N^T é de melhora, passa-se para o *Passo 2*.

No *Passo 2*, em que são realizadas realocações de blocos de tarefas, inicialmente testam-se todas as possíveis realocações de blocos com apenas uma tarefa e, quando não for mais possível melhorar a solução com um determinado tamanho de bloco, passa-se a

explorar movimentos com blocos de tamanho imediatamente superior, até que o tamanho dos blocos seja igual a $n-1$. Se uma solução de melhora é encontrada, volta-se ao *Passo 1*. O procedimento é interrompido quando um ótimo local com relação às vizinhanças N^T e N^{RB} é encontrado. Observa-se que a vizinhança N^R corresponde a realocar blocos com apenas uma tarefa. Logo, a vizinhança N^R está contida na vizinhança N^{RB} e um ótimo local em relação à esta última também é um ótimo local em relação à primeira.

4.8 Redução do Espaço de Busca - GPV-REB

A avaliação de todos os vizinhos gerados em uma busca local requer um grande e muitas vezes injustificável esforço computacional (França Filho, 2007). Por isso, é interessante utilizar estratégias que tenham por objetivo evitar a avaliação de soluções que apresentam determinadas características julgadas ruins e, assim, reduzir o esforço computacional. Como consequência, mais vizinhos podem ser explorados ou, então, problemas de dimensões maiores podem ser resolvidos.

No caso do PSUMAA-JE-TPD, se duas tarefas $i, j \in I$ satisfazem as condições (4.4), (4.5), (4.6) e (4.7), espera-se que boas soluções possuem a tarefa i sequenciada antes da tarefa j.

$$E_i \leq E_j \tag{4.4}$$
$$\alpha_i \leq \alpha_j \tag{4.5}$$
$$T_i \leq T_j \tag{4.6}$$
$$\beta_i \geq \beta_j \tag{4.7}$$

Portanto, se i é sequenciada antes de j na sequência corrente, é proposta a seguinte estratégia de redução do espaço de busca:

- Durante a exploração da vizinhança N^T, a solução produzida pela troca das ordens de processamentos da tarefa i com a tarefa j é rejeitada sem ser avaliada.
- Durante a exploração da vizinhança N^R, as realocações da tarefa i para posições sucessoras à tarefa j são descartadas sem serem avaliadas. Do mesmo modo, as realocações da tarefa j para as posições anteriores à tarefa i também são descartadas.
- Durante a exploração da vizinhança N^{RB}, as realocações de blocos que levam a tarefa i a ser sequenciada depois da tarefa j na sequência de produção são descartadas sem serem avaliadas.

A Figura 4.8 ilustra a situação em que $E_3 \leq E_2$, $\alpha_3 \leq \alpha_2$, $T_3 \leq T_2$ e $\beta_3 \geq \beta_2$. Neste caso, as soluções obtidas pela troca das ordens de processamento da tarefa 3 com a tarefa 2 ou por realocações, de tarefas ou de blocos, que levam a tarefa 3 a ser sequenciada depois da tarefa 2 são descartadas antes de serem avaliadas.

A estratégia de redução do espaço de busca proposta foi utilizada em um algoritmo derivado do algoritmo GPV, denotado por GPV-REB. O GPV-REB se diferencia do GPV pela utilização da estratégia de redução do espaço de busca nas buscas locais da primeira

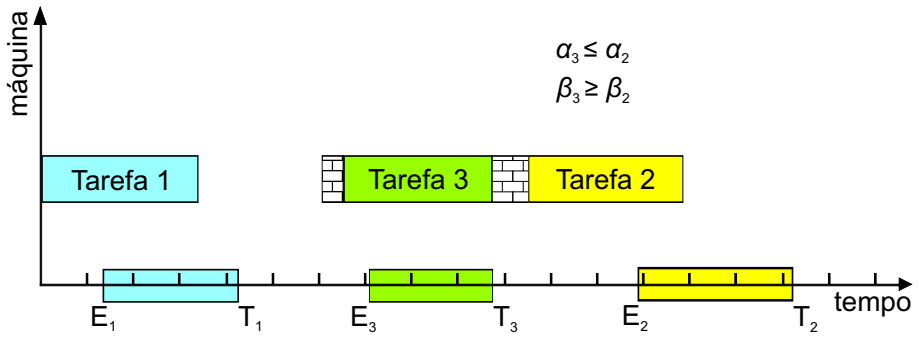

Figura 4.8: Exemplo de situação em que a estratégia de redução do espaço de busca é utilizada.

fase do algoritmo, ou seja, no método de descida randômica utilizado para refinar as soluções parciais na fase de construção de uma solução (MDR - linha 9 do Algoritmo 12) e no VND utilizado para refinar todas as soluções construídas (VND_1 - linha 6 do Algoritmo 10). A segunda fase dos dois algoritmos é a mesma.

Capítulo 5

Resultados Computacionais

Neste capítulo são apresentados os resultados computacionais obtidos pela aplicação das formulações de programação matemática propostas, bem como dos algoritmos heurísticos desenvolvidos. Primeiramente, são apresentados os conjuntos de problemas-teste utilizados. Na Seção 5.2 são apresentados os resultados obtidos com as formulações de programação matemática apresentadas no Capítulo 3. Na Seção 5.3 são apresentados os resultados obtidos com os algoritmos GPV e GPV-REB, bem como os procedimentos utilizados na determinação dos parâmetros de tais algoritmos.

Todos os experimentos computacionais foram realizados em um computador Intel(R) Core(TM) 2 Duo E8400 2.99 GHz, com 2 GB de memória RAM e sistema operacional Windows XP. Apesar de o processador deste equipamento possuir 2 núcleos, o algoritmo desenvolvido não foi otimizado para multiprocessamento.

5.1 Problemas-teste

Foram utilizados três conjuntos de problemas-teste. O primeiro deles, denotado por BDS, foi proposto por Gomes Júnior (2007), sendo baseado nos trabalhos de Wan & Yen (2002), Liaw (1999) e Mazzini & Armentano (2001). Esta base de dados é constituída por grupos de problemas-teste com 8, 9, 10, 11, 12, 15, 20, 25, 30, 40, 50 e 75 tarefas.

Dada uma tarefa i, o tempo de processamento (P_i), o custo por unidade de atraso (β_i) e o custo por unidade de antecipação (α_i) são valores inteiros selecionados, aleatoriamente e com distribuição uniforme, dentro dos intervalos $[1, 100]$, $[20, 100]$ e $[0, \beta_i]$, respectivamente.

O centro da janela de entrega de $i \in I$ é um valor inteiro aleatório dentro do intervalo $\left[\left(1 - FA - \frac{VRJ}{2}\right) \times TTP, \left(1 - FA + \frac{VRJ}{2}\right) \times TTP\right]$, onde TTP é o tempo total de processamento de todos as tarefas, FA é o fator de atraso e VRJ é a variação relativa da janela de entrega. São utilizados os valores 0,1; 0,2; 0,3 e 0,4 para FA e 0,8; 1,0 e 1,2 para VRJ.

Os tamanhos das janelas de entrega são uniformemente e discretamente distribuídos no intervalo $\left[1, \frac{TTP}{n}\right]$, onde n é o número de tarefas de I.

Para toda tarefa $j \in I$ e $j \neq i$, o tempo de *setup* (S_{ij}) é um inteiro aleatório dentro do intervalo $[0, 50]$. Utilizou-se tempos de preparação simétricos, ou seja, $S_{ij} = S_{ji}$.

Como são utilizados quatro valores distintos de *FA* e três valores distintos de *VRJ*, há 12 problemas-teste para cada número de tarefas, totalizando 144 problemas-teste. Doravante, esta base de dados será denotada por BDS.

O segundo conjunto de problemas-teste utilizado, denotado por BDNS, foi gerado considerando-se a metodologia apresentada nos trabalhos de Wan & Yen (2002) e Rabadi *et al.* (2004). Foram gerados grupos de problemas-teste com 6, 7, 8, 9, 10, 11, 12, 15, 20, 30, 40, 50, 75, 100 e 150 tarefas.

Dada uma tarefa $i \in I$, o tempo de processamento (P_i), o custo por unidade de atraso (β_i) e o custo por unidade de antecipação (α_i) são valores inteiros selecionados, aleatoriamente e com distribuição uniforme, dentro dos intervalos $[1, 100]$, $[1, 10]$ e $[1, \beta_i]$, respectivamente.

O centro da janela de entrega de i é um valor inteiro aleatório dentro do intervalo $\left[\left(1 - FA - \frac{VRJ}{2}\right) \times TTP, \left(1 - FA + \frac{VRJ}{2}\right) \times TTP\right]$, onde TTP é o tempo total de processamento de todos as tarefas, *FA* é o fator de atraso e *VRJ* é a variação relativa da janela de entrega. São utilizados os valores 0,1; 0,3; 0,5 e 0,8 para *FA* e 0,4; 0,7; 1,0 e 1,3 para *VRJ*.

O tamanho da janela de entrega de i é um número inteiro selecionado, aleatoriamente e com distribuição uniforme, dentro do intervalo $\left[0, \frac{TTP}{n}\right]$, onde n é o número de tarefas de I.

Para toda tarefa $j \in I$ e $j \neq i$, o tempo de *setup* (S_{ij}) é um inteiro aleatório dentro do intervalo $[5, 30]$. Nesta base de dados, os tempos de *setup* são genéricos e, assim, não são necessariamente simétricos como na base anterior.

Como são utilizados 4 valores distintos de TTP e 4 de VRJ, são 16 problemas-teste de cada tamanho, totalizando 240 problemas-teste. Esta nova base de dados será denotada por BDNS.

O terceiro conjunto de problemas-teste, denotado por BDDT, também foi baseado nos trabalhos de Rabadi *et al.* (2004) e Wan & Yen (2002), mas diferem do anterior na caracterização dos tempos de preparação, que, neste caso, satisfazem a desigualdade triangular. Os problemas-teste deste conjunto foram gerados do modo descrito a seguir.

Dada uma tarefa $i \in I$, o tempo de processamento (P_i), o custo por unidade de atraso (β_i) e o custo por unidade de antecipação (α_i) são números inteiros selecionados, aleatoriamente e com distribuição uniforme, dentro dos intervalos $[1, 40]$, $[1, 10]$ e $[1, \beta_i]$, respectivamente.

O centro da janela de entrega de i é um valor inteiro aleatório dentro do intervalo $\left[\left(1 - FA - \frac{VRJ}{2}\right) \times TTP, \left(1 - FA + \frac{VRJ}{2}\right) \times TTP\right]$, em que TTP é o tempo total de processamento de todos as tarefas, *FA* é o fator de atraso e *VRJ* é a variação relativa da janela de entrega.

O tamanho da janela de entrega é um valor inteiro selecionado, aleatoriamente e com distribuição uniforme, no intervalo $\left[0, \frac{TTP}{n}\right]$, sendo n o número de tarefas.

Para toda tarefa $j \in I$ e $j \neq i$, o tempo de *setup* (S_{ij}) é um número inteiro escolhido, aleatoriamente e com distribuição uniforme, dentro do intervalo $[5, 15]$. Assim, os tempos de *setup* também não são necessariamente simétricos.

Foram gerados conjuntos de problemas-teste com 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 e 20 tarefas, respectivamente, sendo utilizados os valores 0,1; 0,3; 0,5 e 0,8 para *FA* e 0,4; 0,7; 1,0 e 1,3 para *VRJ*. Deste modo, há 16 problemas-teste em cada conjunto, totalizando 240 problemas-teste.

Durante a geração dos problemas-teste desta última base de dados, sempre que um problema gerado não satisfazia a desigualdade triangular, ele era descartado e outro problema com os mesmos valores para *FA* e *VRJ* era gerado. Portanto, todos os problemas-teste desta base de dados satisfazem a desigualdade triangular.

5.2 Resultados Obtidos com as Formulações de Programação Matemática

Os modelos de programação matemática apresentados na Seção 3 foram implementados no modelador AMPL e resolvidos pelo otimizador CPLEX, versão 10.1, da ILOG, em sua configuração padrão. Foram realizados experimentos nos problemas-teste de pequeno porte das três bases de dados apresentadas na Seção 5.1, totalizando três baterias de testes.

Para os modelos MPLIM-G e MPLIM-BG, foi utilizado M = 1000, mesmo valor adotado por Gomes Júnior (2007). Para cada problema-teste, o conjunto H do MPLIM-IT foi determinado pelas expressões (5.1) e (5.2):

$$\begin{aligned} h_{inf} &= \max\Big(0,\ \min\Big\{T_i \mid i \in I\Big\} \ +\ \min\Big\{S_{ij} \mid i, j \in I \ \text{e}\ i \neq j\Big\} \ - \\ &\qquad - \sum_{i \in I} P_i \ - \ \sum_{i \in I} \max\Big\{S_{ij} \mid j \in I \ \text{e}\ i \neq j\Big\}\Big) \end{aligned} \tag{5.1}$$

$$\begin{aligned} h_{sup} &= \max\Big\{E_i \mid i \in I\Big\} \ +\ \sum_{i \in I} P_i \ +\ \sum_{i \in I} \max\Big\{S_{ij} \mid j \in I \ \text{e}\ i \neq j\Big\} \ - \\ &\quad - \ \min\Big\{S_{ij} \mid i, j \in I \ \text{e}\ i \neq j\Big\} \ - \ \min\Big\{P_i \mid i \in I\Big\} \ - \\ &\quad - \ \min\Big(\min\Big\{P_i \mid i \in I\Big\},\ \max\Big\{E_i \mid i \in I\Big\}\Big) \end{aligned} \tag{5.2}$$

em que h_{inf} garante que todas as tarefas podem ser concluídas sem atraso (ou o processamento da primeira tarefa é iniciado na data 0) e h_{sup} assegura que todas as tarefas podem ser executadas sem antecipação. Portanto, tem-se a garantia de que as datas de início de processamento das tarefas na sequência ótima pertencem ao conjunto $H = \{h_{inf}, h_{inf} + 1, h_{inf} + 2, \cdots, h_{sup}\}$.

Considerou-se, ainda, o limite de 3600 segundos para a resolução de cada problema-teste pelo CPLEX. Para os problemas em que este limite de tempo foi atingido, a solução retornada não é necessariamente ótima; porém, um limite inferior para o ótimo de tal problema é retornado pelo CPLEX. Este limite foi utilizado para medir a qualidade da solução retornada, que é obtida pela equação (5.3):

$$gap = \frac{(f^{CPLEX} - L)}{L} \times 100\% \tag{5.3}$$

em que f^{CPLEX} e L representam, respectivamente, o valor da função objetivo da solução e o limite inferior encontrados pelo otimizador. Deste modo, se uma solução ótima é encontrada, tem-se $gap = 0$.

Os resultados obtidos em cada bateria de testes são apresentados a seguir.

5.2.1 Primeira Bateria de Testes - BDS

A primeira bateria de testes foi realizada com os problemas-teste da base de dados BDS. Os problemas-teste dessa base de dados possuem os tempos de *setup* simétricos e não satisfazem, necessariamente, a desigualdade triangular. Deste modo, apenas os modelos MPLIM-G e MPLIM-BG foram utilizados nesta bateria, visto que o MPLIM-IT não permite resolver problemas em que a desigualdade triangular não é satisfeita.

Os resultados encontrados são resumidos na Tabela 5.1, onde a primeira coluna indica o número de tarefas dos problemas-teste de cada conjunto. Ademais, para cada conjunto de problemas, as colunas "Soluções Ótimas" mostram (em porcentagem) em quantos problemas o CPLEX encontrou uma solução ótima usando os respectivos modelos, as colunas "$\overline{gap}$" mostram as médias dos *gap*'s retornados pelos respectivos modelos e as colunas "Tempo Médio" mostram as médias dos tempos (em segundos) demandados por tal otimizador na resolução dos problemas-teste com os respectivos modelos.

Tabela 5.1: Comparação MPLIM-G × MPLIM-BG aplicados à BDS.

# Tarefas	MPLIM-G			MPLIM-BG		
	Soluções Ótimas (%)	$\overline{gap}$ (%)	**Tempo Médio** (s)	**Soluções Ótimas** (%)	$\overline{gap}$ (%)	**Tempo Médio** (s)
08	100,00	0,00	1,52	100,00	0,00	1,61
09	100,00	0,00	35,12	100,00	0,00	31,71
10	100,00	0,00	86,68	100,00	0,00	77,16
11	66,67	15,78	1850,01	75,00	10,53	1974,97
12	50,00	30,51	1957,23	50,00	30,98	1962,74
Média	83,33	9,26	786,11	85,00	8,30	809,64

Na Tabela 5.1, observa-se que o MPLIM-G e o MPLIM-BG proporcionaram ao CPLEX desempenhos semelhantes para a base de dados em questão. Usando-se esses dois modelos, o CPLEX foi capaz de resolver na otimalidade todos os problemas-teste com até 10 tarefas e com tempos médios de processamentos bem próximos uns dos outros. Para os problemas-teste com 11 tarefas, o MPLIM-BG permitiu resolver 75,00% dos casos, contra 66,67% do MPLIM-G, e proporcionou $\overline{gap}$ inferior ao propiciado pelo MPLIM-G. Para estes problemas-teste, o tempo médio demandado pelo CPLEX com o MPLIM-G foi sensivelmente inferior ao despendido com o MPLIM-BG. Para os problemas-teste com 12 tarefas, ambos os modelos proporcionaram ao otimizador resolver 50,00% dos problemas, com $\overline{gap}$'s próximos de 31% e tempos médios próximos de 1960 segundos.

5.2.2 Segunda Bateria de Testes - BDNS

A segunda bateria de testes foi realizada com os problemas-teste da base de dados BDNS. Os problemas-teste dessa base de dados também não satisfazem, necessariamente, a desigualdade triangular para os tempos de *setup*. Do mesmo modo, apenas os modelos MPLIM-G e MPLIM-BG foram utilizados nesta bateria, já que o MPLIM-IT não permite resolver problemas em que a desigualdade triangular não é satisfeita.

A Tabela 5.2 resume os resultados encontrados. Nela, a primeira coluna indica o número de tarefas em cada problema-teste dos conjuntos. Além disso, para cada conjunto de problemas, as colunas "Soluções Ótimas" mostram (em porcentagem) em quantos problemas o CPLEX encontrou uma solução ótima com os respectivos modelos, as colunas "$\overline{gap}$" mostram as médias dos *gap*'s com os respectivos modelos e as colunas "Tempo Médio" mostram as médias dos tempos (em segundos) demandados por tal otimizador na resolução dos problemas-teste com os respectivos modelos.

Tabela 5.2: Comparação MPLIM-G × MPLIM-BG aplicados à BDNS.

# Tarefas	MPLIM-G			MPLIM-BG		
	Soluções Ótimas (%)	$\overline{gap}$ (%)	**Tempo Médio (s)**	**Soluções Ótimas** (%)	$\overline{gap}$ (%)	**Tempo Médio (s)**
06	100,00	0,00	0,10	100,00	0,00	0,09
07	100,00	0,00	0,76	100,00	0,00	0,76
08	100,00	0,00	5,89	100,00	0,00	5,96
09	100,00	0,00	92,46	100,00	0,00	83,15
10	93,75	2,40	823,97	93,75	2,55	832,44
11	43,75	31,30	2275,84	50,00	27,53	2265,33
12	25,00	54,92	2887,26	25,00	54,53	2857,60
Média	80,36	12,66	869,47	81,25	12,09	863,62

Como pode ser observado na Tabela 5.2, os modelos MPLIM-G e MPLIM-BG também propiciaram ao CPLEX comportamentos semelhantes para a base de dados BDNS. Ambos os modelos permitiram resolver na otimalidade todos os problemas-teste com até 9 tarefas e, aproximadamente, a mesma proporção dos demais conjuntos de problemas. Os $\overline{gap}$'s e os tempos médios também foram bem próximos. Essencialmente, a diferença mais sensível ocorreu com os problemas-teste com 11 tarefas. Para este conjunto, o CPLEX conseguiu resolver 50,00% dos problemas por meio do MPLIM-BG, enquanto o MPLIM-G proporcionou ao otimizador resolver apenas 43,75% dos mesmos. Os tempos médios demandados com os dois modelos foram bem próximos uns dos outros, mas o $\overline{gap}$ com o MPLIM-BG foi um pouco menor que o $\overline{gap}$ com o MPLIM-G, 27,53% contra 31,30%.

5.2.3 Terceira Bateria de Testes - BDDT

Na terceira bateria de testes utilizou-se os problemas da base de dados BDDT. Como descrito anteriormente, os problemas-teste dessa base de dados satisfazem a desigualdade triangular. Sendo assim, os três modelos de programação matemática apresentados na Seção 3 foram utilizados nesta bateria.

Os resultados encontrados para o MPLIM-G, para o MPLIM-BG e para o MPLIM-IT são resumidos na Tabela 5.3, na qual a primeira coluna indica o número de tarefas dos problemas-teste de cada conjunto. Para cada conjunto de problemas, as colunas "Soluções Ótimas" mostram (em porcentagem) em quantos problemas o CPLEX encontrou uma solução ótima com os respectivos modelos, as colunas "$\overline{gap}$" mostram as médias dos *gap*'s com os respectivos modelos e as colunas "Tempo Médio" mostram as médias dos tempos (em segundos) demandados por tal otimizador na resolução dos problemas-teste com os respectivos modelos.

Tabela 5.3: Comparação MPLIM-G × MPLIM-BG × MPLIM-IT aplicados à BDDT.

# Tarefas	MPLIM-G			MPLIM-BG			MPLIM-IT		
	Soluções Ótimas (%)	$\overline{gap}$ (%)	**Tempo Médio (s)**	**Soluções Ótimas** (%)	$\overline{gap}$ (%)	**Tempo Médio (s)**	**Soluções Ótimas** (%)	$\overline{gap}$ (%)	**Tempo Médio (s)**
06	100,00	0,00	0,08	100,00	0,00	0,07	100,00	0,00	3,10
07	100,00	0,00	0,88	100,00	0,00	0,87	100,00	0,00	7,15
08	100,00	0,00	8,07	100,00	0,00	9,19	100,00	0,00	11,47
09	100,00	0,00	102,79	100,00	0,00	108,71	100,00	0,00	24,21
10	93,75	2,88	1049,76	93,75	2,59	1019,47	100,00	0,00	57,82
11	43,75	32,42	2377,33	43,75	33,80	2327,45	100,00	0,00	128,14
12	12,50	59,79	3322,83	6,25	60,41	3377,30	100,00	0,00	278,23
13	18,75	66,27	2988,67	18,75	66,40	2999,26	87,50	0,58[1]	1246,11
14	12,50	68,50	3198,65	12,50	69,40	3168,35	37,50	5,36[2]	2758,40
Média	64,58	25,54	1449,90	63,89	25,85	1445,63	91,67	0,66	501,63

[1] *gap* médio relativo aos 93,75% problemas-teste em que uma solução viável foi encontrada.
[2] *gap* médio relativo aos 50,00% problemas-teste em que uma solução viável foi encontrada.

Pela Tabela 5.3, observa-se que, novamente, o MPLIM-G e o MPLIM-BG apresentaram comportamentos bem semelhantes. Usando-se esses dois modelos, o CPLEX não foi capaz de resolver na otimalidade todos os problemas-teste com mais de 9 tarefas, para os quais o $\overline{gap}$ variou de 2,59% à 69,40%. Por outro lado, com o modelo MPLIM-IT, o CPLEX conseguiu resolver na otimalidade todos os problemas-teste com até 12 tarefas. Esta última formulação também exigiu, na maioria dos casos, um menor tempo médio computacional para se alcançar a otimalidade. De fato, com exceção dos três primeiros conjuntos de problemas-teste, a formulação MPLIM-IT demandou até 18 vezes, aproximadamente, menos tempo que as demais formulações.

Ainda na Tabela 5.3, nota-se que o CPLEX aplicado à formulação MPLIM-IT não foi capaz de encontrar, em uma hora de processamento, uma solução viável em 6,25% dos problemas-teste com 13 tarefas, bem como em 50,00% dos problemas-teste com 14 tarefas. Contudo, para os casos em que essa formulação conseguiu gerar uma solução viável, os $\overline{gap}$'s foram bem baixos (0,58% em problemas com 13 tarefas e 5,36% nos de 14 tarefas), se comparados com aqueles produzidos pelas formulações MPLIM-G e MPLIM-BG, que foram sempre superiores a 66%.

Os resultados apresentados na Seção 5.3.4 mostram que o fato de a formulação MPLIM-IT não permitir ao CPLEX encontrar uma solução viável em alguns problemas-teste com

mais de 12 tarefas, dentro do limite de tempo utilizado, está associado, também, aos valores de h_{inf} e h_{sup} utilizados.

5.3 Resultados Obtidos com os Algoritmos GPV e GPV-REB

Os algoritmos heurísticos propostos (GPV e GPV-REB), bem como os procedimentos utilizados para a definição dos seus parâmetros, foram implementados em linguagem C, usando-se o compilador Borland C++ Builder 5.0. Foram realizados experimentos com todos os problemas-teste das três bases de dados apresentadas na Seção 5.1, totalizando três baterias de testes.

Para auxiliar na apresentação e comparação dos resultados, foram utilizadas duas medidas de desempenho, dadas pelas equações (5.4) e (5.5). Nestas equações, para um dado problema-teste i, f_i^* é o valor da função objetivo da melhor solução conhecida para o problema e $f_i^{X^\star}$ e $\overline{f}_i^X$ são, respectivamente, o valor da função objetivo da melhor solução encontrada e a média dos valores da função objetivo aplicada nas soluções encontradas durante todas as execuções do algoritmo X para o problema-teste i.

$$dev_i^{best} = \frac{f_i^{X^\star} - f_i^\star}{f_i^\star} \tag{5.4}$$

$$dev_i^{avg} = \frac{\overline{f}_i^X - f_i^\star}{f_i^\star} \tag{5.5}$$

A primeira equação calcula o desvio do valor da função objetivo da melhor solução encontrada pelo algoritmo X em torno da função objetivo da melhor solução conhecida para o respectivo problema-teste. Já a segunda equação calcula a variabilidade média dos valores da função objetivo das soluções encontradas pelo algoritmo X em torno da melhor solução conhecida para o respectivo problema.

A seguir são apresentados os procedimentos utilizados para definir cada parâmetro do GPV (foram utilizados os mesmos parâmetros no GPV-REB) e os resultados obtidos em cada bateria de testes.

5.3.1 Definição dos Parâmetros

Cada parâmetro do algoritmo GPV foi definido por meio de experimentos em um problema-teste de porte médio (30 tarefas). Definido um parâmetro, este era fixado nos procedimentos de definição dos demais parâmetros.

O primeiro parâmetro a ser definido foi o parâmetro γ da fase de construção de uma solução (ver Seção 4.5). Para isto, dado um valor para γ, foi construída uma solução com cada uma das seis funções adaptativas gulosas definidas na Seção 4.5 e a melhor destas soluções era retornada. Como o objetivo era determinar o valor de γ que proporciona construir soluções de melhor qualidade, não foi utilizado o POP durante as construções das soluções, ou seja, as soluções parciais não foram submetidas à uma busca local. Este procedimento foi realizado 20 vezes para cada um dos seguintes valores para γ: 0; 0,02;

0,04; 0,06; 0,08; 0,10; 0,12; 0,14; 0,16; 0,18; 0,20; 0,22; 0,24; 0,26; 0,28 e 0,30. Os resultados encontrados estão sintetizados no gráfico da Figura 5.1, onde é mostrado o menor valor e a média dos valores retornados para a função objetivo (*fo*) das soluções produzidas nas 20 execuções dos respectivos valores para γ.

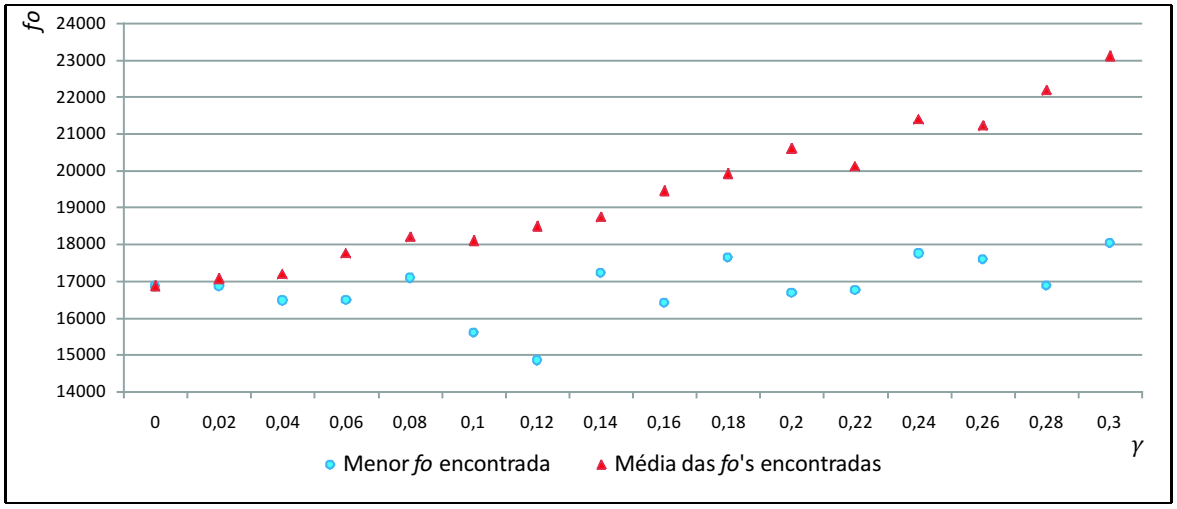

Figura 5.1: Resultados dos testes para definição de γ.

Como pode ser observado no gráfico da Figura 5.1, o menor valor para *fo* foi encontrado para γ igual a 0,12. Já a menor média para *fo* foi encontrada para γ igual a 0. Desde modo, definiu-se γ igual a 0 na primeira vez em que cada uma das seis funções adaptativas gulosas é utilizada e γ igual a 0,12 nas demais.

O segundo parâmetro a ser definido foi o parâmetro *MDRmax* da descida randômica (MDR) utilizada no POP (ver Seção 4.5). Para isto, dado um valor para *MDRmax*, foi construída uma solução com cada uma das seis funções adaptativas gulosas definidas na Seção 4.5 e a melhor destas soluções era retornada. Como o objetivo era determinar o valor de *MDRmax* que proporciona encontrar soluções de melhor qualidade a partir de uma mesma solução inicial, o valor de γ foi fixado em 0. Deste modo, a ordem de inclusão de uma tarefa na solução parcial é sempre a mesma para uma dada função adaptativa gulosa. Este procedimento foi realizado 20 vezes para cada valor de *MDRmax*. Os valores de *MDRmax* testados são da forma $k \times n$, onde n representa o número de tarefas a serem sequenciadas e $k \in \{1,\ 2,\ 3, \ldots,\ 15\}$. Os resultados encontrados estão apresentados no gráfico da Figura 5.2, onde é mostrado o menor valor e a média dos valores retornados para a função objetivo (*fo*) das soluções encontradas nas 20 execuções dos respectivos valores de k.

Como pode ser observado no gráfico da Figura 5.2, o menor valor para *fo* foi encontrado para k igual a 7, ou seja, para *MDRmax* igual a $7 \times n$. Deste modo, definiu-se *MDRmax* igual a $7 \times n$. Devido ao tempo computacional despendido nos problemas com 75 ou mais tarefas ($n \geq 75$), utilizou-se *MDRmax* igual a $5 \times n$ para estes problemas.

Definidos os parâmetros da fase de construção de uma solução, o próximo parâmetro a ser definido foi o parâmetro *VNDmax* do VND_1 (ver Seção 4.6). Para isto, dado um valor para *VNDmax*, foi construída uma solução com cada uma das seis funções adaptativas gulosas definidas na Seção 4.5 e, a seguir, cada solução construída foi submetida ao VND_1.

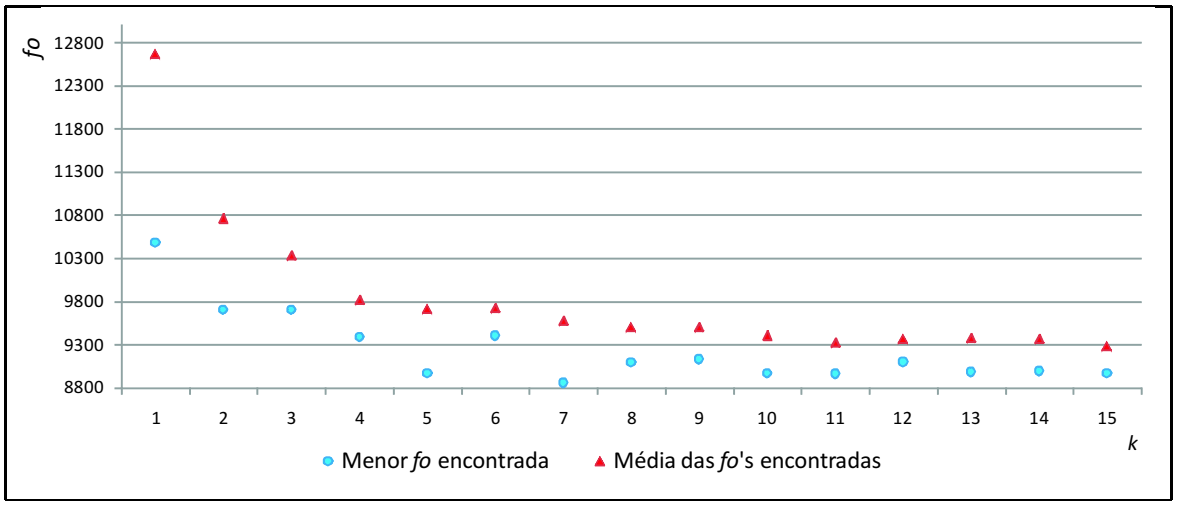

Figura 5.2: Resultados dos testes para definição de *MDRmax*.

A melhor solução encontrada era retornada. Para construir as soluções, γ foi fixado em 0 e o *MDRmax* foi $7 \times n$. Este procedimento foi realizado 20 vezes para cada valor de *VNDmax*. Os valores de *VNDmax* testados também são da forma $k \times n$, onde n representa o número de tarefas a serem sequenciadas e $k \in \{1,\ 2,\ 3, \ldots,\ 15\}$. Os resultados encontrados estão resumidos no gráfico da Figura 5.3, onde é mostrado o menor valor e a média dos valores retornados para a função objetivo (fo) das soluções encontradas nas 20 execuções dos respectivos valores de k.

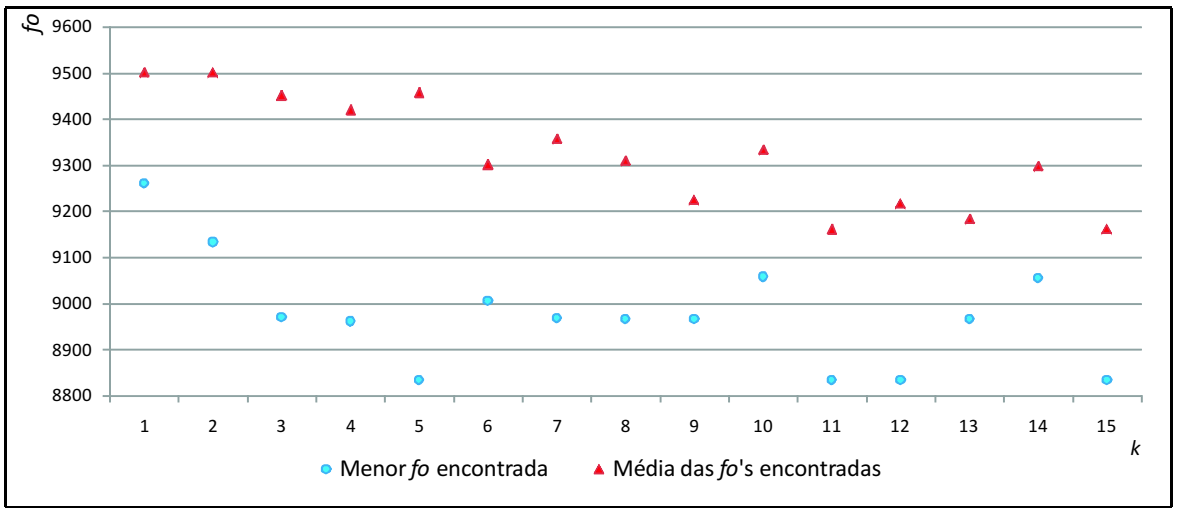

Figura 5.3: Resultados dos testes para definição de *VNDmax*.

Conforme o gráfico da Figura 5.3, o menor valor para fo foi encontrado para k igual a 5, 11, 12 e 15, ou seja, para *VNDmax* igual a $5 \times n$, $11 \times n$, $12 \times n$ e $15 \times n$. Como a menor média para a fo foi encontrada para *VNDmax* igual a $11 \times n$, definiu-se *VNDmax* igual a $11 \times n$. Devido ao tempo computacional despendido para problemas com 75 ou mais tarefas ($n \geq 75$), nestes problemas utilizou-se *VNDmax* igual a $5 \times n$.

Finalmente, o último parâmetro a ser definido foi o parâmetro *GRASPmax*[1] da primeira fase da metodologia proposta (ver Algoritmo 10). Para isto, dado um valor para *GRASPmax*, apenas a primeira fase do algoritmo GPV era aplicada ao problema e a melhor solução encontrada era retornada. Para construir as soluções, utilizou-se γ igual a 0 na primeira vez em que cada uma das seis funções adaptativas gulosas era utilizada e γ igual a 0,12 nas demais. *MDRmax* e *VNDmax* foram fixados em $7 \times n$ e $11 \times n$, respectivamente. Este procedimento foi realizado 20 vezes para cada valor de *GRASPmax*. Os valores de *GRASPmax* testados foram: 6, 12, 18, 24 e 30. Foram testados múltiplos de 6 para *GRASPmax* por terem sido utilizadas seis funções adaptativas gulosas distintas para construir soluções. Os resultados encontrados estão sintetizados no gráfico da Figura 5.4, onde é mostrado o menor valor e a média dos valores retornados para a função objetivo (*fo*) das soluções encontradas durante as 20 execuções dos respectivos valores para *GRASPmax*.

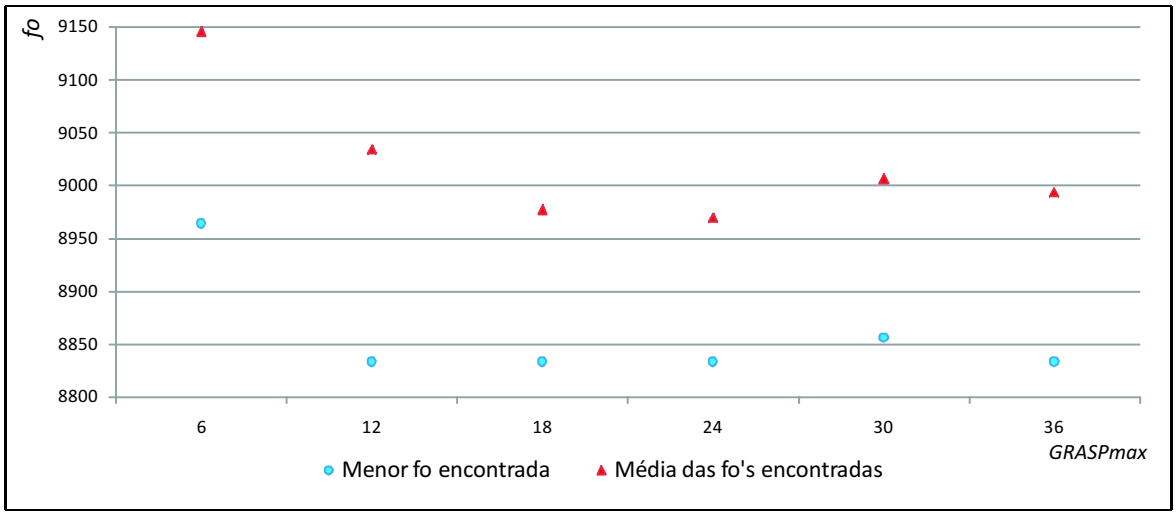

Figura 5.4: Resultados dos testes para definição de *GRASPmax*.

De acordo o gráfico da Figura 5.4, o menor valor para *fo* foi encontrado para *GRASPmax* igual a 12, 18, 24 e 36. Como a menor média para a *fo* foi encontrada para *GRASPmax* igual a 24, definiu-se *GRASPmax* igual a 24. Devido ao tempo computacional despendido nos problemas com 75 ou mais tarefas ($n \geq 75$), utilizou-se *GRASPmax* igual a 12 para estes problemas.

Os parâmetros utilizados estão resumidos na Tabela 5.4, onde n representa o número de tarefas a serem sequenciadas.

5.3.2 Primeira Bateria de Testes - BDS

A primeira bateria de experimentos foi realizada com os problemas-teste da base de dados BDS. Os problemas-teste de tal base de dados possuem os tempos de *setup* simétricos.

[1]É importante lembrar que esta fase é baseada no procedimento GRASP, mas, diferentemente do GRASP tradicional, utiliza-se *GRASPmax* iterações sem melhora na solução corrente e não simplesmente *GRASPmax* iterações.

Tabela 5.4: Parâmetros utilizados.

Parâmetros	**Valores**
γ	0 e 0,12
$MDRmax$	$7 \times n$, se $n \leq 50$ e $5 \times n$, se $n \geq 75$
$VNDmax$	$11 \times n$, se $n \leq 50$ e $5 \times n$, se $n \geq 75$
$GRASPmax$	24, se $n \leq 50$ e 12, se $n \geq 75$

Cada problema-teste foi resolvido 30 vezes com cada algoritmo proposto, exceto aqueles com 75 tarefas, os quais foram resolvidos apenas 10 vezes em virtude do maior tempo computacional demandado. Os resultados obtidos pelo algoritmo GPV foram comparados com os alcançados em Gomes Júnior (2007), Penna (2009) e Ribeiro (2009), visto que estes autores também realizaram experimentos computacionais com esta base de dados.

Um resumo dos resultados obtidos pelo algoritmo GPV e pelos algoritmos da literatura que também fizeram experimentos com a base de dados em questão é apresentado na Tabela 5.5. Nesta tabela, a primeira coluna indica o número de tarefas nos problemas-teste de cada grupo. Nas colunas "Gomes Júnior (2007)", "Penna (2009)", "Ribeiro (2009)" e "GPV" são apresentados os resultados obtidos pelos respectivos algoritmos. Para um grupo de problemas-teste com i tarefas, as colunas "Tempo" mostram as médias dos tempos demandados (em segundos) pelos respectivos algoritmos, as colunas "$\overline{dev}^{best}$" apresentam as médias dos dev_i^{best}'s (em porcentagem) obtidos pelos respectivos algoritmos aplicados nos problemas deste conjunto e as colunas "$\overline{dev}^{avg}$" apresentam as médias dos dev_i^{avg}'s obtidos pelos respectivos algoritmos aplicados nos problemas-teste do conjunto.

Pela Tabela 5.5, observa-se que o algoritmo GPV e o algoritmo proposto por Gomes Júnior (2007) foram capazes de encontrar as melhores soluções conhecidas em todos os problemas-teste com até 25 tarefas. Além disso, para estes mesmos problemas, os $\overline{dev}^{avg}$'s obtidos pelo GPV foram sempre menores que os obtidos por tal autor, sendo o maior $\overline{dev}^{avg}$ apresentado pelo GPV igual a 1,53% e o maior obtido pelo algoritmo de Gomes Júnior (2007) igual a 2,32%. Já para os problemas-teste com mais de 25 tarefas, os $\overline{dev}^{best}$'s obtidos pelos dois algoritmos alternaram os menores valores e os $\overline{dev}^{avg}$'s apresentados pelo algoritmo de Gomes Júnior (2007) foram sempre superiores que os respectivos $\overline{dev}^{avg}$'s apresentados pelo GPV, sendo que, para os problemas com 75 tarefas, o $\overline{dev}^{avg}$ obtido pelo GPV foi de 13,49%, contra os 19,22% obtido pelo algoritmo deste autor. Com relação aos tempos computacionais, não é possível uma comparação justa, uma vez que as máquinas usadas foram diferentes. A esse respeito, observa-se, apenas, que para os problemas com mais de 30 tarefas, os tempos médios demandados pelo algoritmo de Gomes Júnior (2007) foram de 30% a 74% maiores que os demandados pelo GPV.

Ainda na Tabela 5.5 é possível observar que os algoritmos propostos por Penna (2009) e Ribeiro (2009) obtiveram os menores valores para $\overline{dev}^{best}$ e $\overline{dev}^{avg}$ na maioria dos casos. Porém, para os problemas com mais de 20 tarefas, número de tarefas a partir do qual o tempo demandado passa a ser significativo, os dois algoritmos demandaram tempos

Tabela 5.5: Comparação GPV × Gomes Júnior (2007) × Penna (2009) × Ribeiro (2009) aplicados à BDS.

♯	**Gomes Júnior (2007)**			**Penna (2009)**			**Ribeiro (2009)**			**GPV**		
Tarefas	$\overline{dev}^{best}$ (%)	$\overline{dev}^{avg}$ (%)	**Tempo**[1] (s)	$\overline{dev}^{best}$ (%)	$\overline{dev}^{avg}$ (%)	**Tempo**[2] (s)	$\overline{dev}^{best}$ (%)	$\overline{dev}^{avg}$ (%)	**Tempo**[3] (s)	$\overline{dev}^{best}$ (%)	$\overline{dev}^{avg}$ (%)	**Tempo**[4] (s)
08	0,00	0,03	0,04	0,00	0,00	0,06	0,00	0,00	0,94	0,00	0,00	0,11
09	0,00	0,06	0,07	0,00	0,00	0,09	0,00	0,15	1,26	0,00	0,00	0,17
10	0,00	0,02	0,11	0,00	0,00	0,15	0,00	0,24	1,60	0,00	0,00	0,24
11	0,00	0,12	0,20	0,00	0,00	0,25	0,00	0,03	2,21	0,00	0,06	0,46
12	0,00	0,21	0,29	0,00	0,00	0,37	0,00	0,07	2,81	0,00	0,02	0,74
15	0,00	1,47	0,94	0,00	0,47	1,13	0,00	0,76	6,02	0,00	0,51	1,47
20	0,00	1,65	4,35	0,00	0,64	4,93	0,00	0,73	20,60	0,00	1,49	5,19
25	0,00	2,32	13,29	0,00	1,09	14,90	0,00	1,02	45,72	0,00	1,53	14,22
30	0,20	3,34	40,07	0,00	1,68	39,93	0,00	1,60	112,06	0,04	2,41	30,84
40	0,44	4,66	155,79	0,08	3,32	190,61	0,00	2,41	335,88	1,10	3,89	111,52
50	1,37	7,29	492,28	0,32	4,95	630,77	0,00	4,08	896,10	1,65	6,48	309,03
75	7,89	19,22	1368,08	0,82	7,46	6308,74	0,99	7,61	2005,05	6,08	13,49	787,90
Média	0,82	3,37	172,96	0,10	1,63	599,33	0,08	1,56	285,85	0,74	2,49	105,16

[1]Testes realizados em um PC Athlon XP 64 Bits 3000+ (aproximadamente 2 GHz), com 1 GB de RAM.

[2]Testes realizados em um PC Intel Core 2 Quad (Q6600) 2,40 GHz, com 4 GB de RAM.

[3]Testes realizados em um PC Pentium Core 2 Duo 2,1 GHz, com 4 GB de memória RAM.

[4]Testes realizados em um PC Intel Core 2 Duo (E8400) 2.99 GHz, com 2 GB de memória RAM.

médios superiores aos demandados pelo GPV nos respectivos conjuntos de problemas. Apesar de terem sido utilizadas máquinas diferentes, os tempos médios demandados pelo algoritmo de Penna (2009) e Ribeiro (2009) em tais conjuntos de problemas chegaram a ser aproximadamente 800% e 360%, respectivamente, superiores aos demandados pelo algoritmo GPV.

Os resultados encontrados nos testes computacionais realizados com o algoritmo GPV-REB estão sintetizados na Tabela 5.6. Na primeira coluna é indicado o número de tarefas nos problema-teste de cada conjunto. Nas colunas "GPV" e "GPV-REB" são apresentados os resultados obtidos pelos respectivos algoritmos. Para um grupo de problemas-teste com i tarefas, as colunas "Tempo", "$\overline{dev}^{best}$" e "$\overline{dev}^{avg}$" são tais como na Tabela 5.5.

Tabela 5.6: Comparação GPV × GPV-REB aplicados à BDS.

♯	**GPV**			**GPV-REB**		
Tarefas	$\overline{\boldsymbol{dev}}^{\boldsymbol{best}}$ (%)	$\overline{\boldsymbol{dev}}^{\boldsymbol{avg}}$ (%)	**Tempo** (s)	$\overline{\boldsymbol{dev}}^{\boldsymbol{best}}$ (%)	$\overline{\boldsymbol{dev}}^{\boldsymbol{avg}}$ (%)	**Tempo** (s)
08	0,00	0,00	0,11	0,23	0,23	0,06
09	0,00	0,00	0,17	0,00	0,00	0,13
10	0,00	0,00	0,24	0,00	0,04	0,13
11	0,00	0,06	0,46	0,15	0,19	0,37
12	0,00	0,02	0,74	0,00	0,98	0,52
15	0,00	0,51	1,47	1,48	2,53	0,90
20	0,00	1,49	5,19	0,04	3,77	3,15
25	0,00	1,53	14,22	1,57	3,86	6,96
30	0,04	2,41	30,84	0,72	4,55	19,26
40	1,10	3,89	111,52	1,77	6,73	60,36
50	1,65	6,48	309,03	2,13	8,70	151,74
75	6,08	13,49	787,90	8,63	16,34	467,71
Média	0,74	2,49	105,16	1,39	3,99	59,27

Conforme pode ser observado na Tabela 5.6, para os conjuntos de problemas-teste com até 50 tarefas, o algoritmo GPV-REB obteve valores de $\overline{dev}^{best}$ menores ou iguais a 2,13%. De fato, em dois destes conjuntos o $\overline{dev}^{best}$ foi próximo de 1,60% e nos demais, menores que 1,00%. Para os problemas-teste com 75 tarefas o $\overline{dev}^{best}$ foi de 8,63%, valor próximo daquele obtido com o algoritmo GPV. Já os $\overline{dev}^{avg}$'s obtidos por meio do GPV-REB foram sempre superiores aos apresentados pelo GPV nos conjuntos correspondentes. No entanto, os valores dos $\overline{dev}^{avg}$'s obtido pelo GPV-REB aproximam-se dos obtidos com o GPV à medida em que consideram-se conjuntos de problemas com mais tarefas. Finalmente, para os problemas em que o tempo demandado é mais significativo, isto é, problemas com mais de 20 tarefas, os tempos demandados pelo GPV foram de 60% a 200%, aproximadamente, superiores aos demandados pelo GPV-REB nos conjuntos de problemas correspondentes.

5.3.3 Segunda Bateria de Testes - BDNS

Na segunda bateria de testes utilizou-se os problemas da base de dados BDNS. Os problemas-teste dessa base de dados não possuem, necessariamente, os tempos de *setup* simétricos e satisfazendo a desigualdade triangular, sendo assim, mais genéricos. Assim como na primeira bateria de testes, cada problema foi resolvido 30 vezes, exceto aqueles com mais de 50 tarefas, os quais foram resolvidos apenas 10 vezes devido ao maior tempo computacional demandado. Os resultados obtidos pelos algoritmos GPV e GPV-REB foram comparados com os resultados do algoritmo proposto por Ribeiro (2009), dado que este é o único autor da literatura que realizou experimentos com esta base de dados.

Uma síntese dos resultados encontrados pelos algoritmos GPV e GPV-REB e pelo algoritmo de Ribeiro (2009) é apresentada na Tabela 5.7. Nela, a primeira coluna indica o número de tarefas dos problemas de cada grupo de problemas-teste. Nas colunas "Ribeiro (2009)", "GPV" e "GPV-REB" são apresentados os resultados obtidos pelos respectivos algoritmos. Para um grupo de problemas-teste com i tarefas, as colunas "Tempo" mostram as médias dos tempos demandados (em segundos) pelos respectivos algoritmos, as colunas "$\overline{dev}^{best}$" apresentam as médias dos dev_i^{best}'s (em porcentagem) obtidos pelos respectivos algoritmos aplicados nos problemas deste grupo e as colunas "$\overline{dev}^{avg}$" apresentam as médias dos dev_i^{avg}'s obtidos pelos respectivos algoritmos aplicados nos problemas do grupo. Os resultados de Ribeiro (2009) relativos aos problemas-teste com 150 tarefas não são apresentados porque tal autor não resolveu tais problemas.

Tabela 5.7: Comparação GPV × GPV-REB × Ribeiro (2009) aplicados à BDNS.

♯	**Ribeiro (2009)**			**GPV**			**GPV-REB**		
Tarefas	$\overline{dev}^{best}$ (%)	$\overline{dev}^{avg}$ (%)	**Tempo**[1] (s)	$\overline{dev}^{best}$ (%)	$\overline{dev}^{avg}$ (%)	**Tempo**[2] (s)	$\overline{dev}^{best}$ (%)	$\overline{dev}^{avg}$ (%)	**Tempo**[2] (s)
006	0,00	0,00	0,76	0,00	0,00	0,04	0,00	0,00	0,03
007	0,00	0,00	0,76	0,00	0,00	0,07	0,00	0,00	0,03
008	0,00	0,00	1,03	0,00	0,00	0,30	0,00	0,76	0,25
009	0,00	0,00	1,35	0,00	0,00	0,37	0,01	0,18	0,29
010	0,00	0,00	1,61	0,00	0,20	0,45	0,47	0,86	0,34
011	0,00	0,00	2,20	0,00	0,01	0,40	0,11	0,16	0,40
012	0,00	0,00	3,32	0,00	0,00	0,52	0,00	0,43	0,42
015	0,00	0,01	7,49	0,00	0,01	1,26	0,00	0,54	0,73
020	0,00	0,30	23,88	0,00	0,10	4,52	0,49	1,83	2,85
030	0,19	1,40	172,51	0,01	0,58	36,06	0,51	1,99	18,17
040	0,18	1,17	801,67	0,06	0,79	132,94	0,36	1,51	67,04
050	0,30	1,45	1575,11	0,06	1,28	440,49	0,65	2,57	215,60
075	5,90	5,90	4978,02	3,10	5,64	1187,76	4,14	6,66	636,86
100	2,36	2,62	22299,79	0,21	2,32	4612,79	0,57	3,79	3015,16
150	-	-	-	0,14	2,53	49524,76	0,83	2,62	32833,61

[1]Testes realizados em um PC Pentium Core 2 Duo 2,1 GHz, com 4 GB de memória RAM.

[2]Testes realizados em um PC Intel Core 2 Duo (E8400) 2.99 GHz, com 2 GB de memória RAM.

De acordo com a Tabela 5.7, o GPV obteve valores de $\overline{dev}^{best}$ sempre menores ou iguais

que os respectivos valores obtidos pelo algoritmo de Ribeiro (2009). Isso significa que as soluções produzidas pelo GPV foram, em média, melhores ou iguais às produzidas pelo algoritmo de tal autor. Ambos os algoritmos encontraram $\overline{dev}^{avg}$'s próximos de zero nos conjuntos de problemas-teste com até 20 tarefas. Já nos conjuntos de problemas com mais de 20 tarefas, os valores dos $\overline{dev}^{avg}$'s obtidos pelo GPV foram relativamente inferiores aos respectivos valores encontrados pelo algoritmo de Ribeiro (2009). Ademais, os tempos médios demandados pelo algoritmo de Ribeiro (2009) para resolver os problemas de cada conjunto de problemas-teste foram sempre significativamente maiores que os demandados pelo GPV nos problemas dos respectivos conjuntos. Para o conjunto de problemas-teste com 100 tarefas, por exemplo, o tempo médio demandado pelo algoritmo proposto por Ribeiro (2009) foi aproximadamente 4,8 vezes superior ao demandado pelo GPV.

Na Tabela 5.7 também é possível observar que o algoritmo GPV obteve valores de $\overline{dev}^{best}$ menores que 1% para todos os conjuntos de problemas-teste, exceto para o conjunto de problemas com 75 tarefas. Mesmo assim, para este último conjunto, o $\overline{dev}^{best}$ foi menor que o obtido pelo algoritmo de Ribeiro (2009). Os $\overline{dev}^{avg}$'s foram relativamente baixos, sendo o maior valor igual a 6,66% e, para os problemas com 150 tarefas, próximo do valor obtido pelo GPV. Os tempos médios demandados também foram inferiores os demandados pelos outros dois algoritmos. Se comparado aos tempos médios demandados pelo GPV, os demandados pelo GPV-REB foram de 34% a 50% menores.

5.3.4 Terceira Bateria de Testes - BDDT

A terceira bateria de experimentos foi realizada com os problemas-teste da base de dados BDDT. Os tempos de processamento e de *setup* dos problemas-teste de tal base de dados satisfazem a desigualdade triangular. Cada problema-teste foi resolvido 30 vezes. Novamente os resultados obtidos pelos algoritmos GPV e GPV-REB foram comparados com os resultados de Ribeiro (2009), pois este último é o único autor da literatura que realizou experimentos com a base de dados em questão.

Os resultados obtidos pelos algoritmos GPV e GPV-REB e por Ribeiro (2009) estão resumidos na Tabela 5.8, onde a primeira coluna indica o número de tarefas nos problemas-teste de cada conjunto de problemas. Nas colunas "Ribeiro (2009)", "GPV" e "GPV-REB" são apresentados os resultados obtidos pelos respectivos algoritmos. Para um grupo de problemas-teste com i tarefas, as colunas "Tempo" mostram as médias dos tempos demandados (em segundos) pelos respectivos algoritmos, as colunas "$\overline{dev}^{best}$" apresentam as médias dos dev_i^{best}'s (em porcentagem) obtidos pelos respectivos algoritmos aplicados nos problemas deste grupo e as colunas "$\overline{dev}^{avg}$" apresentam as médias dos dev_i^{avg}'s obtidos pelos respectivos algoritmos aplicados nos problemas-teste do grupo.

Pela Tabela 5.8, observa-se que o algoritmo GPV, bem como o algoritmo proposto por Ribeiro (2009), foi capaz de encontrar todas as respectivas melhores soluções conhecidas para os problemas-teste da base de dados BDDT. Apesar de o mesmo não ter ocorrido com o algoritmo GPV-REB, o maior $\overline{dev}^{best}$ obtido com este algoritmo foi de 0,35%. Os três algoritmos obtiveram valores de $\overline{dev}^{avg}$ menores que 1,00%. Em relação aos tempos médios demandados em cada conjunto de problemas-teste, novamente o algoritmo de Ribeiro (2009) demandou tempos médios maiores que os respectivos tempos médios demandados pelo algoritmo GPV, que por sua vez demandou tempos médios maiores que os despen-

Tabela 5.8: Comparação GPV × GPV-REB × Ribeiro (2009) aplicados à BDDT.

♯	**Ribeiro (2009)**			**GPV**			**GPV-REB**		
Tarefas	$\overline{\boldsymbol{dev}}^{best}$ (%)	$\overline{\boldsymbol{dev}}^{avg}$ (%)	**Tempo**[1] (s)	$\overline{\boldsymbol{dev}}^{best}$ (%)	$\overline{\boldsymbol{dev}}^{avg}$ (%)	**Tempo**[2] (s)	$\overline{\boldsymbol{dev}}^{best}$ (%)	$\overline{\boldsymbol{dev}}^{avg}$ (%)	**Tempo**[2] (s)
06	0,00	0,00	0,67	0,00	0,00	0,04	0,00	0,00	0,02
07	0,00	0,00	0,86	0,00	0,00	0,17	0,00	0,06	0,04
08	0,00	0,00	1,13	0,00	0,00	0,29	0,00	0,27	0,06
09	0,00	0,00	1,47	0,00	0,00	0,34	0,00	0,02	0,09
10	0,00	0,00	1,78	0,00	0,00	0,41	0,00	0,03	0,15
11	0,00	0,00	2,43	0,00	0,04	0,32	0,00	0,15	0,19
12	0,00	0,00	3,88	0,00	0,00	0,49	0,01	0,09	0,31
13	0,00	0,00	4,87	0,00	0,01	0,69	0,00	0,25	0,44
14	0,00	0,00	5,75	0,00	0,04	0,91	0,00	0,11	0,56
15	0,00	0,02	8,50	0,00	0,04	1,25	0,00	0,19	0,71
16	0,00	0,06	11,64	0,00	0,14	2,36	0,04	0,61	1,20
17	0,00	0,04	1,47	0,00	0,04	2,34	0,09	0,36	1,40
18	0,00	0,42	15,23	0,00	0,05	3,03	0,05	0,37	1,95
19	0,00	0,04	22,16	0,00	0,01	3,80	0,35	0,89	2,59
20	0,00	0,02	30,32	0,00	0,14	5,26	0,10	0,88	3,43
Média	0,00	0,04	7,48	0,00	0,03	1,45	0,04	0,29	0,88

[1]Testes realizados em um PC Pentium Core 2 Duo 2,1 GHz, com 4 GB de memória RAM.

[2]Testes realizados em um PC Intel Core 2 Duo (E8400) 2.99 GHz, com 2 GB de memória RAM.

didos pelo algoritmo GPV-REB. Tais tempos médios foram, no máximo, 3,43 segundos para o algoritmo GPV-REB, 5,26 segundos para o algoritmo GPV e 30,32 segundos para o algoritmo de Ribeiro (2009).

Dado um determinado problema-teste da base de dados BDDT, a melhor solução produzida para ele foi utilizada para determinar o conjunto $H = \{h_{inf},\ h_{inf}+1,\ h_{inf}+2,\ h_{sup}\}$ necessário na formulação de programação matemática MPLIM-IT (ver Seção 3.3). Os valores de h_{inf} e de h_{sup} utilizados são dados, respectivamente, pela data de início da primeira tarefa e pela data de conclusão da última tarefa na melhor sequência encontrada pelo algoritmo GPV para o problema. Este procedimento foi utilizado para determinar o conjunto H de cada problema-teste da base de dados e, em seguida, utilizou-se o CPLEX para resolver o MPLIM-IT relativo à cada problema.

Considerou-se o limite de 3600 segundos para a resolução de cada problema-teste pelo CPLEX e a qualidade das soluções retornadas foram medidas via a equação (5.3) apresentada na Seção 5.2, à página 52.

Os resultados encontrados são resumidos na Tabela 5.9. Nela, a primeira coluna indica o número de tarefas nos problemas de cada conjunto de problemas-teste. Para cada conjunto de problemas, a coluna "Soluções Ótimas" mostra (em porcentagem) em quantos problemas o CPLEX encontrou uma solução ótima com o modelo MPLIM-IT, a coluna "$\overline{gap}$" mostra as médias dos gap's e a coluna "Tempo Médio" mostra as médias dos tempos demandados (em segundos) por tal otimizador na resolução dos problemas-teste com o MPLIM-IT.

Tabela 5.9: Resultados do MPLIM-IT associado ao GPV.

# Tarefas	MPLIM-IT		
	Soluções Ótimas (%)	$\overline{gap}$ (%)	**Tempo Médio (s)**
06	100,00	0,00	1,93
07	100,00	0,00	5,06
08	100,00	0,00	10,44
09	100,00	0,00	16,99
10	100,00	0,00	41,70
11	100,00	0,00	105,65
12	100,00	0,00	297,89
13	100,00	0,00	795,70
14	87,50	0,50	944,55
15	68,75	1,61	1593,53
16	50,00	1,92[1]	2563,40
17	37,50	5,86[1]	2911,82
Média	86,98	0,82	774,05

[1] *gap* médio relativo aos 87,50% dos problemas-teste em que uma solução viável foi encontrada.

Segundo a Tabela 5.9, o modelo MPLIM-IT associado ao algoritmo GPV permitiu ao CPLEX resolver na otimalidade todos os problemas-teste com até 13 tarefas. Nos conjuntos de problemas-teste com 14 e 15 tarefas, o maior $\overline{gap}$ encontrado foi igual a 1,61%.

Ainda na Tabela 5.9, observa-se que o CPLEX aplicado à formulação MPLIM-IT associada ao algoritmo GPV não foi capaz de encontrar, em uma hora de processamento, uma solução viável em 12,50% dos conjuntos de problemas-teste com 16 e 17 tarefas. Contudo, para os casos em que essa formulação conseguiu gerar uma solução viável, os $\overline{gap}$'s também foram baixos (1,92% nos problemas com 16 tarefas e 5,86% nos de 17 tarefas).

É interessante observar que houve uma melhora significativa quanto aos tempos demandados, às dimensões dos problemas resolvidos e às qualidades das soluções obtidas com o MPLIM-IT associado ao algoritmo GPV em relação aos respectivos resultados obtidos com o MPLIM-IT sem o auxílio de tal algoritmo (ver Tabela 5.3, Seção 5.2.3). Por exemplo, com o auxílio do algoritmo GPV foi possível encontrar soluções viáveis e com baixo $\overline{gap}$ para todos os problemas-teste com até 15 tarefas, enquanto o MPLIM-IT sem o GPV não permitiu encontrar soluções viáveis para todos os problemas com 13 e 14 tarefas.

5.4 Teste de Probabilidade Empírica

Visto que é computacionalmente inviável executar os algoritmos propostos por Gomes Júnior (2007), Penna (2009) e Ribeiro (2009) na mesma máquina em que foi executado o algoritmo GPV, para todos os problemas-teste considerados, o comportamento de tais algoritmos foram comparados via um teste de probabilidade empírica proposto por Aiex

et al. (2002). Para realizar tal teste, escolheu-se um problema com 40 tarefas da base de dados BDS. Em seguida, determinou-se o valor da função objetivo da melhor solução conhecida para o problema como sendo o alvo a ser atingido (portanto, um alvo difícil). Cada um destes algoritmos foi executado 100 vezes no problema escolhido e cada execução foi interrompida somente quando o alvo foi atingido. Neste último momento, registrou-se o tempo despendido. Para cada algoritmo, os tempos iguais foram eliminados e o restante foi ordenado de forma crescente. Ao tempo da posição i, denotado por t_i, associou-se a probabilidade $p_i = \dfrac{i - 0,5}{N}$, sendo N o número de tempos não repetidos.

Os gráficos com os pontos $(t_i,\ p_i)$ relativos aos algoritmos analisados são apresentados na Figura 5.5. Segundo este gráfico, os algoritmos propostos por Gomes Júnior (2007), Penna (2009) e o GPV apresentam comportamentos semelhantes para o problema-teste em questão, com vantagem dos dois últimos, que requerem um menor tempo de processamento para obter um dado nível de probabilidade de alcançar o valor alvo. Com os algoritmos GPV e o proposto por Penna (2009), a partir de 140 segundos, aproximadamente, tem-se mais de 90% de probabilidade de atingir o alvo para o problema-teste escolhido. Já o algoritmo de Gomes Júnior (2007) possui tal probabilidade somente a partir de 190 segundos, aproximadamente. O algoritmo proposto por Ribeiro (2009) é o mais lento entre os algoritmos analisados, pois, para qualquer tempo dado, ele possui a probabilidade de alcançar o alvo menor que os demais algoritmos.

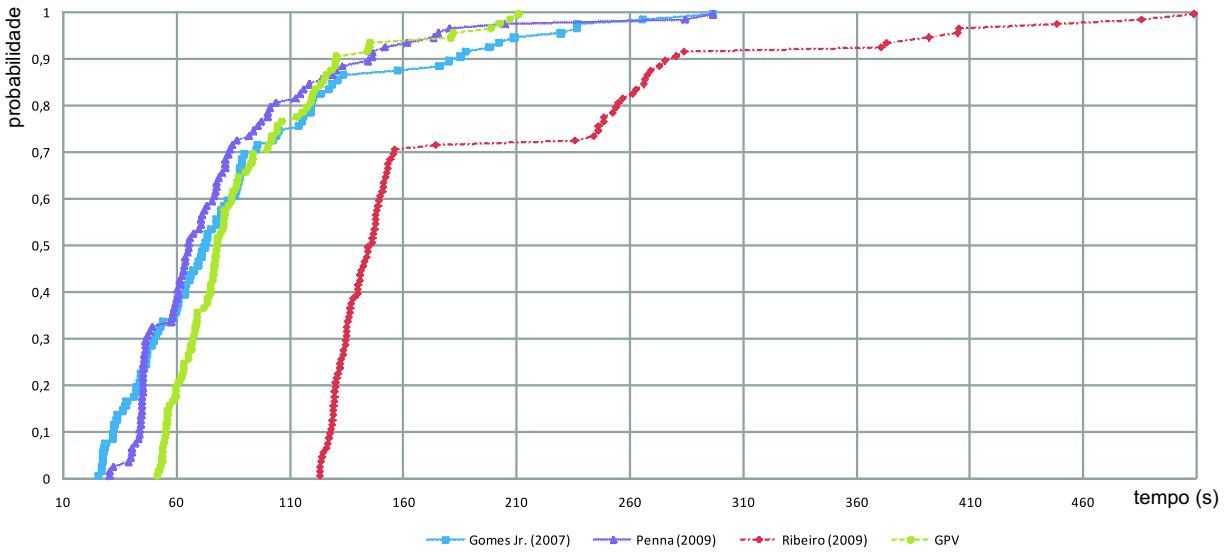

Figura 5.5: Teste de probabilidade empírica.

O teste de probabilidade empírica também foi realizado com outros dois problemas com 30 tarefas da base de dados BDS. Os resultados encontrados foram semelhantes aos apresentados na Figura 5.5.

Capítulo 6

Conclusões e Trabalhos Futuros

Este capítulo apresenta as conclusões decorrentes do presente trabalho, bem como sugestões para trabalhos futuros.

6.1 Conclusões

Este trabalho tratou o problema de sequenciamento de tarefas em uma máquina com penalidades por antecipação e atraso da produção, considerando janelas de entrega distintas e tempo de preparação da máquina dependente da sequência, denotado por PSUMAA-JE-TPD.

Foram propostas, inicialmente, duas novas formulações de programação matemática para representar o problema. A primeira delas, denotada por MPLIM-BG, consiste em um aperfeiçoamento da formulação proposta por Gomes Júnior (2007) (MPLIM-G), pois exige menos variáveis e restrições que a formulação daquele autor. A segunda formulação proposta, denotada por MPLIM-IT, é indexada no tempo.

Propôs-se, também, um algoritmo heurístico para determinar a melhor sequência de produção. Tal algoritmo, denominado GPV, é composto de duas fases. Na primeira, repetidamente, constrói-se uma solução, baseada no procedimento de construção da metaheurística GRASP e no Princípio da Otimalidade Próxima (POP), para ser refinada por uma Descida em Vizinhança Variável (*Variable Neighborhood Descent* - VND). Na segunda fase, faz-se o pós-refinamento da solução proveniente da fase anterior por meio de outro VND. Para cada sequência gerada pelo algoritmo, um procedimento de complexidade polinomial encontrado na literatura é acionado para determinar as datas ótimas de conclusão das tarefas. Com o objetivo de reduzir o custo computacional, adicionalmente é proposta uma estratégia de redução do espaço de busca. Essa estratégia foi utilizada durante a primeira fase do GPV, originando, assim, o algoritmo denominado GPV-REB.

Visto que os problemas-teste da única base de dados (BDS) existente na literatura referente ao problema tratado possuem os tempos de preparação da máquina simétricos, foram geradas outras duas novas bases de dados. Em uma delas, BDDT, os problemas-teste envolvem até 20 tarefas e satisfazem a desigualdade triangular com relação aos tempos de preparação da máquina. Na outra, BDNS, os problemas-teste envolvem até 150 tarefas e não satisfazem, necessariamente, a uma determinada propriedade, sendo, portanto, mais genérica.

O otimizador CPLEX 10.1 foi utilizado para resolver as três formulações matemáticas em tela, aplicadas em problemas-teste com até 14 tarefas. Os resultados computacionais mostram que as formulações MPLIM-G e MPLIM-BG proporcionam resultados semelhantes e que a formulação MPLIM-IT é mais eficiente que estas duas na resolução de problemas com até 12 tarefas, dado que esta última proporcionou ao CPLEX encontrar soluções com garantia de melhor qualidade e com menor tempo médio de processamento que as demais. Para os problemas-teste com mais de 12 tarefas, devido ao limite de tempo estabelecido para o CPLEX retornar uma solução, os resultados obtidos não permitem uma comparação efetiva entre os três modelos.

Os algoritmos GPV e GPV-REB foram aplicados em todos os problemas-teste disponíveis, divididos em três baterias de testes. A primeira delas foi realizada com a base de dados BDS, que é composta de problemas-teste com até 75 tarefas. Os resultados encontrados pelo GPV foram comparados com os de Gomes Júnior (2007), Penna (2009) e Ribeiro (2009). O GPV apresentou desempenho superior ao algoritmo proposto por Gomes Júnior (2007), tanto em relação à qualidade das soluções encontradas, quanto em relação ao tempo computacional médio demandado. Apesar de os resultados encontrados pelo GPV serem um pouco inferiores aos encontrados pelos algoritmos de Penna (2009) e Ribeiro (2009), os tempos computacionais médios demandados pelo GPV foram significativamente inferiores aos demandados pelos algoritmos de tais autores.

Na segunda e na terceira bateria de testes foram utilizadas, respectivamente, as bases de dados BDNS e BDDT. Os resultados encontrados pelo GPV foram comparados com os de Ribeiro (2009), que foi o único autor a utilizar também esta base de dados. No conjunto de problemas-teste BDNS, o GPV mostrou-se mais eficiente que o algoritmo de tal autor, encontrando soluções de melhor qualidade e com tempo computacional bastante inferior. Já com a base de dados BDDT, as soluções encontradas por ambos os algoritmos foram semelhantes, porém o GPV exigiu menor tempo computacional. Ainda na terceira bateria de testes, utilizou-se o GPV para determinar o horizonte de planejamento do MPLIM-IT, o que possibilitou ao CPLEX encontrar soluções bem próximas das ótimas em todos os problemas-teste com até 15 tarefas.

Nas três baterias de testes, os tempos computacionais demandados pelo algoritmo GPV-REB foram sempre significativamente inferiores aos respectivos tempos despendidos pelo GPV. Apesar de as soluções encontradas pelo GPV-REB serem um pouco inferiores àquelas encontradas pelo GPV, esta diferença diminui à medida em que se aumenta o número de tarefas do problema. Isto sugere que vale a pena investir em técnicas e propriedades que permitam reduzir o espaço de busca.

Enfim, conclui-se que a metodologia proposta é competitiva com aquelas existentes na literatura, tendo o baixo custo computacional como o seu grande aliado. Desta forma, foi possível resolver problemas-teste com até 150 tarefas em tempo computacional viável.

6.2 Trabalhos Futuros

Como trabalhos futuros, sugere-se:

- Comparar o GPV com o GRASP tradicional afim de evidenciar os ganhos;
- Fazer um estudo comparativo do GPV com e sem o VND_2 como estratégia de pós-

otimização;

- Testar a aplicação do VND_2 como busca local do GRASP e não como estratégia de pós-otimização. Testar, também, tal procedimento juntamente com a estratégia de redução do espaço de busca;
- Testar novas estratégias de redução do espaço de busca;
- Fazer um estudo de qual tamanho de bloco proporciona melhores resultados durante a exploração da vizinhança N^{RB};
- Experimentar novos valores para M no MPLIM-G e no MPLIM-BG, visto que tal valor pode ter impacto no desempenho do CPLEX;
- Estudar novas estratégias para determinar os limites h_{inf} e h_{sup} do MPLIM-IT, dado que esse modelo é dependente de tais limites.

Publicações

A seguir são listados os trabalhos provenientes da presente pesquisa que foram apresentados e aceitos para publicação nos anais dos respectivos eventos.

Título: Princípio da Otimalidade Próxima e GRASP para o problema de sequenciamento de tarefas em uma máquina com penalidades por antecipação e atraso da produção.
Autores: Bruno Ferreira Rosa e Marcone Jamilson Freitas Souza.
Evento: IV Semana de Ciência & Tecnologia do CEFET-MG.
Local: Belo Horizonte/MG - Brasil.
Período: 20 a 26 de outubro de 2008.

Título: Uma nova formulação de programação matemática indexada no tempo para uma classe de problemas de sequenciamento em uma máquina.
Autores: Bruno Ferreira Rosa e Marcone Jamilson Freitas Souza.
Evento: XLI SBPO - Simpósio Brasileiro de Pesquisa Operacional.
Local: Porto Seguro/BA - Brasil.
Período: 1 a 4 de setembro de 2009.

Título: Formulações de programação matemática para o problema de sequenciamento em uma máquina com janelas de entrega distintas e tempo de preparação dependente da sequência de produção.
Autores: Bruno Ferreira Rosa, Marcone Jamilson Freitas Souza e Sérgio Ricardo de Souza.
Evento: XXXII CNMAC - Congresso Nacional de Matemática Aplicada e Computacional.
Local: Cuiabá/MT - Brasil.
Período: 8 a 11 de setembro de 2009.

Título: Princípio da Otimalidade Próxima aplicado ao problema de sequenciamento de tarefas em uma máquina com penalidades por antecipação e atraso da produção.
Autores: Bruno Ferreira Rosa, Marcone Jamilson Freitas Souza e Sérgio Ricardo de Souza.
Evento: IX SBAI - Simpósio Brasileiro de Automação Inteligente.
Local: Brasília/DF - Brasil.
Período: 20 a 23 de setembro de 2009.

Referências Bibliográficas

Aiex, R. M., Resende, M. G. C., & Ribeiro, C. C. 2002. Probability Distribution Of Solution Time In Grasp: An Experimental Investigation. *Journal of Heuristics*, **8**, 343–373.

Alidaee, B., & Dragan, I. 1997. A note on minimizing the weighted sum of tardy and early completion penalties in a single machine: A case of small common due date. *European Joumal of Operational Research*, **96**, 559–563.

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. 1999. A review of scheduling research involving setup considerations. *Omega: The International Journal of Management Science*, **27**, 219–239.

Almeida, M. T., & Centeno, M. 1998. A composite heuristic for the single machine early/tardy job scheduling problem. *Computers & Operations Research*, **25**(7/8), 625–635.

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

Belfiore, P. P., da Silva, V. F., & Ronconi, D. P. 2006. Scatter Search para programação de tarefas em uma única máquina com penalidades de adiantamento e atraso e data de entrega comum. *In: Anais do XXVI Encontro Nacional de Engenharia de Produção.* Associação Brasileira de Engenharia de Produção, Fortaleza, CE, Brasil.

Biskup, D., & Feldmann, M. 2001. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers & Operations Research*, **28**, 787–801.

Bustamante, L. M. 2007. *Minimização do custo de antecipação e atraso para o problema de sequenciamento de uma máquina com tempo de preparação dependente da sequência: aplicação em uma usina siderúrgica.* Dissertação de mestrado, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte.

Chang, P. C. 1999. A Branch and Bound Approach for Single Machine Scheduling with Earliness and Tardiness Penalties. *Computers & Mathematics with Applications*, **37**, 133–144.

Christofoletti, L. M. 2002. *Métodos de Reinício Aplicados ao Sequenciamento em Uma Máquina com Tempos de Preparação e Datas de Entrega*. Dissertação de mestrado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas.

de Paula, M. R. 2008. *Heurísticas para a minimização dos atrasos em sequenciamento de máquinas paralelas com tempos de preparação dependentes da sequência*. Dissertação de mestrado, Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte.

Feldmann, M., & Biskup, D. 2003. Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers & Industrial Engineering*, **44**, 307–323.

Feo, T. A., & Resende, M. G. C. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, **6**, 109–133.

França Filho, M. F. 2007. *GRASP e Busca Tabu aplicados a problemas de programação de tarefas em máquinas paralelas*. Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas.

Gendreau, M. 2003. An Introduction to Tabu Search. *37-54 2 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, **13**(5), 533–549.

Glover, F. 1996. *Tabu Search and adaptive memory programming - advances, applications and challenges*. Interfaces in computer science and operations research edn. Kluwer Academic Publishers. Pages 1–75.

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

Glover, F., & Laguna, M. 2003. Scatter Search and Path Relinking: Advances and Applications. *Chap. 1, pages 1–35 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Gomes Júnior, A. C. 2007. *Problema de Sequenciamento em uma Máquina com Penalidades por Antecipação e Atraso: Modelagem e Resolução*. Dissertação de mestrado, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte.

Gray, P., Hart, W., Painton, L., Phillips, C., Trahan, M., & Wagner, J. 1997. *A Survey of Global Optimization Methods*. Tech. rept. Sandia National Laboratories. Disponível em $< http://www.cs.sandia.gov/opt/survey >$. Acesso em: 27 nov. 2008.

Gupta, S. R., & Smith, J. S. 2006. Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, **175**, 722–739.

Hansen, P., & Mladenović, N. 2003. Variable Neighborhood Search. *145-184 6 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Henderson, D., Jacobson, S. H., & Johnson, A. W. 2003. The Theory and Pratice of Simulated Annealing. *Chap. 10, pages 287–319 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics*. Kluwer Academic Publishers.

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**, 190–201.

James, R. J. W. 1997. Using tabu search so solve the common due date early/tardy machine scheduling problem. *Computers & Operations Research*, **24**(3), 199–208.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. 1983. Optimization by simulated annealing. *Science*, **220**, 671–680.

Koulamas, C. 1996. Single-machine scheduling with time windows and earliness/tardiness penalties. *European Journal of Operational Research*, **91**, 190–202.

Lee, C. Y., & Choi, J. Y. 1995. A Genetic Algorithm for Job Sequencing Problems with Distinct Due Dates and General Early-Tardy Penalty Weights. *Computers and Operations Research*, **22**(8), 857–869.

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**(2), 231–243.

Li, G. 1997. Single machine earliness and tardiness scheduling. *European Joumal of Operational Research*, **96**, 546–558.

Liaw, C.-F. 1999. A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers & Operations Research*, **26**, 679–693.

Lourenço, H. R., Martin, O. C., & Stützle, T. 2003. Iterated Local Search. *Chap. 11, pages 321–353 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Mazzini, R., & Armentano, V. A. 2001. A heuristic for single machine scheduling with early and tardy costs. *European Journal of Operational Research*, **128**, 129–146.

M'Hallah, R. 2007. Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. *Computers & Operations Research*, **34**, 3126–3142.

Mladenović, N., & Hansen, P. 1997. Variable Neighborhood Search. *Computers and Operations Research*, **24**(11), 1097–1100.

Mondal, S. A. 2002. Minimization of squared deviation of completion times about a common due date. *Computers & Operations Research*, **29**, 2073–2085.

Mondal, S. A., & Sen, A. K. 2001. Single machine weighted earliness tardiness penalty problem with a common due date. *Computers & Operations Research*, **28**, 649–669.

Panwalkar, S. S., Dudek, R. A., & Smith, M. L. 1973. Sequencing research and the industrial scheduling problem. *Pages 29–38 of:* Elmaghraby, S. E. (ed), *Symposium on the Theory of Scheduling and its Applications*.

Penna, P. H. V. 2009. *Um algoritmo heurístico híbrido para minimizar os custos com a antecipação e o atraso da produção em ambientes com janelas de entrega e tempos de preparação dependentes da sequência*. Dissertação de mestrado, Programa de Pós-Graduação em Engenharia Mineral, Universidade Federal de Ouro Preto (UFOP), Ouro Preto.

Rabadi, G., Mollaghasemi, M., & Anagnostopoulos, G. C. 2004. A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. *Computers & Operations Research*, **31**, 1727–1751.

Reeves, C. 2003. Genetic Algorithms. *Chap. 3, pages 55–82 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Resende, M. G. C., & Ribeiro, C. C. 2003. Greedy Randomized Adaptive Search Procedures. *219-249 8 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Ribeiro, C. C. 1996. Metaheuristics and Applications. In *Advanced School on Artificial Intelligence*, Estoril, Portugal.

Ribeiro, F. F. 2009. *Um algoritmo genético adaptativo para a resolução do problema de sequenciamento em uma máquina com penalização por antecipação e atraso da produção*. Dissertação de mestrado, Programa de Pós-Graduação em Modelagem Matemática e Computacional, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG), Belo Horizonte.

Sourd, F. 2005. Optimal timing of a sequence of tasks with general completion costs. *European Journal of Operational Research*, **165**, 82–96.

Souza, M. J. F. 2008. *Inteligência Computacional para Otimização*. Notas de aula. Disponível em $< http : //www.iceb.ufop.br/decom/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm >$. Acesso em: 20 nov. 2008.

Souza, M. J. F., Penna, P. H. V., Gonçalves, F. A. C. A., & Ochi, L. S. 2008. Uma heurística híbrida para minimizar custos com antecipação e atraso em sistemas de produção com janelas de entrega e tempos de preparação dependentes da sequência. *Pages 1–16 of: Anais do XI Simpósio de Pesquisa Operacional e Logística da Marinha*.

Szwarc, W. 1996. The weighted common due date single machine scheduling problem revisited. *Computers and Operations Research*, **23**(3), 255–262.

Tavakkoli-Moghaddam, R., Moslehi, G., Vasei, M., & Azaron, A. 2006. A branch-and-bound algorithm for a single machine sequencing to minimize the sum of maximum earliness and tardiness with idle insert. *Applied Mathematics and Computation*, **174**, 388–408.

Valente, J. M. S., & Alves, R. A. F. S. 2005a. Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers & Operations Research*, **32**, 557–569.

Valente, J. M.S., & Alves, R. A. F. S. 2005b. Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. *Computers & Industrial Engineering*, **48**, 363–375.

Ventura, J. A., & Radhakrishnan, S. 2003. Single machine scheduling with symmetric earliness and tardiness penalties. *European Journal of Operational Research*, **144**, 598–612.

Wan, G., & Yen, B. P.-C. 2002. Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *European Journal of Operational Research*, **142**, 271–281.

Weise, T. 2008. *Global Optimization Algorithms: Theory and Application*. 2ª edn. Disponível em: $< http://www.it-weise.de/ >$. Acesso em: 01 dez. 2008.

Weng, X., & Ventura, J. A. 1996. Scheduling about a given common due date to minimize mean squared deviation of completion times. *European Journal of Operational Research*, **88**, 328–335.

Ying, K.-C. 2008. Minimizing earliness-tardiness penalties for common due date single-machine scheduling problems by a recovering beam search algorithm. *Computers & Industrial Engineering*, doi:10.1016/j.cie.2008.01.008.

Ziviane, N. 2004. *Projeto de Algoritmos com implementações em PASCAL e C*. 2ª edn. Thomson.

Livros Grátis

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

Milhares de Livros para Download:

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