RELATÓRIO TÉCNICO-CIENTÍFICO

SEQUENCIAMENTO EM UMA MÁQUINA: OTIMIZAÇÃO HEURÍSTICA VIA MULTIPROCESSAMENTO PARALELO

Relatório Técnico-Científico Final referente ao tema Sequenciamento em uma máquina, vinculado ao desenvolvimento do projeto de pesquisa FAPEMIG PPM/CEX 00357/09

Frederico Augusto de Cezar Almeida Gonçalves

Marcone Jamilson Freitas Souza

Sérgio Ricardo de Souza

Belo Horizonte, Maio de 2010.

Resumo

Este trabalho tem seu foco no problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. São considerados tempos de preparação da máquina dependentes da sequência de produção, bem como a existência de janelas de entrega distintas. Para resolução do problema, propõe-se um algoritmo heurístico de processamento paralelo. Esse algoritmo, denominado GTSPR-M, combina os procedimentos heurísticos GRASP, *Variable Neighborhood Descent*, Busca Tabu e Reconexão por Caminhos. GRASP e *Variable Neighborhood Descent* são usados para gerar uma solução inicial, Busca Tabu para refinamento desta solução, e, por fim, a Reconexão por Caminhos como estratégia de pós-otimização. Para cada sequência gerada pela heurística é utilizado um algoritmo de tempo polinomial para determinar a data ótima de início de processamento de cada tarefa. Os resultados computacionais mostraram que houve melhoria em relação a vários algoritmos da literatura, tanto com relação à qualidade da solução final quanto em relação ao desvio médio e tempo de processamento.

PALAVRAS-CHAVE: Sequenciamento em uma máquina, GRASP, Descida em Vizinhança Variável, Busca Tabu, Reconexão por Caminhos, Processamento Paralelo, *Threads*.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

AEPG Algoritmos Evolutivos Paralelos Globais

AG Algoritmos Genéticos

AGA1 Algoritmo Genético Adaptativo, proposto em Ribeiro (2009)

API *Application Program Interface*

BS *Beam Search*

BT Busca Tabu

CC-NUMA Memória de Acesso Não-Uniforme com Coerência de Cache

CE Conjunto Elite

EDD *Earliest Due Date*

FIFO *First In First Out*

GILS-VND-RT Algoritmo heurístico proposto em Gomes Jr. et al. (2007), baseado em GRASP, ILS, VND, que explora o espaço de soluções usando movimentos de Realocação e Troca

GRASP *Greedy Randomized Adaptive Search Procedures*

GTSPR Algoritmo heurístico proposto em Souza et al. (2008), baseado em GRASP, Busca Tabu e Reconexão por Caminhos

GTSPR-M Algoritmo heurístico proposto no presente trabalho, baseado em GRASP, Busca Tabu e Reconexão por Caminhos, explorando processamento paralelo

ILS *Iterated Local Search*

IRPDD *Inventory Routing Problem with Direct Deliveries*

LC Lista de Candidatos

LRC Lista Restrita de Candidatos

MPLIM Modelo de Programação Linear Inteira Mista

MPLIM-BG MPLIM proposto em Rosa (2009)

MPLIM-G MPLIM proposto em Gomes Jr. et al. (2007)

MPLIM-IT MPLIM indexado no tempo, proposto em Rosa (2009)

MPP Processadores Paralelos Massivos

MRD Método Randômico da Descida

NORMA *No Remote Memory Access*

PCV Problema do Caixeiro Viajante

PDDOIP Procedimento de Determinação das Datas Ótimas de Início de Processamento das tarefas

PLIM Programação Linear Inteira Mista

PR *Path Relinking*

PSUM Problema de sequenciamento em uma máquina

PSUMAA Problema de sequenciamento em uma máquina com minimização das penalidades por antecipação e atraso

PSUMAA-JP Problema de sequenciamento em uma máquina com janelas de entrega, penalidades por antecipação e atraso da produção e tempos de preparação dependentes da sequência de produção

RBS *Recovering Beam Search*

SA *Simulated Annealing*

SMP Multiprocessadores Simétricos

SPT *Shortest Processing Time*

SSI *Single System Image*

TDD *Tardiest Due Date*

VMI *Vendor Managed Inventory*

VND *Variable Neighborhood Descent*

VNS *Variable Neighborhood Search*

WSPT *weighted shortest processing time*

Sumário

1 INTRODUÇÃO

Este trabalho tem seu foco no estudo do problema de sequenciamento de tarefas envolvendo penalizações pela antecipação e atraso da produção. De acordo com França Filho (2007), este estudo é mais recente do que estudos voltados para problemas onde o objetivo envolve uma função não-decrescente do instante de conclusão do processamento da tarefa, tais como: tempo médio de fluxo, soma ponderada de atrasos e *makespan*. Nestes problemas, o atraso na conclusão das tarefas gera os custos mais elevados. Com o advento da adoção da filosofia *just-in-time* por muitas empresas, a penalização pela antecipação da produção tornou-se parte deste problema. Esta penalização se justifica pelo fato de que concluir uma tarefa antecipadamente pode resultar em custos financeiros extras pela necessidade antecipada de capital e/ou espaço para armazenamento e/ou de outros recursos para manter e gerenciar o estoque.

De acordo com as datas de entrega das tarefas, são encontrados na literatura três tipos de problemas:

***Common Due Dates*:** Datas de entrega comuns.

***Distinct Due Dates*:** Datas de entrega distintas.

***Distinct Due Windows*:** Janelas de entrega distintas.

Com relação às datas de entrega comuns, considera-se uma única data para finalizar as tarefas. Já para as datas de entrega distintas, considera-se uma data de entrega associada a cada tarefa. Por final, para janelas de entrega distintas, considera-se um período de conclusão associado a cada tarefa. Segundo Wan e Yen (2002), janelas de entrega distintas são aplicadas em situações de incertezas ou tolerâncias com relação às datas de entrega e não há custos se as tarefas forem concluídas dentro da janela de entrega.

Muitas propriedades podem ser consideradas no estudo de problemas de sequenciamento de tarefas envolvendo penalizações pela antecipação e atraso da produção onde

as datas de entrega das tarefas são comuns. Estas propriedades podem beneficiar os algoritmos aplicados na resolução destes problemas, visto que uma redução do esforço computacional gasto por estes algoritmos na exploração do espaço de soluções pode ser obtido através destas propriedades. Com relação a esta classe de problemas, as tarefas que terminam seu processamento na data devida ou antes são ordenadas de forma decrescente pela relação entre o tempo de processamento e a penalização pela antecipação; enquanto as tarefas que começam seu processamento na data devida ou depois são sequenciadas de forma crescente pela relação entre o tempo de processamento e a penalização pelo atraso (BAKER; SCUDDER, 1990). Em função disso, diz-se que a sequência é V-*shaped*. Outra propriedade interessante é que na sequência ótima não há tempos ociosos entre duas tarefas consecutivas.

Em problemas de sequenciamento nos quais as datas de entrega são distintas, assim como em problemas de sequenciamento com janelas de entrega distintas, as propriedades aplicadas aos problemas de sequenciamento com datas de entrega comuns não são necessariamente válidas, o que torna esta classe de problemas mais complexa. Nestes problemas, há a necessidade de saber se é permitida a ociosidade das máquinas. Segundo França Filho (2007), existem situações em que a ociosidade não é permitida por gerar custos maiores que aqueles decorrentes da conclusão antecipada das tarefas. Entretanto, há situações em que vale a pena manter uma máquina ociosa, mesmo que haja uma tarefa em condições de ter seu processamento iniciado. Para ilustrar estas situações, são consideradas duas tarefas i e j consecutivas na sequência de produção, as quais têm que ser processadas em uma máquina disponível somente no instante 20. Os valores associados ao tempo de processamento, data de entrega, multa por atraso e multa por antecipação da produção de cada tarefa são exibidos na Tabela 1.

Tabela 1: Exemplo de processamento com datas e janelas de entrega distintas

Tarefa	Processamento	Data de Entrega	Multa por Antecipação	Multa por Atraso
i	15	50	22	100
j	25	55	2	20

De acordo com os valores apresentados na Tabela 1, se a tarefa i começar no instante 20, será concluída no instante 35 com uma penalização pela antecipação igual a 330 ($= 15 \times 22$). A seguir, a tarefa j poderá começar no instante 35 sendo finalizada no instante 60, com uma multa pelo atraso igual 100 ($= 5 \times 20$). Esta sequência resulta em um custo total de 430 ($= 330 + 100$). A Figura 1 exibe esta sequência.

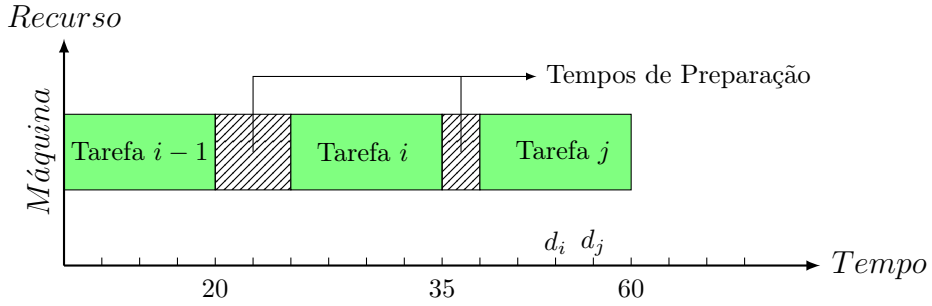


Figura 1: Sequência sem tempo ocioso entre tarefas

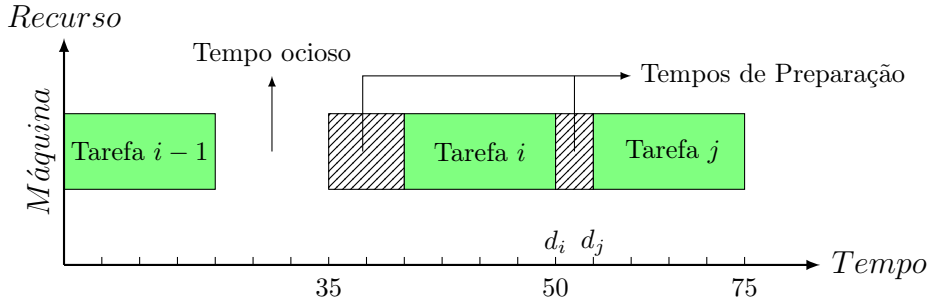


Figura 2: Sequência com tempo ocioso entre tarefas

Uma outra situação considera que a tarefa i seja iniciada no instante 35, sendo concluída no instante 50, sem multa por antecipação ou atraso da produção. A seguir, a tarefa j poderá começar no instante 50 sendo finalizada no instante 75, com uma multa por atraso de 400 ($= 20 \times 20$). Esta sequência é mostrada na Figura 2. O custo total nesta última situação é de 400 unidades, menor que aquele no qual a tarefa i começava no instante 20. Assim, quando não há restrições com relação à ociosidade das máquinas, adiar o início de processamento de certas tarefas, ou seja, inserir tempos ociosos entre tarefas, poderá produzir soluções de menores custos.

Observa-se que na maioria dos trabalhos científicos, os tempos de preparação das tarefas na sequência de produção, ou tempo de *setup*, têm sido negligenciados ou adicionados ao tempo de processamento de cada tarefa. Neste tipo de situação os tempos de *setup* são considerados independentes da sequência das tarefas. Entretanto, alguns estudos, tais como Panwalkar, Dudek e Smith (1973), *apud* Gupta e Smith (2006), indicam que em muitos processos produtivos, o tempo de preparação é dependente da sequência de produção. De acordo com Kim e Bobrowski (1994), para modelar melhor casos práticos, esses tempos devem ser considerados explicitamente sempre que eles forem significativamente maiores que os tempos de processamento. Uma revisão de trabalhos sobre sequenciamento da produção com tempos de preparação pode ser encontrada em Allahverdi, Gupta e Al-

dowaisan (1999).

No presente trabalho, é estudada uma classe de problemas de sequenciamento em uma máquina com minimização das penalidades por antecipação e atraso da produção (*Single Machine Scheduling for Minimizing Earliness and Tardiness Penalties*), ou simplesmente PSUMAA. Trata-se aqui o PSUMAA com janelas de entrega distintas e tempos de preparação da máquina dependentes da sequência da produção, doravante representado por PSUMAA-JP. Este problema representa uma generalização dos problemas de sequenciamento em uma máquina com penalidades por antecipação e atraso. De fato, quando a janela de entrega de uma tarefa reduz-se a uma data de entrega, tem-se o problema com datas de entrega distintas. Caso as janelas sejam as mesmas para todas as tarefas e estas se reduzem a uma mesma data, tem-se o problema com datas comuns de entrega. Considerando-se tempo de preparação nulo, tem-se o problema de sequenciamento com tempo de preparação independente da sequência.

Esse tema foi tratado por Gomes Jr. et al. (2007) por meio de um algoritmo heurístico baseado em *Iterated Local Search* (ILS) e GRASP. Posteriormente, Souza et al. (2008) propuseram um algoritmo baseado em Busca Tabu, combinado com *Variable Neighborhood Descent* (VND) e Reconexão por Caminhos. Em Ribeiro (2009), foi desenvolvido um algoritmo genético, o qual usa 5 operadores de cruzamento que são aplicados de acordo com o desempenho em gerações passadas. Os operadores mais bem sucedidos têm maior probabilidade de serem usados. O algoritmo ainda usa diversas regras de despacho para gerar a população inicial por meio da fase de construção GRASP. Em Rosa (2009) foi desenvolvida uma formulação de programação matemática indexada no tempo, bem como um algoritmo heurístico baseado em GRASP e no princípio da otimalidade próxima.

Todos esses algoritmos desenvolvidos não exploram a capacidade de multiprocessamento da geração atual dos computadores. Com o aparecimento de recursos *multicore* nos computadores atuais é possível melhorar o desempenho da execução dos mesmos. O presente trabalho, então, trata de cobrir esta lacuna. Propõe-se, assim, o desenvolvimento de um algoritmo heurístico paralelo para resolver o problema em questão. Em linhas gerais, ele funciona como segue. Na primeira fase, aplica-se GRASP (Seção 5.2.5) para gerar uma solução inicial. Nesta fase, a solução é construída distribuindo-se o processamento entre vários *threads*, sendo estes processados em paralelo. A melhor solução encontrada por algum destes *threads* é utilizada como solução inicial para o algoritmo. A seguir, na segunda fase, esta solução é refinada pelo procedimento Busca Tabu (Seção 5.2.6). Neste procedimento, a geração de vizinhos bem como o refinamento destes são executados de

forma paralela. Durante a execução do procedimento Busca Tabu também é construído um conjunto de soluções denominado Conjunto Elite. Este conjunto se caracteriza por possuir boas soluções e diferenciadas entre si. O Conjunto Elite é utilizado na terceira fase, na qual é executado o procedimento Reconexão por Caminhos (Seção 5.2.7) como estratégia de pós-otimização. A Reconexão é aplicada a cada par de soluções do Conjunto Elite, sendo cada par executado em modo paralelo em um *thread*.

1.1 Objetivos

Este trabalho tem por objetivo geral o desenvolvimento de um algoritmo heurístico híbrido paralelo, baseado em GRASP, Busca Tabu, VND e Reconexão por Caminhos para resolver problemas de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção, considerando a existência de janelas de tempo e tempos de preparação de máquina dependentes da sequência.

Os objetivos específicos são os seguintes:

1. Desenvolver um algoritmo heurístico paralelo para resolver o problema em questão em menor tempo computacional que a sua versão sequencial, proposta em Souza et al. (2008);
2. Implementar o algoritmo proposto na linguagem C/C++;
3. Comparar o algoritmo proposto com a sua versão sequencial e com outros algoritmos da literatura;
4. Publicar resultados da pesquisa em eventos científicos.

1.2 Motivação

Além da importância prática desse problema, com inúmeras aplicações industriais (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999), de acordo com Du e Leung (1990), Liaw (1999), Biskup e Feldmann (2001), Wan e Yen (2002), este problema pertence à classe NP-difícil. Isto implica em uma grande dificuldade de se resolver este problema na otimalidade e em tempos computacionais aceitáveis, mesmo que este envolva um número mediano de tarefas (Gomes Jr. et al., 2007). Esta aplicabilidade associada à grande dificuldade na resolução do problema motivaram o desenvolvimento deste trabalho. Além disso, todos

os trabalhos encontrados na literatura não exploram a resolução do mesmo por meio de processamento paralelo. Com a popularização de máquinas que se caracterizam por possuírem dois ou mais núcleos de processamento, o desenvolvimento de um algoritmo que aproveite a disponibilidade destes núcleos implicará em ganho de desempenho.

1.3 Estrutura do trabalho

O restante deste trabalho está estruturado como segue. No capítulo 2 são descritas as técnicas heurísticas utilizadas neste trabalho, bem como os conceitos mais importantes sobre paralelização de algoritmos. No capítulo 3 o problema em questão é caracterizado, sendo a revisão bibliográfica sobre o tema apresentada no capítulo 4. O capítulo 5 trata da metodologia proposta e está subdividido em duas seções. Na primeira, Seção 5.1, são apresentados vários modelos matemáticos aplicados na resolução do problema estudado. Já na segunda, Seção 5.2, apresenta-se o algoritmo heurístico paralelo desenvolvido neste trabalho. Resultados computacionais são apresentados no capítulo 6. Finalmente, no capítulo 7, conclui-se este trabalho, bem como apresentam-se possíveis propostas a serem exploradas em trabalhos futuros.

2 CONCEITOS BÁSICOS

Neste capítulo são apresentados alguns conceitos básicos necessários ao entendimento deste trabalho. Inicialmente é apresentado o método GRASP (Seção 2.1). Na Seção 2.2 são apresentados alguns modelos de agendamento de tarefas. A seguir, o método de refinamento *MDR* (Seção 2.3) é apresentado, assim como as metaheurísticas VND (Seção 2.4), Busca Tabu (Seção 2.5) e o mecanismo Reconexão por Caminhos (Seção 2.6). Também é apresentada uma definição para *threads* (Seção 2.7). Por fim, são apresentados alguns Sistemas de Processamento Paralelo na Seção 2.8.

2.1 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedures*), proposto por Feo e Resende (1995), é um método iterativo de duas fases: construção da solução e o refinamento dela. Na sua definição, o método GRASP combina elementos de um algoritmo guloso, assim como elementos de aleatoriedade. Seu pseudocódigo para um problema de minimização é apresentado no Algoritmo 1. De acordo com ele, o procedimento executa *GRASPmax* vezes e retorna a melhor solução representada por v^*.

Algoritmo 1: GRASP

Entrada: $f()$, $g()$, $N()$, $GRASPmax$

Saída: v^*

```
1  início
2      f* ← ∞
3      v* ← ∅
4      para i=1 até GRASPmax faça
5          v ← SOLUCAO-INICIAL(γ, g())
6          v' ← BUSCA-LOCAL(v, f(), N())
7          se (f(v') < f*) então
8              f* ← f(v')
9              v* ← v'
10         fim
11     fim
12     retorne v*
13 fim
```

Na construção da solução, linha 5, uma solução inicial é gerada pela inserção gradativa de um elemento. Para isto, uma lista LC de candidatos é formada. Esta lista é ordenada segundo algum critério de classificação dos elementos. Este critério é baseado em uma função adaptativa gulosa dada por g. Após a ordenação de LC, os elementos mais bem classificados formam a lista restrita de candidatos LRC. O parâmetro $\gamma \in [0, 1]$ é utilizado para dimensionar LRC, e desta maneira, controlar o nível de aleatoriedade do método. Quando $\gamma = 0$, o método executa de maneira puramente gulosa; em contrapartida, quando $\gamma = 1$, a execução é totalmente aleatória. A cada passo, um elemento de LRC é aleatoriamente sorteado e adicionado à solução em construção. A seguir, os elementos são novamente reclassificados em LC para refletir as mudanças advindas da inserção anterior, e assim, LRC também é atualizada. A construção da solução encerra quando todos os elementos forem analisados. O Algoritmo 2 descreve este método.

Algoritmo 2: SOLUCAO-INICIAL

Entrada: γ, $g()$

Saída: v

1 **início**
2 $v \leftarrow \emptyset$
3 Inicialize a lista LC de candidatos
4 **enquanto** $(LC \neq \emptyset)$ **faça**
5 $g_{\min} \leftarrow \min\{g(t) \mid t \in LC\}$
6 $g_{\max} \leftarrow \max\{g(t) \mid t \in LC\}$
7 $LRC \leftarrow \{t \in LC \mid g(t) \leq g_{\min} + \gamma \times (g_{\max} - g_{\min})\}$
8 Selecione aleatoriamente um elemento $t \in LRC$
9 $v \leftarrow v \cup \{t\}$
10 Atualize a lista LC de candidatos
11 **fim**
12 **retorne** v
13 **fim**

Encerrada a fase de construção, a solução retornada possivelmente não é um ótimo local com relação à vizinhança $N()$ adotada. Em função disso, uma busca local (linha 6 do Algoritmo 1) é aplicada à solução construída no intuito de refiná-la. Considerando-se problemas de minimização, um procedimento básico de busca local é definido no Algoritmo 3.

Algoritmo 3: BUSCA-LOCAL

Entrada: v, $N()$

Saída: v

1 **início**
2 $L \leftarrow \{v' \in N(v) \mid f(v') < f(v)\}$
3 **enquanto** $(L \neq \emptyset)$ **faça**
4 Selecione um vizinho $v' \in L$
5 $v \leftarrow v'$
6 $L \leftarrow \{v' \in N(v) \mid f(v') < f(v)\}$
7 **fim**
8 **retorne** v
9 **fim**

2.2 Modelos de agendamento

Apresentam-se, a seguir, alguns modelos de agendamento de tarefas comumente utilizados.

***Earliest Due Date* - EDD:** Neste modelo, as tarefas são ordenadas pela data de entrega, sendo a mais recente a primeira.

***Tardiest Due Date* - TDD:** Neste modelo, as tarefas são ordenadas pela data de entrega, sendo a mais tardia a primeira.

***Shortest Processing Time* - SPT:** Neste modelo, as tarefas são ordenadas pelo tempo de processamento, sendo a de menor tempo a primeira.

Para ilustrar, seja um conjunto de 6 tarefas com as datas de entrega e tempos de processamento constantes na Tabela 2. Pela regra EDD, a sequência de produção seria: $v_{EDD} = (1\ 6\ 3\ 2\ 4\ 5)$; enquanto pela regra TDD seria $v_{TDD} = (5\ 4\ 2\ 3\ 6\ 1)$ e pela regra SPT: $v_{SPT} = (6\ 3\ 2\ 4\ 1\ 5)$.

Tarefa	Processamento	Data de Entrega
1	5	1
2	3	4
3	2	3
4	4	5
5	6	6
6	1	2

Tabela 2: Modelo de Agendamento

2.3 MDR

O Método de Descida Randômica (*Random Descent Method*), é uma variação do Método de Descida convencional para problemas de minimização. Na Descida convencional ou Descida Completa (conhecida na literatura inglesa por *Best Improvement Method*), uma avaliação completa da vizinhança é executada; desta maneira, é necessário um esforço computacional maior para a execução do método. Já a Descida Randômica evita essa exploração exaustiva, resultando em um menor esforço computacional. Nesta variação, os vizinhos da solução corrente são analisados de forma aleatória como se segue: dado um

vizinho qualquer da solução corrente, ele somente é aceito como a nova solução corrente se ele for melhor avaliado que a atual solução corrente; caso contrário, a solução corrente permanece inalterada. Após *MDRmax* iterações sem melhora, o refinamento é encerrado. Devido ao fato de que nenhuma exploração completa da vizinhança é executada, não há garantia que a solução refinada seja de fato um ótimo local. Esse procedimento está descrito no Algoritmo 4. Nela, v representa a solução a ser refinada, $f()$ uma função de avaliação, $N()$ a estrutura de vizinhança adotada e *MDRmax* o número máximo de iterações sem melhora.

Algoritmo 4: MDR

```
   Entrada: v, f(), N(), MDRmax
   Saída: v
 1 início
 2 |  IterCorrente ← 0
 3 |  enquanto (IterCorrente < MDRmax) faça
 4 |  |  Selecione aleatoriamente um vizinho v' ∈ N(v)
 5 |  |  se (f(v') < f(v)) então
 6 |  |  |  v ← v'
 7 |  |  |  IterCorrente ← 0
 8 |  |  fim
 9 |  |  IterCorrente ← IterCorrente + 1
10 |  fim
11 |  retorne v
12 fim
```

2.4 VND

A Descida em Vizinhança Variável (*Variable Neighborhood Descent* – VND), proposta por (MLADENOVIĆ; HANSEN, 1997), é uma metaheurística que tem como objetivo explorar o espaço de soluções através de trocas sistemáticas das estruturas de vizinhanças. O procedimento considera a existência de várias estruturas de vizinhanças $N = \{N^{(1)}, N^{(2)}, \ldots, N^{(r)}\}$, as quais são utilizadas no refinamento de uma solução. Assim, o procedimento começa o processo de refinamento partindo vizinhança de menor índice, geralmente a de menor complexidade. A troca de vizinhança ocorre quando se alcança um ótimo local com relação à vizinhança adotada. No caso de se encontrar alguma melhora nessa nova vizinhança, a solução é aceita e o procedimento recomeça o refinamento retornando à primeira vizinhança da lista. O procedimento VND para quando a solução corrente represente um ótimo local com relação a todas as vizinhanças consideradas. O procedimento VND se baseia em três princípios básicos:

- Um ótimo local com relação a uma dada estrutura de vizinhança não corresponde necessariamente a um ótimo local com relação a uma outra estrutura de vizinhança;
- Um ótimo global corresponde a um ótimo local para todas as estruturas de vizinhança;
- Para muitos problemas, ótimos locais com relação a uma ou mais estruturas de vizinhança são relativamente próximos.

A seguir, é apresentado no Algoritmo 5, o pseudocódigo de um procedimento VND básico. Nesse pseudocódigo, v representa a solução a ser refinada, $f()$ uma função de avaliação, $N()$ as estruturas de vizinhança adotadas e r o número de vizinhanças adotadas.

Algoritmo 5: VND

```
   Entrada: v, f(), N(), r
   Saída: v
 1 início
 2 |  k ← 1
 3 |  enquanto (k ≤ r) faça
 4 |  |  Selecione o melhor vizinho v' ∈ N^(k)(v)
 5 |  |  se (f(v') < f(v)) então
 6 |  |  |  v ← v'
 7 |  |  |  k ← 1
 8 |  |  fim
 9 |  |  k ← k + 1
10 |  fim
11 |  retorne v
12 fim
```

A análise do espaço de soluções pelo VND pode ser, no entanto, custosa computacionalmente. Em tais casos, uma busca pela primeira solução de melhora pode ser aplicada, bem como uma busca apenas em certo percentual da vizinhança. Por fim, uma busca baseada no procedimento MDR (vide Seção 2.3), também é uma alternativa viável.

2.5 Busca Tabu

A Busca Tabu (*Tabu Search* – TS), de acordo com Glover e Laguna (1997), é um procedimento de busca local que explora o espaço de soluções utilizando uma estrutura de memória. Ele consiste em avaliar a vizinhança de uma solução e mover para o seu melhor vizinho. Após este movimento, o vizinho ou atributos que melhor representem

o vizinho são armazenados em uma estrutura de memória. Esta estrutura é utilizada para impedir por um tempo determinado o retorno a soluções previamente visitadas. A restrição é adotada para que a busca não fique presa em ótimos locais e consiga explorar o espaço de soluções mais efetivamente.

Para exemplificar o procedimento, considere a solução inicial v e a vizinhança $N()$ adotada. Após a avaliação da vizinhança $N(v)$, o melhor vizinho v' de v é selecionado. Esta escolha poderá implicar em um retorno a uma solução visitada anteriormente, e desta maneira, criar um ciclo que impedirá o progresso do método. De forma a contornar este problema, a escolha do melhor vizinho deverá obedecer a alguns critérios de seleção. Para tal, uma estrutura de memória conhecida como lista tabu T é utilizada para auxiliar a escolha do vizinho. São armazenados em T os movimentos tabu m, assim conhecidos os atributos que impeçam o retorno à solução previamente visitada. Durante $|T|$ iterações, as soluções geradas por movimentos tabu em T não serão permitidas. A lista tabu possui tamanho fixo e funciona como uma fila, isto é, se a lista estiver cheia e um novo item for adicionado, o mais antigo sai. Assim, para que o vizinho v' seja selecionado, o movimento m aplicado a v para gerar v' não deve fazer parte de T.

Apesar da lista tabu reduzir o risco da formação de ciclos, esta restrição também poderá impedir a seleção de vizinhos ainda não visitados. Para que este tipo de movimento não seja proibido, um outro critério de seleção também é adotado. A função de aspiração $A()$ é utilizada para que em certas circunstâncias, movimentos presentes em T sejam permitidos. Desta maneira, considerando um vizinho v' avaliado pela função de avaliação $f()$, existe um nível de aspiração dado por $A(f(v'))$, que permitirá a escolha deste vizinho quando $f(v') < A(f(v'))$, mesmo que o movimento gerado na escolha do vizinho esteja em T. Para cada valor retornado na avaliação de qualquer vizinho v' de v, existe um outro valor dado por $A(f(v'))$ que determina o ponto de aspiração corrente do procedimento. Uma estratégia comumente adotada é considerar $A(f(v)) = f(v_{\min})$, onde $v_{\min}$ é a melhor solução corrente. Se algum vizinho gerado por um movimento presente em T for melhor avaliado que $v_{\min}$, então isso prova que esse vizinho se trata de uma solução ainda não visitada.

O procedimento pode ser interrompido de várias maneiras. Entretanto, duas delas são mais comumente aplicadas. Na primeira, o procedimento é interrompido após $TABUmax$ iterações sem melhora. Já na segunda, quando se conhece o limite inferior (no caso de problema de minimização) ou superior (no caso de maximização) de um problema e o método atingir este limite, a sua execução também pode ser interrompida.

A seguir é apresentado o pseudocódigo do procedimento Busca Tabu no Algoritmo 6. Neste algoritmo, v representa a solução corrente a ser refinada, $f()$ a função de avaliação, $A()$ a função de aspiração, $N()$ a vizinhança adotada, *difElite* e *CE* os parâmetros utilizados para construção do Conjunto Elite utilizado no método Reconexão por Caminhos (Vide Seção 2.6) e *TABUmax*, o número máximo de iterações sem melhora. O procedimento retorna a melhor solução em v^*.

Algoritmo 6: TABU

Entrada: v, $f()$, $A()$, $N()$, *difElite*, *CE*, *TABUmax*
Saída: v^*

```
1  início
2      v* ← v                                              // Melhor solução
3      T ← ∅                                               // Lista Tabu
4      IterCorrente ← 0                                    // Iteração corrente
5      MelhorIter ← 0                                      // Iteração da melhor solução
6      Inicialize a função de aspiração A
7      enquanto (IterCorrente − MelhorIter ≤ TABUmax) faça
8          Seja v' ← v ⊕ m o melhor elemento de V ⊆ N(v), e tal que o movimento m
           não seja tabu (m ∉ T) ou, se tabu, v' atenda a condição de aspiração,
           definida por f(v') < A(f(v'))
9          Atualize a lista tabu T
10         v ← v'
11         se (f(v) < f(v*)) então
12             v* ← v
13             MelhorIter ← IterCorrente
14         fim
15         IterCorrente ← IterCorrente + 1
16         Atualize a função de aspiração A
17         Atualize o Conjunto Elite CE
18     fim
19     retorne v*
20 fim
```

2.6 Reconexão por Caminhos

A Reconexão por Caminhos (*Path Relinking*) é uma estratégia que faz um balanço entre intensificação e diversificação. Foi proposta originalmente por (GLOVER, 1996) para ser utilizada em conjunto com a Busca Tabu (Vide Seção 2.5). Considerando um par de soluções, sendo uma delas a solução base e a outra solução guia, o objetivo deste método é partir da solução base e caminhar para a solução guia por meio da inserção gradativa de atributos da solução guia na solução base.

Durante a execução da Busca Tabu, um conjunto de soluções de boa qualidade é formado (Vide linha 17 do Algoritmo 6). Este conjunto, conhecido na literatura como Conjunto Elite (CE), possui tamanho fixo e para fazer parte dele a solução candidata deve obedecer a algum dos seguintes critérios:

- Ser melhor que a melhor solução de CE.
- Ser melhor que a pior solução de CE e ainda se diferenciar de todas as outras soluções em um certo percentual dos atributos, definido por *difElite*.

O segundo critério é adotado para que CE não seja formado por soluções muito parecidas. Estando CE completo, para que uma nova solução seja inserida, a de pior avaliação é removida. Na intenção de gerar soluções de melhor qualidade, novos caminhos são gerados e explorados partindo-se de soluções pertencentes a CE e levando a outras soluções de CE.

A Reconexão por Caminhos pode ser aplicada em duas estratégias:

- Como pós-otimização, sendo aplicada entre todos os pares de soluções de CE.
- Como intensificação, sendo aplicada a cada ótimo local obtido após a fase de busca local.

Na primeira estratégia, a pós-otimização é aplicada após o término da Busca Tabu. Para tal, a Reconexão por Caminhos é aplicada a todos os pares possíveis de soluções de CE, sendo retornada a melhor solução encontrada. Já na segunda estratégia, a intensificação é realizada aplicando-se a Reconexão por Caminhos a cada ótimo local após a busca local. Desta maneira, o par de soluções a ser avaliado será formado pelo ótimo local da busca e alguma solução aleatoriamente selecionada de CE. Se a solução guia é definida como a solução selecionada de CE e a solução base como o ótimo local, esta Reconexão por Caminhos é dita Progressiva (*forward*). Entretanto, se a solução guia é definida como o ótimo local e a solução base como a solução pertencente a CE, esta Reconexão por Caminhos é dita Regressiva (*backward*).

Dentre as duas estratégias descritas, a segunda é considerada mais eficiente (ROSSETI, 2003), assim como a Reconexão por Caminhos Regressiva (RIBEIRO; UCHOA; WERNECK, 2002). A seguir, é apresentado no Algoritmo 7 o procedimento Reconexão por Caminhos aplicado a um problema de minimização. Neste procedimento, são passados como

parâmetros a solução base e a solução guia, bem como a função de avaliação e a vizinhança adotada, respectivamente. O método retorna a melhor solução em v e encerra sua execução quando a solução base for igual à solução guia.

Algoritmo 7: PR

```
   Entrada: sol_base, sol_guia, f(), N()
   Saída: v'
 1 início
 2 |  Inicialize ListaAtributos com todos os atributos da solução guia
 3 |  enquanto (sol_base ≠ sol_guia) faça
 4 |  |  melhorAtributo ← ∅
 5 |  |  melhorValor ← ∞
 6 |  |  para j = 1 até total de atributos de ListaAtributos faça
 7 |  |  |  Insira em sol_base o atributo j de sol_guia
 8 |  |  |  se (f(sol_base) < melhorValor) então
 9 |  |  |  |  melhorAtributo ← j
10 |  |  |  |  melhorValor ← f(sol_base)
11 |  |  |  fim
12 |  |  |  Desfaça a inserção do atributo j em sol_base
13 |  |  fim
14 |  |  Insira em sol_base o atributo melhorAtributo de sol_guia
15 |  |  Retire de ListaAtributos o atributo melhorAtributo
16 |  |  v' ← Aplique uma busca local em sol_base preservando todos os atributos já
   |  |  inseridos
17 |  fim
18 |  retorne v'
19 fim
```

O método proposto permite a utilização tanto da estratégia de pós-otimização quanto da estratégia de intensificação. Em ambos os casos, os pares de soluções deverão ser passados para o método através dos parâmetros sol_{guia} e sol_{base}. Para a estratégia de intensificação, a ordem em que esses parâmetros são passados definem o tipo de execução: *forward* e *backward*.

2.7 Threads

Thread ou processo leve (*processa*), de acordo com Tanenbaum e Woodhull (2002), é uma unidade básica de utilização da CPU. O *thread* tal como o processo tradicional, possui seu próprio contador de programa, conjunto de registradores e uma pilha de execução. Um processo tradicional possui um único *thread* de controle; entretanto, se um processo possuir mais do que um único *thread* de controle, ele poderá executar mais tarefas a cada

momento. Neste tipo de processo, os seus *threads* de controle compartilharão a seção de código, a seção de dados, arquivos abertos, dentre outros recursos do Sistema Operacional.

Existem vários benefícios com a utilização de *threads*, dentre eles se destacam: capacidade de resposta, compartilhamento de recursos, economia de recursos e utilização de arquitetura de multiprocessadores. A capacidade de resposta é melhorada devido ao fato de que vários *threads* poderão estar em execução, permitindo assim que um processo continue executando mesmo que algum dos seus *threads* esteja bloqueado. O compartilhamento de recursos é uma vantagem, pois permite que vários *threads* estejam em execução dentro do mesmo espaço de endereçamento. Este compartilhamento de recursos implica em uma economia dos recursos do sistema, visto que a alocação de memória e outros recursos para a criação de um novo processo é dispendiosa computacionalmente. Como os *threads* de controle compartilham o mesmo espaço de endereçamento, é mais econômico criar novos *threads*, bem como comutar seus contextos. Uma outra grande vantagem da utilização de *threads* está no fato de que uma arquitetura com vários processadores permite que os vários *threads* possam executar em paralelo, cada qual podendo executar em um dos processadores disponíveis. Já para uma arquitetura com um único processador, a comutação entre os *threads* pode ser tão rápida que dará a impressão de um falso paralelismo.

O suporte a *threads* pode ser fornecido em nível de *kernel* ou em nível de usuário, isto é, os *threads* podem ser gerenciados pelo *kernel* do Sistema Operacional ou serem gerenciados por uma biblioteca no nível de usuário. Embora as duas opções pareçam equivalentes, elas se diferem com relação ao desempenho. De acordo com Tanenbaum e Woodhull (2002), a comutação de *threads* no espaço usuário é muito mais rápida que no *kernel*; desta maneira, o gerenciamento de *threads* no espaço do usuário é geralmente mais rápido. Entretanto, se algum dos *threads* do processo for bloqueado em função de alguma chamada de sistema, o *kernel* bloqueará todo o processo. Isso ocorre uma vez que o *kernel* desconhece a existência dos outros *threads*, ou seja, ele enxerga somente o processo responsável pelos *threads*.

Existem diferentes modelos para suporte a *threads*, tanto em nível de *kernel* quanto de usuário. De acordo com Tanenbaum e Woodhull (2002), os seguintes modelos são apresentados:

1. Modelo muitos-para-um
2. Modelo um-para-um

3. Modelo muitos-para-muitos

No modelo muitos-para-um (Figura 3), muitos *threads* no nível de usuário são mapeados para um único *thread* do *kernel*. Neste modelo, o gerenciamento dos *threads* é mais rápido; entretanto, se algum *thread* realizar alguma chamada de sistema de bloqueio, o processo inteiro será bloqueado. Em uma arquitetura com multiprocessadores, esse modelo não é capaz de executar em modo paralelo, uma vez que um único *thread* do *kernel* está disponível para atender os *threads* do nível de usuário. O modelo muitos-para-um geralmente é utilizado em Sistemas Operacionais que não fornecem suporte a *threads* em nível de *kernel*.

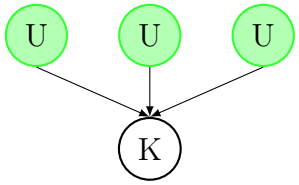


Figura 3: Modelo muitos-para-um

O modelo um-para-um (Figura 4) mapeia cada *thread* do usuário para um *thread* do *kernel*. Ele provê uma concorrência maior entre os *threads* do que o modelo muitos-para-um, visto que o bloqueio em função de alguma chamada de sistema, afetará somente o processo que fez a chamada, permitindo que os demais *threads* continuem sua execução. Entretanto, a criação excessiva de *threads* do usuário pode sobrecarregar o sistema. Este modelo se beneficia de uma arquitetura com multiprocessadores, já que vários *threads* podem estar executando no *kernel*. Uma desvantagem do modelo um-para-um está no fato de que a criação de um *thread* do usuário implica na criação de um *thread* no *kernel*. Desta maneira, o tempo necessário para a criação de um *thread* do usuário é incrementado com o *overhead* gasto para criação de um *thread* no *kernel*.

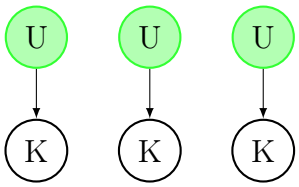


Figura 4: Modelo um-para-um

Por final, no modelo muitos-para-muitos (Figura 5), muitos *threads* do usuário são multiplexados para um número menor ou igual de *threads* no kernel. O número de *threads*

do *kernel* pode ser específico. Diferentemente do modelo um-para-um, a criação excessiva de *threads* no nível de usuário não sobrecarregará o Sistema Operacional, visto que o Sistema Operacional agendará esse *thread* para algum *thread* disponível no *kernel*. Os *threads* do *kernel* poderão executar em paralelo em uma arquitetura com multiprocessadores, assim como uma chamada de sistema de bloqueio de algum *thread* não bloqueará o processo inteiro.

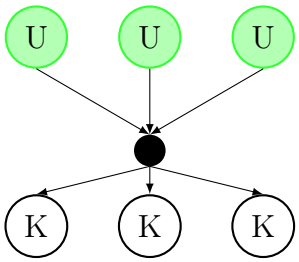


Figura 5: Modelo muitos-para-muitos

Neste trabalho foi utilizada a biblioteca de *threads* de usuário *Pthreads*. Esta biblioteca refere-se ao padrão *POSIX* IEEE 1003.1c, o qual define uma *Application Program Interface* – API para criação e sincronização dos *threads*. A implementação utilizada foi a fornecida pelo Sistema Operacional *Ubuntu GNU/Linux* 8.10, sendo esta API implementada pela biblioteca *GNU C Library*, mais comumente conhecida como *glibc*. O padrão para suporte a *threads* nesse sistema operacional é o modelo um-para-um (Figura 4), motivo pelo qual esse foi o modelo utilizado na execução dos experimentos.

2.8 Sistema de Processamento Paralelo

Um Sistema de Processamento Paralelo ou Distribuído é um conjunto de elementos que se comunicam através de uma rede de interconexão que tem, como um dos seus objetivos, melhorar o desempenho na execução de tarefas.

Uma das maneiras de se melhorar a eficiência na execução de tarefas é distribuir o processamento de uma tarefa entre os vários elementos que compõem o Sistema Paralelo. De acordo com Pitanga (2003), um Sistema Distribuído também pode ser classificado como um conjunto de computadores com o seu próprio *clock* e que não possuem memória compartilhada, sendo visto por seus usuários como um sistema de recurso único, ou imagem única do sistema (*Single System Image* – SSI). Estes computadores se comunicam entre si por troca de mensagens.

Uma outra representação de Sistema de Processamento Paralelo é o de Computador Paralelo ou Sistema Multiprocessado. Nesta arquitetura, uma coleção de elementos de processamento estão disponíveis na mesma máquina. Segundo Pitanga (2003), os Sistemas de Processamento Paralelo mais comuns podem ser caracterizados da seguinte maneira:

- Processadores Paralelos Massivos – MPP;
- Multiprocessadores Simétricos – SMP;
- Memória de Acesso Não-Uniforme com Coerência de Cache – CC-NUMA;
- Sistemas Distribuídos.

Um Sistema MPP é um Sistema de Processamento Paralelo com arquitetura de memória compartilhada e centralizada. Ele é composto por centenas de elementos processadores interconectados por uma rede de alta velocidade em um computador bem grande.

Os Sistemas SMP possuem de 2 a 64 processadores, os quais possuem uma visão global da memória. Cada processador tem acesso a toda memória através de um barramento dedicado. Sistemas SMP têm problemas quanto a escalabilidade, visto que a saturação do barramento e coerência na memória cache do processadores são problemas limitadores dessa arquitetura. O acesso a memória pelos processadores pode se dar de três formas: *Uniform Memory Access* – UMA, *Non Uniform Memory Access* – NUMA e *Cache-Only Memory Architecture* – COMA.

O CC-NUMA é um sistema de multiprocessadores escaláveis com arquitetura de coerência de *cache* com acesso não uniforme. Tal como o Sistema SMP, o Sistema CC-NUMA tem uma visão global da memória. Entretanto, a memória é dividida em tantos blocos quantos forem os processadores.

Os Sistemas Distribuídos ou Sistemas com Multicomputadores possuem vários computadores interconectados por uma rede de comunicação. Nestes sistemas, o modelo de memória é o *No Remote Memory Access* – NORMA, onde não existe um espaço de endereçamento global e cada processador tem sua memória local e a troca de dados é realizada por troca de mensagens. De acordo com Pitanga (2003), este modelo de memória pode gradualmente ser substituído pelo modelo de memória compartilhada distribuída – DSM. As redes de computadores podem ser vistas como arquiteturas NORMA; entretanto, se diferenciam quanto a velocidade e confiabilidade na interconexão.

O desenvolvimento de *softwares* paralelos é um dos grandes desafios de Sistemas de Processamento Paralelo, devido ao elevado tamanho e complexidade dos sistemas.

Contudo, o desenvolvimento de *softwares* em sistemas com multiprocessadores pode se beneficiar da API OpenMP com implementações disponíveis para as linguagens C, C++ e Fortran. Outro recurso para o desenvolvimento de *softwares* paralelos em sistemas multiprocessados é a utilização de *threads* (Vide Seção 2.7). Com a utilização de *threads*, os sistemas operacionais podem tirar proveito de uma arquitetura com vários elementos de processamento (vários processadores ou processadores *multicore*) distribuindo a execução dos *threads* entre os elementos de processamento disponíveis. Já para Sistemas com Multicomputadores podem ser utilizadas as APIs MPI e PVM, que disponibilizam rotinas para passagem de mensagens.

O método heurístico proposto neste trabalho (Vide Seção 5.2) foi executado em um sistema SMP com um processador de quatro núcleos. O algoritmo paralelo em questão foi desenvolvido com o uso de *threads* (Vide Seção 2.7).

3 CARACTERIZAÇÃO DO PROBLEMA

Este capítulo caracteriza o problema estudado neste trabalho. Na Seção 3.1 são apresentadas algumas variações conhecidas na literatura para o problema de sequenciamento de tarefas. A Seção 3.2 caracteriza detalhadamente o problema de sequenciamento abordado neste trabalho. É apresentado na Seção 3.3 o procedimento utilizado para calcular a melhor data de início das tarefas em dada sequência de produção.

3.1 Classificação dos Problemas de Sequenciamento

No estudo proposto por Allahverdi et al. (2008), foram revisados mais de trezentos artigos sobre problemas de sequenciamento. Nesse estudo, os autores classificaram os problemas da seguinte maneira:

1. **Sequenciamento em uma máquina (*single-machine*)**: Considera-se a existência de uma única máquina e todas as tarefas são processadas nesta máquina;

2. **Sequenciamento em máquinas paralelas (*parallel machines*)**: Considera-se a existência de m máquinas em paralelo. Estas máquinas poderão ser homogêneas, heterogêneas ou completamente não relacionadas. Cada tarefa poderá ser processada em qualquer uma das máquinas;

3. ***Flow shop***: Em um ambiente *flow shop* com m máquinas, considera-se a existência de m estágios em série, onde cada estágio poderá ser constituído por uma ou mais máquinas. Neste tipo de problema, cada tarefa deve ser processada em cada um dos m estágios na mesma ordem. Os tempos de preparação podem ser diferentes para cada tarefa nos diferentes estágios. O *flow shop* pode ser classificado da seguinte maneira:

(a) ***regular flow shop***: Considera-se a existência de uma única máquina em cada estágio;

(b) ***no-wait flow shop***: Neste problema, as operações para uma determinada tarefa são processadas de forma contínua do início ao fim. Não há interrupção em alguma das máquinas ou entre as máquinas;

(c) ***flexible (hybrid) flow shop***: Considera-se a existência de mais de uma máquina em cada estágio;

(d) ***assembly flow shop***: Em um ambiente *assembly flow shop* com dois estágios, considera-se a existência de $k + 1$ máquinas diferentes. Dadas n tarefas onde cada tarefa passa por $k+1$ operações, as k primeiras operações serão conduzidas ao primeiro estágio em paralelo, restando ao segundo estágio a operação final, também conhecida como operação de montagem(*assembly operation*);

4. ***Job shop***: Considera-se a existência de m máquinas diferentes e cada tarefa possui sua rota de processamento. Nesta rota, algumas máquinas podem não ser usadas e outras podem ser repetidas;

5. ***Open shop***: Este ambiente é semelhante ao *Job shop*. No entanto, cada tarefa deve ser processada uma única vez em cada uma das m máquinas seguindo qualquer rota de processamento.

O problema abordado no presente trabalho se encaixa no tipo de sequenciamento em uma máquina. De acordo com as definições de (ALLAHVERDI et al., 2008), ele pode ser denotado pela seguinte notação:

$$1/ST_{sd}/\sum E_j + \sum T_j \tag{3.1}$$

De acordo com a notação (3.1), primeiro elemento 1 representa sequenciamento em uma máquina, o elemento ST_{sd} representa tempo de preparação (***S**etup **T**ime*) dependente da sequência (***s**equence-**d**ependent*) e por final, o elemento $\sum E_j + \sum T_j$ representa o objetivo a ser minimizado, neste caso, representa a minimização do total da antecipação ($\sum E_j$) e do total do atraso ($\sum T_j$).

3.2 Descrição do Problema de Sequenciamento abordado

No presente trabalho, é estudada uma classe de problemas de sequenciamento em uma máquina com penalidades por atraso e antecipação da produção (PSUMAA). Essa classe de problemas, denotada por PSUMAA-JP, considera a existência de janelas de tempo desejadas para o processamento das tarefas, bem como de tempos de preparação da máquina dependentes da sequência de produção. O PSUMAA-JP possui as seguintes características:

(i) Uma máquina deve processar um conjunto de n tarefas;

(ii) Cada tarefa i possui um tempo de processamento P_i. O término desta tarefa i é delimitado pela janela de entrega $[E_i, T_i]$, onde E_i e T_i indicam respectivamente a data inicial e data final desejadas para o término de seu processamento. Se a tarefa i for finalizada antes de E_i, há um custo α_i por unidade de tempo de antecipação. Caso a tarefa seja finalizada após T_i, então há um custo β_i por unidade de tempo de atraso. Não há custo algum se as tarefas forem concluídas dentro da janela de entrega;

(iii) A máquina pode executar no máximo uma tarefa por vez. Seu funcionamento é não preemptível, ou seja, uma vez iniciado o processamento de uma tarefa, não é permitida a sua interrupção;

(iv) Todas as tarefas estão disponíveis para processamento na data 0;

(v) Entre duas tarefas i e j imediatamente consecutivas é necessário um tempo S_{ij} de preparação da máquina, chamado tempo de *setup*. Assume-se que a máquina não necessita de preparação para processamento da primeira tarefa, ou seja, a primeira tarefa que será processada na máquina tem tempo de preparação igual a 0;

(vi) É permitido tempo ocioso entre o processamento de duas tarefas imediatamente consecutivas;

A seguir, na Figura 6, é apresentado um esquema de sequenciamento que exemplifica graficamente alguns dos itens do PSUMAA-JP citados anteriormente.

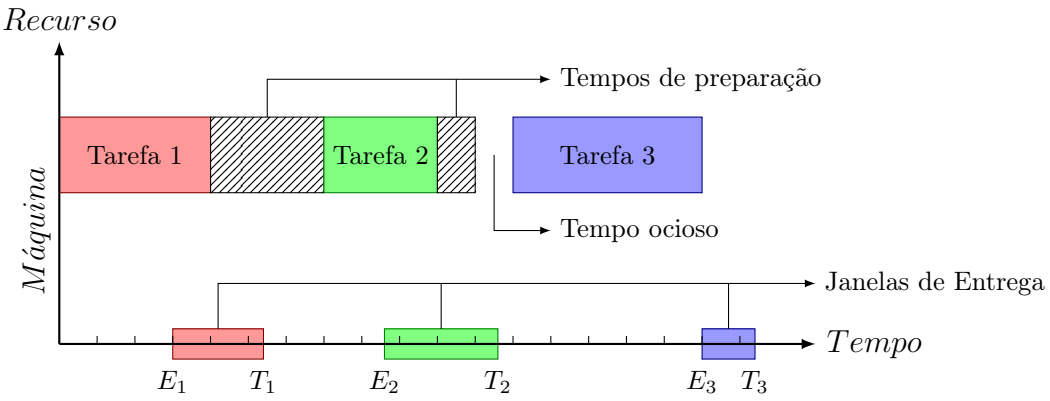


Figura 6: Exemplo PSUMAA

3.3 Procedimento de Datas Ótimas

O Procedimento de determinação da data ótima de início de processamento (PDDOIP) é utilizado para calcular a melhor data de início do processamento das tarefas pertencentes a uma sequência de produção. A sua execução necessita que a sequência de produção seja fornecida. Para este propósito, toma-se a sequência $V = \{J_1, J_2, \cdots, J_n\}$ composta por n tarefas.

Como descrito anteriormente (Seção 3.2), o problema abordado no presente trabalho possui uma janela de entrega desejada para cada tarefa da sequência, e com base nesta janela, o PDDOIP irá determinar a melhor data de início de processamento das tarefas.

O PDDOIP foi implementado em Gomes Jr. et al. (2007), o qual se baseou nos trabalhos de Wan e Yen (2002) e Lee e Choi (1995). Como o tempo de preparação é dependente da sequência de produção no PSUMAA-JP, o PDDOIP adicionou ao tempo de processamento de uma tarefa j imediatamente sucessora a uma tarefa i, o tempo de preparação dado por S_{ij}.

Considera-se s_k como a data de início de processamento da tarefa k e C_k como a data de término do processamento desta tarefa. Adota-se também o subconjunto de tarefas $B_j = \{J_1, J_2, ..., J_m\} \subseteq V$ como um bloco no sequenciamento, sendo um bloco entendido como uma sequência de tarefas realizadas consecutivamente sem tempo ocioso entre elas. Adicionalmente, assume-se a existência de l blocos em V e que $prim(B_j)$ e $ult(B_j)$ representem a primeira e a última tarefa no bloco B_j, respectivamente.

A primeira tarefa $J_1 \in V$ tem o seu processamento programado como se segue:

- Se $P_{J_1} \leq E_{J_1}$, então a tarefa J_1 é programada para ser finalizada na data E_{J_1}.
- Se $P_{J_1} > E_{J_1}$, então a tarefa J_1 é iniciada na data 0, sendo finalizada em P_{J_1}.

As demais tarefas são programadas da seguinte maneira:

- Se $C_{J_k} + S_{(J_k)(J_{k+1})} + P_{J_{k+1}} < E_{J_{k+1}}$, então a tarefa J_{k+1} tem seu processamento programado para ser finalizado na data $E_{J_{k+1}}$, isto é, a tarefa J_{k+1} é programada para ser finalizada no início da sua janela de entrega; desta forma, iniciando um novo bloco.
- Por outro lado, se $C_{J_k} + S_{(J_k)(J_{k+1})} + P_{J_{k+1}} \geq E_{J_{k+1}}$, então a tarefa J_{k+1} tem seu processamento programado para ser finalizado na data $C_{J_k} + S_{(J_k)(J_{k+1})} + P_{J_{k+1}}$ e é incluída como último elemento do bloco corrente.

A cada inclusão de uma tarefa $\in V$, todos os blocos formados durante este processo devem ser verificados. Esta verificação valida o posicionamento para o início da execução de cada bloco formado. O custo mínimo de um bloco B_j ocorre nos pontos extremos da sua função de custo, ou seja, no início ou no final da janela de entrega de uma das tarefas pertencentes a este bloco. Em virtude da natureza linear convexa por partes da função de custo, o custo mínimo pode ser obtido pela comparação dos somatórios das penalidades das tarefas pertencentes ao bloco, no início e no final de cada janela de entrega no bloco. As penalidades por antecipação são consideradas como valores negativos, enquanto as penalidades por atraso são consideradas como valores positivos. A partir destes somatórios são fornecidas as inclinações m das retas pertencentes à função de custo.

Como dito anteriormente, o custo mínimo de um bloco B_j se dá nos pontos extremos de sua função de custo, o que ocorre quando a inclinação m torna-se maior ou igual a zero. Neste caso, todo o bloco é movido em direção ao ponto mínimo até que um dos três casos a seguir aconteça:

1. $s_{prim(B_j)} = 0$.
2. O ponto mínimo é alcançado.
3. $s_{prim(B_j)}$ torna-se igual a $C_{ult(B_{j-1})} + S_{(ult(B_{j-1}))(prim(B_j))}$.

No último caso (3), a movimentação do bloco B_j resulta em uma união com o bloco B_{j-1}. Esta união implica em um novo bloco $B_{j-1} \leftarrow B_{j-1} \cup B_j$. Assim, é necessária a

obtenção do ponto mínimo no novo bloco B_{j-1}. O procedimento citado é realizado até que os casos (1) ou (2) ocorram para cada bloco.

Para exemplificar este procedimento, considera-se a Tabela 3, na qual o tempo de processamento é representado por P_i, o custo por antecipação da tarefa por α_i, o custo por atraso do processamento da tarefa por β_i, os tempos de preparação da máquina por S_{ij}, E_i a data inicial e T_i a data final desejadas para processamento da tarefa i. Essa Tabela, extraída de Gomes Jr. et al. (2007), é relativa a quatro tarefas.

Tabela 3: Dados de Entrada

						Preparação			
Tarefas	P_i	E_i	T_i	α_i	β_i	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

Na Tabela 3, por exemplo, a tarefa 3 demanda 4 unidades de tempo de processamento e deve ser realizada entre as datas 5 e 9. Caso seja finalizada antes do início da janela, há uma penalidade de 7 unidades; caso seja finalizada após o final da janela de entrega, há uma penalidade de 14 unidades. Caso na sequência de produção a tarefa 4 seja agendada para ser processada imediatamente após a tarefa 3, há a necessidade de 2 unidades de tempo para prepararação da máquina. Observe que, nesse exemplo, os tempos de preparação são simétricos.

Dada a sequência $v = \{3, 4, 1, 2\}$, o procedimento se inicia com a definição da data de conclusão da primeira tarefa (tarefa 3). Esta tarefa é programada para ser finalizada no início da sua janela de entrega, pois como descrito anteriormente, se $P_3 \leq E_3$, a tarefa é programada para terminar em $E_3 = 5$. O ponto mínimo do primeiro bloco é definido como $E_3 - P_3 = 1$, as inclinações m da reta pertencentes à função de custo são calculadas da seguinte forma: a penalidade por antecipação ($\alpha_3 = 7$) assume valor negativo e a penalidade por atraso ($\beta_3 = 14$) assume valor positivo. A programação da primeira tarefa, bem como as inclinações m, são mostradas na Figura 7.

Determina-se, agora, a programação da segunda tarefa (tarefa 4). Esta tarefa termina seu processamento em $C_4 = C_3 + S_{34} + P_4$. Ela é então adicionada no final do bloco corrente. Com o bloco corrente alterado, é necessário verificar o seu ponto mínimo. Para tal, as possíveis datas de início de processamento do bloco são calculadas posicionando cada tarefa pertencente ao bloco no início e no final da sua janela de entrega. Neste exemplo, inicialmente será utilizada a tarefa 4 para o cálculo das datas de início do bloco.

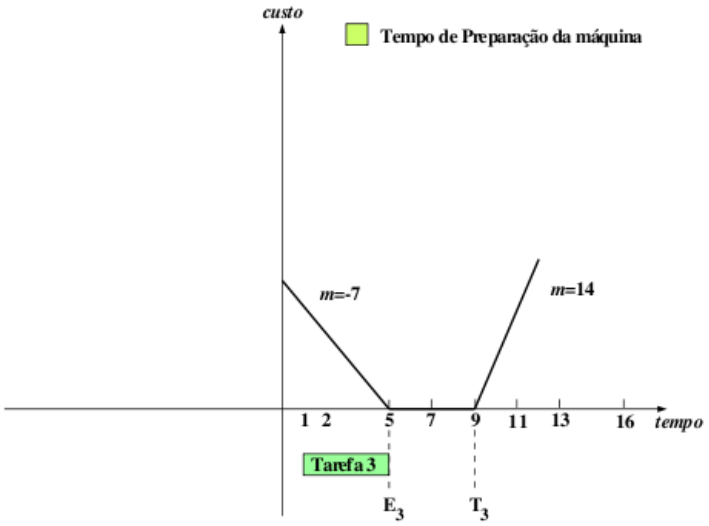
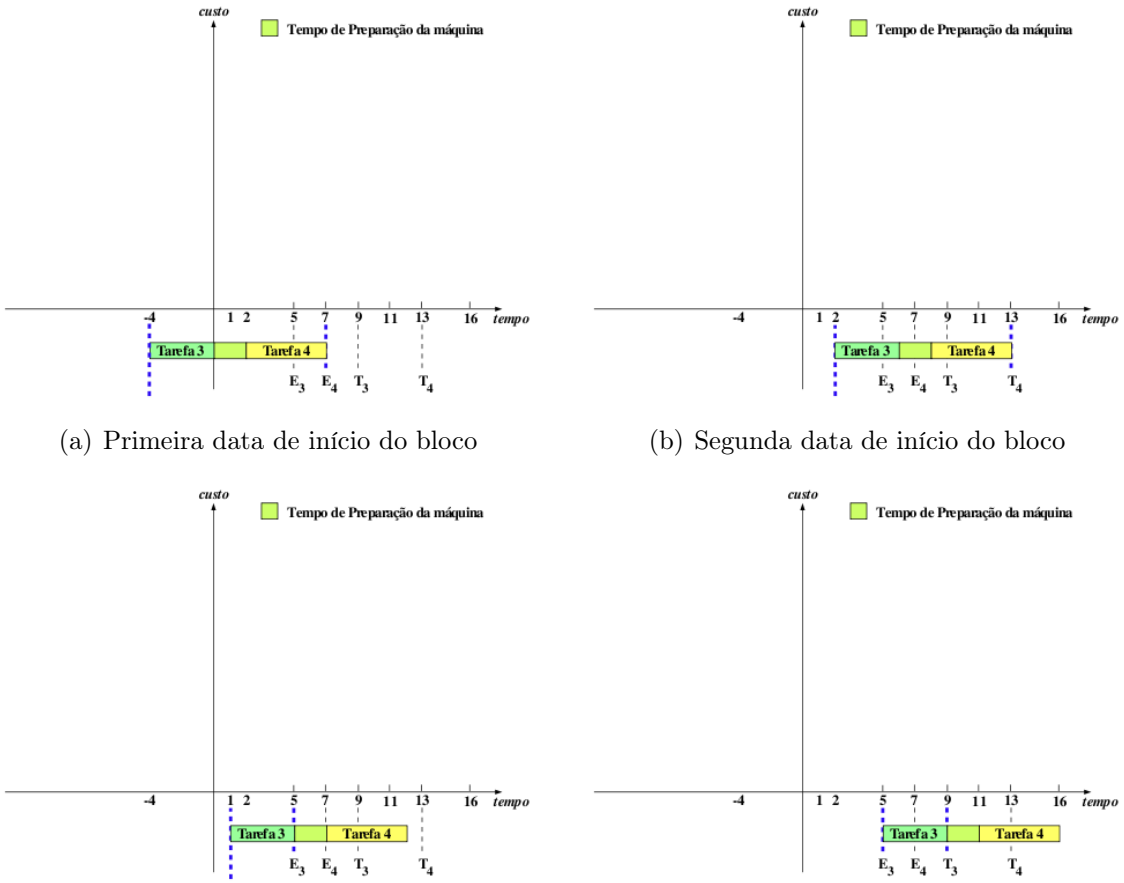


Figura 7: 1º Passo do Procedimento de Datas Ótimas

Posicionando a tarefa 4 no início da sua janela de entrega, ou seja, posicionando a tarefa na data 7, tem-se como resultado o bloco exibido na Figura 8(a). Verifica-se que o bloco deve começar o seu processamento no instante -4 para que a tarefa 4 seja posicionada na data 7. Desta maneira, a primeira data de início é dada no instante -4. Para a próxima data, a tarefa 4 é posicionada no final da sua janela de entrega, no caso, na data 13. O bloco resultante é mostrado na Figura 8(b). O posicionamento da tarefa na data 13 resulta no início do bloco no instante 2. A seguir, a tarefa 3 é selecionada para o cálculo das datas restantes. Posicionando-a na data inicial da sua janela de entrega (data 5), tem-se como resultado o bloco exibido na Figura 8(c). Assim, o bloco deve iniciar no instante 1 para que a tarefa esteja posicionada no início da sua janela de entrega. Por final, a tarefa 3 é posicionada na data final da sua janela de entrega, resultando no bloco mostrado na Figura 8(d). A data de início do bloco resultante do posicionamento da tarefa na data 9 é dada no instante 5. Este processo, resulta nas seguintes datas de início de processamento do bloco: -4, 2, 1 e 5.

As inclinações m da reta pertencente à função de custo são calculadas como se segue. Em instantes inferiores a -4, as duas tarefas estão adiantadas; desta forma, a inclinação é dada por $m = -7 - 8 = -15$. Para instantes dentro do intervalo -4 e 1, somente a tarefa 3 está adiantada; assim, a inclinação é dada por $m = -7$. Entre os instantes 1 e 2, a tarefa 3 deixa de estar adiantada; desta forma, a inclinação é dada por $m = 0$. A partir do instante 2, as tarefas passam a ficar atrasadas, passando a inclinação então a ser positiva. Até o instante 3 somente a tarefa 4 está atrasada, sendo a inclinação dada por $m = 16$. Por final, a partir do instante 3, as tarefas 3 e 4 tornam-se atrasadas, sendo



(a) Primeira data de início do bloco

(b) Segunda data de início do bloco

(c) Terceira data de início do bloco

(d) Quarta data de início do bloco

Figura 8: Cálculo das possíveis datas de início do bloco do Passo 2

a inclinação dada por $m = 14 + 16 = 30$. O ponto mínimo é obtido no ponto em que a inclinação m passa a ser maior ou igual a zero. Esta situação ocorre entre os instantes 1 e 2. Como o bloco está posicionado no instante 1, nenhuma alteração é necessária. A Figura 9 exibe esta programação.

Após a programação da tarefa 4, a próxima programação é da tarefa 1. Ambas as tarefas 3 e 4 terminam o seu processamento dentro da sua janela desejada. Entretanto, a tarefa 1 tem o seu processamento calculado para terminar após a sua janela de entrega no instante $C_1 = 17$. Como $C_4 + S_{41} + P_1 \geq E_1$, a tarefa 1 é inserida no final do bloco corrente (Figura 10).

Com a alteração do bloco, o ponto mínimo deve ser novamente calculado. Posicionando-se cada tarefa do bloco no início e no final da sua janela de entrega, as seguintes datas de início de processamento são obtidas: $-13, -11, -4, 1, 2$ e 5. A seguir, para definir o ponto mínimo do bloco, são calculadas a inclinações m da reta pertencente à função de

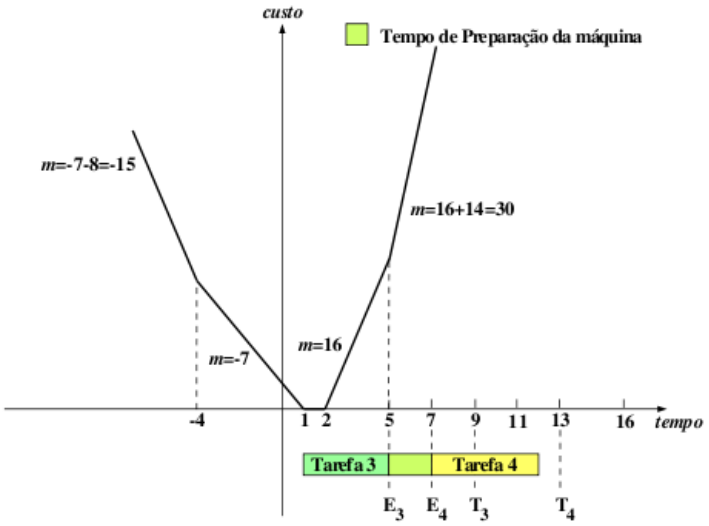


Figura 9: 2º Passo do Procedimento de Datas Ótimas

custo como se segue. Com o início do bloco antes do instante -13, todas as tarefas são antecipadas; assim, a inclinação é dada por $m = -7 - 8 - 9 = -24$. Entre os instantes -13 e -11, as tarefas 3 e 4 continuam adiantadas e a tarefa 1 deixa de ser antecipada, resultando assim na inclinação $m = -7 - 8 = -15$. Para instantes dentro do intervalo -11 e -4, as tarefas 3 e 4 são adiantadas e a tarefa 1 passa a ser atrasada; desta forma, a inclinação é dada por $m = -7 - 8 + 18 = 3$. Neste momento, a inclinação passa a ser maior ou igual a zero, resultando no ponto mínimo igual a -11. O bloco então deve ser movido na direção deste ponto, até que um dos casos citados anteriormente aconteça. Neste exemplo, a movimentação do bloco resulta em $s_{prim(j)} = 0$ (Caso 1). A programação desta tarefa é mostrada na Figura 11.

Já para a última tarefa 2, como $C_1 + S_{12} + P_2 \geq E_2$, ela é adicionada ao final do bloco corrente. Baseado nas possíveis datas de início de processamento do bloco, o novo ponto mínimo deve ser calculado. Este processo é repetido até que todas as tarefas sejam programadas, assim como todos os blocos sejam avaliados quanto ao seu ponto mínimo.

O procedimento para o PDDOIP é descrito no Algoritmo 8. De acordo com ele, o procedimento MudaBloco (Linha 26) movimenta os blocos quando necessário em acordo com os casos citados anteriormente.

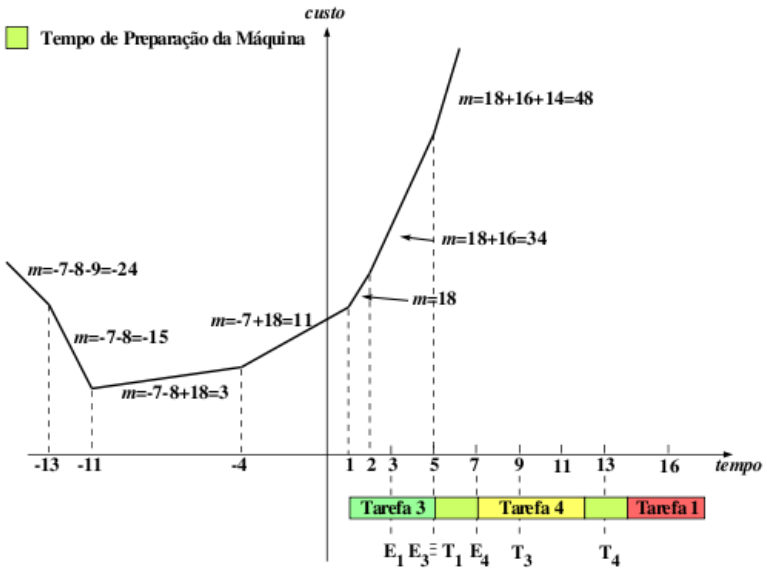


Figura 10: 3º Passo do Procedimento de Datas Ótimas: antes da aplicação

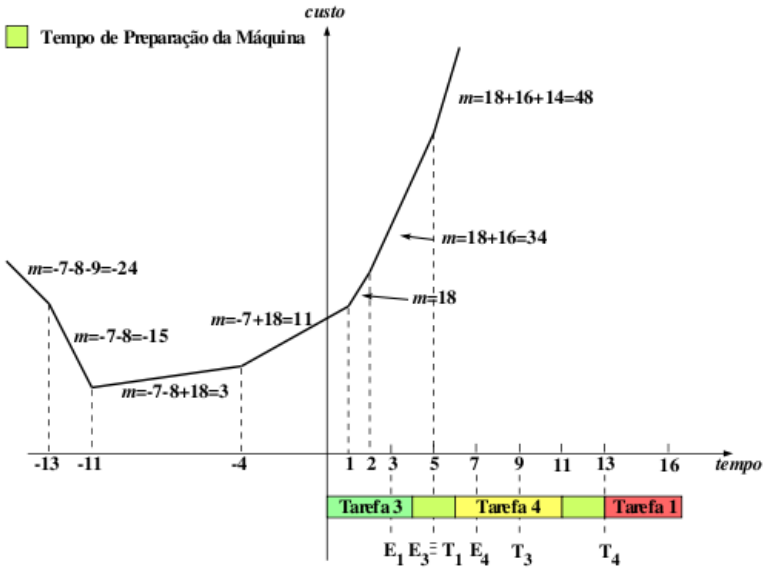


Figura 11: 3º Passo do Procedimento de Datas Ótimas: depois da aplicação

Algoritmo 8: PDDOIP

Entrada: n, v, $f(v)$
Saída: $f(v)$

início
  $B \leftarrow 1$ // Inicialização
  $prim(B) \leftarrow 1$
  $ult(B) \leftarrow 1$
  $s_1 \leftarrow \max\{0, E_1 - P_1\}$
  $C_1 \leftarrow \max\{E_1, P_1\}$ // Demais Tarefas
  para $i = 2$ **até** n **faça**
    se $(C_{i-1} + P_i + S_{(i-1)(i)} < E_i)$ **então**
      $B \leftarrow B + 1$
      $prim(B) \leftarrow i$
      $ult(B) \leftarrow i$
      $s_i \leftarrow E_i - P_i + S_{(i-1)(i)}$
      $C_i \leftarrow E_i$
    senão
      se $(C_{i-1} + P_i + S_{(i-1)(i)} = E_i)$ **então**
        $ult(B) \leftarrow i$
        $s_i \leftarrow C_{i-1} + S_{(i-1)(i)}$
        $C_i \leftarrow E_i$
      senão
        $ult(B) \leftarrow i$
        $s_i \leftarrow C_{i-1} + S_{(i-1)(i)}$
        $C_i \leftarrow s_i + P_i$
      fim
    fim
    repita
      $MudaBloco(B)$
    até ((*até todos os blocos estiverem no ponto mínimo*) *ou* $(s_{prim(1)} = 0)$)
  fim
  retorna $f(v)$
fim

4 REVISÃO DE LITERATURA

A produção de bens sob encomenda por meio da filosofia *just-in-time* tornou-se uma opção comumente adotada pelas empresas nos últimos anos. Em virtude disso, um planejamento criterioso da produção se faz necessário, visto que a antecipação ou atraso da produção poderão implicar em custos extras para as empresas. Dentre estes custos, podem ser citados: custos de armazenagem pela antecipação da produção e custos por multas contratuais devido ao atraso da produção.

O problema de sequenciamento em uma máquina visando a minimização das penalidades por antecipação e atraso da produção (PSUMAA) trata um dos casos mais simples dos problemas de planejamento da produção no contexto da filosofia *just-in-time*. Entretanto, este problema é difícil de ser resolvido na sua otimalidade com tempos computacionais aceitáveis, devido ao fato de pertencer à classe NP-difícil (DU; LEUNG, 1990; WAN; YEN, 2002).

O PSUMAA com datas de entrega comum tem sido estudado em vários trabalhos. Em (GORDON; PROTH; CHU, 2002) os autores produziram um estudo unificado sobre datas comuns de entrega relacionadas a problemas de sequenciamento em uma máquina e em máquinas paralelas. De acordo com o autor, a filosofia *just-in-time* foi fator determinante no estudo das datas de entrega em problemas de sequenciamento. Segundo esse autor, o trabalho de Jackson (1955) foi o ponto de partida do estudo. Também foram revisados alguns modelos para atribuição da data de entrega e apresentadas várias formulações para a função objetivo dos problemas de sequenciamento em uma máquina e em máquinas paralelas, além de algumas propriedades para datas comuns de entrega.

James (1997) propôs um algoritmo utilizando a metaheurística Busca Tabu para resolver o PSUMAA com data comum de entrega, penalidades pela antecipação e atraso da produção e sem permitir tempo ocioso entre as tarefas. As estruturas de vizinhanças utilizadas foram: troca entre tarefas adjacentes, troca entre duas tarefas quaisquer e inserção de uma tarefa à frente de outra tarefa qualquer. Segundo o autor, experimen-

tos demonstraram que a vizinhança baseada na inserção de tarefas produziram resultados melhores que a vizinhança com troca entre duas tarefas quaisquer e que a troca entre duas tarefas foi melhor que a troca envolvendo tarefas adjacentes. Contudo, movimentos híbridos baseados nas vizinhanças de inserção e troca entre duas tarefas quaisquer produziram melhores soluções. O método proposto utilizou duas técnicas para explorar o espaço de soluções. Na primeira, chamada *job space solution*, as soluções visitadas não satisfazem a propriedade *V-shaped*; enquanto na segunda, denominada *early/tardy solution space*, as soluções visitadas satisfazem essa propriedade.

Li (1997) e Liaw (1999) também trataram o PSUMAA com datas comuns de entrega, sem permitir tempo ocioso de máquina. O primeiro autor decompõe o problema em dois subproblemas com uma estrutura mais simples, de forma que o limite inferior do problema é a soma dos limites inferiores desses dois subproblemas. O limite inferior de cada subproblema é determinado por relaxação lagrangiana. Um algoritmo *branch-and-bound* é apresentado e usado para resolver instâncias de até 50 tarefas, dobrando a dimensão de problemas que podiam ser resolvidos na otimalidade com algoritmos exatos até aquela data. O autor propôs, também, procedimentos heurísticos baseados em busca local para resolver problemas de dimensões mais elevadas. No segundo trabalho é apresentado um algoritmo *branch-and-bound* que faz uso de procedimentos para determinar limites inferiores e superiores fortes. Regras de dominância são usadas para tentar eliminar nós não promissores na árvore de busca. O algoritmo obteve bom desempenho na resolução de problemas com até 50 tarefas. Para estudos futuros, o autor propôs desenvolver técnicas mais apuradas para as regras de dominância, visando diminuir as ramificações do algoritmo *branch-and-bound* e torná-lo viável na execução com problemas de mais de 50 tarefas.

Biskup e Feldmann (2001) trataram o PSUMAA com datas comuns de entrega e propuseram um gerador de problemas-teste, com o qual geraram um total de 280 instâncias. Foram apresentadas duas heurísticas para resolver estes problemas. Segundo os autores, o estudo também teve a intenção de utilizar os problemas-teste gerados para futuras comparações de desempenho com diferentes metodologias para resolução dos problemas.

Feldmann e Biskup (2003) resolveram problemas do PSUMAA com datas comuns de entrega (BISKUP; FELDMANN, 2001) por meio de três métodos: Algoritmo Evolutivo, *Simulated Annealing* – SA e uma versão aperfeiçoada da heurística *Threshold Accepting*, sendo esta última uma variante de *Simulated Annealing*. Segundo os autores, o método *Threshold Accepting* foi mais eficiente na obtenção de melhores soluções. O mesmo pro-

blema foi tratado por Hino, Ronconi e Mendes (2005), os quais apresentaram métodos baseados nas metaheurísticas Busca Tabu e Algoritmos Genéticos, bem como hibridizações destas. Nesse trabalho, o autor aponta uma melhor qualidade das soluções do Algoritmo Genético sobre a Busca Tabu, bem como uma similaridade dos resultados do Algoritmo Genético com o Algoritmo Híbrido.

YING (2008) trata do PSUMAA com datas comuns de entrega das tarefas e com tempo de *setup* de cada tarefa incluído em seu tempo de processamento e independente da sequência de produção. O problema é resolvido pelo algoritmo *Recovering Beam Search* – RBS, que é uma versão aperfeiçoada do algoritmo *Beam Search* – BS. Este, por sua vez, consiste em um algoritmo *branch-and-bound* em que somente os w nós mais promissores de cada nível da árvore de busca são retidos para ramificação futura, enquanto os nós restantes são podados permanentemente. Para evitar decisões equivocadas com respeito à poda de nós que conduzam à solução ótima, o algoritmo RBS utiliza uma fase de recobrimento que busca por soluções parciais melhores que dominem aquelas anteriormente selecionadas. O método proposto foi comparado com os de Feldmann e Biskup (2003), Hino, Ronconi e Mendes (2005), alcançando melhores resultados.

Rabadi, Mollaghasemi e Anagnostopoulos (2004) focaram no PSUMAA com datas comuns de entrega e tempo de preparação da máquina dependente da sequência de produção. Os autores apresentam um procedimento *branch-and-bound* que é capaz de resolver na otimalidade, em tempos aceitáveis, problemas-teste de até 25 tarefas, o que representava um avanço porque, até aquela data, os algoritmos exatos para essa classe de problemas eram capazes de resolver somente problemas-teste de até 8 tarefas.

O PSUM com penalidade por atraso e data de entrega distinta foi estudado por Bilge, Kurtulan e Kirac (2007), os quais utilizaram a metaheurística Busca Tabu para resolvê-lo. Para gerar a solução inicial os autores utilizaram vários métodos, entre eles as heurísticas *earliest due date* – EDD e *weighted shortest processing time* – WSPT. A BT desenvolvida usou uma estrutura de vizinhança híbrida com movimentos de troca e realocação; porém, os autores utilizaram uma estratégia para criação de uma lista de candidatos, se valendo de algumas propriedades do problema, para não permitir movimentos que não levariam a boas soluções. Foi também utilizado um tempo tabu (*tabu tenure*) dinâmico para evitar a ciclagem.

Lee e Choi (1995) aplicaram Algoritmos Genéticos – AG ao PSUMAA com datas de entrega distintas. Para determinar a data ótima de início do processamento de cada tarefa da sequência produzida pelo AG, desenvolveram um algoritmo específico, de complexidade

polinomial, que explora as características do problema.

Chang (1999) tratou o PSUMAA com datas distintas de entrega por meio de um algoritmo *branch-and-bound*. É analisado o desempenho desse algoritmo para resolver problemas com até 45 tarefas. Também foi desenvolvido um esquema para o cálculo de diferentes limites inferiores, o qual se baseia no procedimento de eliminação de sobreposição de uma sequência. Propriedades e teoremas sobre esse procedimento de eliminação de sobreposição são apresentados.

Gupta e Smith (2006) propuseram dois algoritmos, um baseado em GRASP e outro na heurística *space-based search*, para a resolução do problema de sequenciamento em uma máquina considerando a existência de data de entrega para cada tarefa e tempo de preparação de máquina dependente da sequência de produção, tendo como objetivo a minimização do tempo total de atraso. O algoritmo GRASP proposto foi dividido em três fases: Construção, Refinamento e Reconexão por Caminhos. A fase de refinamento do método GRASP baseou-se nas heurísticas *Variable Neighborhood Search* – VNS e *Variable Neighborhood Descent* – VND. Segundo os autores, a contribuição está em uma nova função custo para a fase de construção, uma nova variação do método VND para a fase de refinamento e uma fase de Reconexão por Caminhos usando diferentes vizinhanças.

Hallah (2007) tratou o PSUMAA com datas de entrega distintas, não sendo permitida a existência de tempo ocioso entre as tarefas. O autor propõe um método híbrido que combina heurísticas de busca local (regras de despacho, método da descida e *Simulated Annealing*) e um algoritmo evolucionário.

Wan e Yen (2002) apresentaram um algoritmo baseado na metaheurística Busca Tabu – TS para resolver o PSUMAA com janelas de entrega distintas. Para cada sequência de tarefas gerada pela Busca Tabu é acionado um procedimento de complexidade polinomial para determinar a data ótima de início do processamento de cada tarefa da sequência. Este último procedimento é uma adaptação daquele proposto em Lee e Choi (1995). Nesta adaptação, consideram-se janelas de entrega no lugar de datas de entrega.

Uma formulação de programação linear inteira mista para o PSUMAA-JP é desenvolvida em Gomes Jr. et al. (2007). O modelo desenvolvido foi utilizado para resolver na otimalidade problemas de até 12 tarefas. Esta modelagem serviu para comparar os resultados obtidos por um algoritmo heurístico baseado em GRASP, *Iterated Local Search* (ILS) e *Variable Neighborhood Descent* (VND), também proposto pelos autores. Para cada sequência gerada pela heurística, é acionado um algoritmo para determinar a data ótima de conclusão do processamento de cada tarefa. Tal algoritmo de datas ótimas foi

adaptado de Wan e Yen (2002), e inclui no tempo de processamento de uma tarefa o tempo de preparação da máquina, já que quando ele é acionado, já se conhece a sequência de produção. O algoritmo desenvolvido pelos autores, nomeado GILS-VND-RT, foi capaz de encontrar todas as soluções ótimas conhecidas para os problemas com 8 a 12 tarefas.

Rosa (2009) desenvolveu dois modelos de programação linear inteira mista para o PSUMAA-JP. Dentre as duas novas formulações propostas nesse trabalho, uma delas se trata de um aperfeiçoamento do modelo proposto por Gomes Jr. et al. (2007), e a outra, uma formulação indexada no tempo. Foi proposto, também, um algoritmo heurístico de duas fases baseado em GRASP, VND e no princípio da otimalidade próxima. Segundo o autor, a formulação indexada no tempo possibilitou a obtenção de um maior número de soluções ótimas, bem como um menor esforço computacional gasto na resolução dos problemas. Já o algoritmo heurístico, nomeado GPV, obteve tempos computacionais inferiores aos resultados da literatura e se mostrou competitivo com os algoritmos existentes até então.

Ribeiro (2009) propôs um Algoritmo Genético Adaptativo, denominado AGA, para a resolução do PSUMAA-JP. A população inicial é gerada a partir da fase de construção GRASP, tendo como função guia várias regras de despacho. Para cada indivíduo gerado é utilizado o algoritmo PDDOIP de Gomes Jr. et al. (2007) para determinar a data ótima de início de processamento de cada tarefa na sequência dada. São utilizados cinco operadores de cruzamento e um de mutação. Os operadores melhor sucedidos em gerações passadas têm maior probabilidade de serem escolhidos. O algoritmo também aplica periodicamente a Reconexão por Caminhos visando a encontrar soluções intermediárias de melhor qualidade.

Souza et al. (2008) desenvolveram um algoritmo híbrido sequencial de três fases baseado em GRASP, VND, Busca Tabu e Reconexão por caminhos para o PSUMAA-JP. Na primeira fase foi utilizado um procedimento baseado em GRASP para gerar a solução inicial. As soluções geradas neste método são refinadas por um procedimento baseado em VND. Na segunda fase, a melhor solução construída pelo procedimento GRASP é então refinada por um procedimento baseado em Busca Tabu. Durante a execução da Busca Tabu, um conjunto de soluções denominado conjunto elite é formado. Na terceira e última fase, como estratégia de pós-otimização é aplicado o procedimento Reconexão por Caminhos aos pares de solução do conjunto elite.

Para uma revisão abrangente de vários estudos sobre problemas de sequenciamento com tempos de preparação, apontam-se os trabalhos de Allahverdi, Gupta e Aldowaisan

(1999), Allahverdi et al. (2008).

Com relação à utilização de algoritmos paralelos, de nosso conhecimento, nenhuma aplicação ainda foi desenvolvida para tratar o problema em questão. Contudo, alguns trabalhos que utilizam algoritmos paralelos aplicados a outros problemas são citados a seguir.

Ribas et al. (2009b) apresentaram um *framework* que implementa a abstração *MapReduce* para a linguagem C++. A utilização deste *framework* torna o desenvolvimento de aplicações paralelas mais simples, uma vez que não é requerido ao desenvolvedor nenhum conhecimento do mecanismo de paralelização do problema em estudo. Para validar o *framework* proposto, foi desenvolvido um algoritmo paralelo para resolver o Problema do Caixeiro Viajante – PCV. Segundo os autores, os resultados comprovaram a eficiência desta ferramenta.

Ribas et al. (2009a) desenvolveram um algoritmo paralelo para resolver o problema de planejamento operacional de lavra com alocação dinâmica de caminhões. O algoritmo desenvolvido combina a paralelização do *Iterated Local Search* – ILS com os procedimentos GRASP e *Variable Neighborhood Descent*. Foram utilizados oito movimentos para explorar o espaço de soluções. Os resultados produzidos foram comparados aos de um modelo de programação linear inteira mista resolvido pelo otimizador CPLEX e um procedimento heurístico sequencial baseado em ILS, GRASP e VND (COELHO; RIBAS; SOUZA, 2008). Segundo os autores, houve uma melhoria na qualidade das suas soluções finais quando comparadas às obtidas pela versão sequencial do algoritmo.

Garcia et al. (2001) desenvolveram um algoritmo memético paralelo aplicado ao problema de sequenciamento em uma máquina com datas de entrega distintas e tempo de preparação da máquina dependente da produção. Foram revisados alguns modelos clássicos de algoritmos evolutivos paralelos e a estrutura geral dos algoritmos meméticos. O algoritmo proposto foi baseado em Algoritmos Evolutivos Paralelos Globais – AEPG. Nessa implementação, utilizou-se programas mestre-escravo. Foram realizados experimentos computacionais com 8 problemas-teste, sendo 4 com 71 tarefas e outros 4 com 100. Observou-se um ganho de desempenho com a utilização de mais processadores.

Standerski (2003) resolveu o problema de roteirização com reabastecimento e entregas diretas (*Inventory Routing Problem with Direct Deliveries* – IRPDD). Foram consideradas algumas simplificações do modelo, mas segundo o autor a aplicabilidade não foi descaracterizada. Utilizou-se sistemas computacionais paralelos de baixo custo, também conhecidos como *beowulf clusters* e a reposição do estoque gerenciada pelo fornecedor (*Vendor Ma-*

naged Inventory – VMI). Foi proposto um algoritmo genético paralelo baseado em ilhas com operadores de migração. O algoritmo foi implementado em Fortran e MPI e testado num *beowulf cluster* de 6 nós. Foram obtidas acelerações quase lineares no tempo de processamento.

Resende e Ribeiro (2005) descreveram estratégias simples para paralelização de algoritmos baseados em GRASP. Nesse trabalho, os autores categorizaram a estratégia de paralelização de duas formas: *multiple-walk independent-thread* e *multiple-walk cooperative-thread*. Foi proposto um algoritmo híbrido baseado em GRASP e Reconexão por Caminhos. O algoritmo proposto foi testado com 3 problemas-teste da literatura utilizando-se as duas estratégias de paralelização citadas anteriormente. Segundo os autores, a estratégia *multiple-walk cooperative-thread* levou a melhores resultados neste algoritmo híbrido, tornando-o mais rápido e robusto.

Muhammad (2006) apresentou um algoritmo de busca local paralelo para computar uma árvore de *Steiner* em um plano bidimensional. Como vantagem deste algoritmo, não se tem o *overhead* gerado por parte da comunicação entre os processos. Segundo o autor, a busca local paralela permitiu resolver problemas maiores e melhorar a qualidade das soluções finais.

Campos, Yoshizaki e Belfiore (2006) propuseram a utilização de metaheurísticas e computação paralela para a resolução de um problema real de roteamento de veículos com frota heterogênea, janelas de tempo e entregas fracionadas. Nesse problema, a demanda dos clientes pode ser maior que a capacidade dos veículos. A solução do problema consiste na determinação de um conjunto de rotas econômicas que devem atender a necessidade de cada cliente respeitando várias restrições descritas no trabalho. O algoritmo proposto utilizou uma adaptação da heurística construtiva de Clarke e Wright (1964) como solução inicial e para refinamento a heurística Meta-RaPS (MORAGA et al., 2001). Posteriormente, implementa-se um algoritmo genético paralelo que é resolvido com o auxílio de um *cluster* de computadores, com o objetivo de explorar novos espaços de soluções. A estratégia adotada para o algoritmo genético paralelo é a de ilhas com a utilização de operadores de migração. Os resultados obtidos demonstram que a heurística construtiva básica apresenta resultados satisfatórios para o problema, mas pode ser melhorada substancialmente com o uso de técnicas mais sofisticadas. A aplicação do algoritmo genético paralelo de múltiplas populações proporcionou um redução no custo total da operação na ordem de 10%, em relação à heurística construtiva e 13%, quando comparada às soluções utilizadas originalmente pela empresa.

4.0.1 Considerações Finais

Na revisão bibliográfica deste trabalho, apresentou-se uma série de artigos publicados nos principais periódicos e anais de eventos científicos, bem como dissertações abordando problemas de sequenciamento. Foram revisados trabalhos de sequenciamento em uma máquina com datas comuns de entrega, datas de entrega distintas e com janelas de entrega distintas. Ao final da revisão, um estudo sobre trabalhos com processamento paralelo, aplicados a outras classes de problemas, também foi apresentado. Diante da revisão apresentada, ressalta-se que nenhum dos trabalhos estudados explorou o uso de processamento paralelo na resolução do problema de sequenciamento tratado no presente trabalho.

5 METODOLOGIA

Neste capítulo é apresentada a metodologia proposta para resolver o PSUMAA-JP. Na Seção 5.1 apresentam-se três formulações de programação matemática para o problema, enquanto na Seção 5.2 é descrito um algoritmo heurístico.

5.1 Formulação de Programação Matemática

Nesta seção são apresentados três modelos matemáticos para resolução do PSUMAA-JP: MPLIM-G, MPLIM-BG e MPLIM-IT. O primeiro modelo foi proposto por Gomes Jr. et al. (2007), o segundo e o terceiro modelo foram propostos por Rosa (2009). Ambos os modelos foram desenvolvidos utilizando programação linear inteira mista (PLIM), sendo o terceiro modelo uma formulação matemática indexada no tempo. Devido à alta complexidade do PSUMAA-JP, os modelos propostos foram utilizados para resolver problemas-teste com até 12 tarefas. Todos os problemas-teste foram resolvidos com o otimizador CPLEX, versão 9.1.

5.1.1 MPLIM-G

Este modelo foi proposto por Gomes Jr. et al. (2007). Para auxílio da modelagem são utilizadas duas tarefas fictícias, 0 (zero) e $n+1$. A tarefa 0 antecede imediatamente a primeira tarefa da sequência e a tarefa $n+1$ sucede imediatamente a última tarefa na sequência de produção. Adota-se também uma constante M suficientemente grande. Os valores de P_0 e P_{n+1} são iguais a zero, bem como $S_{0i} = 0$ e $S_{i,n+1} = 0$, $\forall i = 1, \cdots, n$. Considera-se ainda s_i como a data de início do processamento da tarefa i, e_i o tempo de antecipação da tarefa i e t_i o tempo de atraso da tarefa i. As variáveis binárias y_{ij} determinam a sequência de produção e são definidas como se segue:

$$y_{ij} = \begin{cases} 1, & \text{se a tarefa } j \text{ é sequenciada imediatamente após a tarefa } i; \\ 0, & \text{caso contrário.} \end{cases}$$

A formulação proposta pelos autores é descrita a seguir:

minimizar:

$$z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{5.1}$$

sujeito a:

$$\begin{aligned}
s_j - s_i - y_{ij}(M + S_{ij}) &\geq P_i - M && \forall i = 0, 1, \cdots, n && (5.2)\\
& && \forall j = 1, 2, \cdots, n+1 \text{ e } i \neq j \\
\sum_{i=0, i\neq j}^{n} y_{ij} &= 1 && \forall j = 1, 2, \cdots, n+1 && (5.3)\\
\sum_{j=1, j\neq i}^{n+1} y_{ij} &= 1 && \forall i = 0, 1, \cdots, n && (5.4)\\
s_i + P_i + e_i &\geq E_i && \forall i = 1, 2, \cdots, n && (5.5)\\
s_i + P_i - t_i &\leq T_i && \forall i = 1, 2, \cdots, n && (5.6)\\
s_i &\geq 0 && \forall i = 0, 1, \cdots, n+1 && (5.7)\\
e_i &\geq 0 && \forall i = 1, 2, \cdots, n && (5.8)\\
t_i &\geq 0 && \forall i = 1, 2, \cdots, n && (5.9)\\
y_{ij} &\in \{0, 1\} && \forall i, j = 0, 1, \cdots, n+1 \text{ e } i \neq j && (5.10)
\end{aligned}$$

O PSUMAA-JP tem como objetivo a minimização de custos. De acordo com este modelo, a função objetivo representada pela equação (5.1) tem como critério de otimização a minimização do custo total de antecipação e atraso de uma sequência de produção. A seguir, as restrições (5.2) não permitem que a tarefa sucessora j comece antes que a tarefa predecessora i tenha terminado. Desta maneira, têm-se a garantia de que uma tarefa i tenha tempo suficiente para terminar antes que a tarefa j sequenciada imediatamente após i comece a ser processada. As variáveis y_{ij} definem a sequência de produção, desta maneira, as restrições (5.3) e (5.4) garantem que cada tarefa tenha somente uma tarefa imediatamente antecessora e uma tarefa imediatamente sucessora, respectivamente. Assim, quando $y_{ij} = 1$, as restrições (5.2) são reduzidas a $s_j \geq s_i + S_{ij} + P_i$, ou seja, a tarefa j só pode ser iniciada após a tarefa predecessora i ser processada e a máquina ser preparada para a tarefa j. Quando $y_{ij} = 0$, as restrições (5.2) são reduzidas a $s_j \geq s_i + P_i - M$, e portanto desprezível, dado que M é uma constante com o valor arbitrariamente grande. As restrições (5.5) e (5.6) definem a antecipação e o atraso de cada tarefa respeitando a janela de entrega de cada tarefa. As restrições (5.7), (5.8), (5.9) e (5.10) definem o tipo

das variáveis do problema.

5.1.2 MPLIM-BG

O modelo MPLIM-BG (ROSA, 2009) é baseado no modelo da Subseção 5.1.1. Entretanto, neste modelo apenas uma tarefa fictícia 0 (zero) é utilizada para auxiliar a modelagem. Esta tarefa é sequenciada duas vezes, antecedendo imediatamente a primeira tarefa e sucessedendo imediatamente a última tarefa. O valor de P_0 é igual a zero, bem como $S_{0i} = 0$ e $S_{i0} = 0$, $\forall i = 1, \cdots, n$. As variáveis $y_{i,n+1}$ e $y_{n+1,i} \forall i = 0, 1, \cdots, n$, assim como, s_{n+1} não são mais utilizadas neste modelo e de acordo com Rosa (2009), a redução destas variáveis implica numa economia de $2n + 3$ variáveis e $n + 1$ restrições, onde n representa o número de tarefas a serem sequenciadas. Tal como no MPLIM-G, considera-se s_i como a data de início do processamento da tarefa i, e_i o tempo de antecipação da tarefa i, t_i o tempo de atraso da tarefa i e as variáveis binárias y_{ij} determinam a sequência de produção e são definidas como se segue:

$$y_{ij} = \begin{cases} 1, & \text{se a tarefa } j \text{ é sequenciada imediatamente após a tarefa } i; \\ 0, & \text{caso contrário.} \end{cases}$$

A formulação é apresentada a seguir:

minimizar:

$$z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{5.11}$$

sujeito a:

$$
\begin{aligned}
s_j - s_i - y_{ij}(M + S_{ij}) &\geq P_i - M && \forall i = 0, 1, \cdots, n && (5.12)\\
& && \forall j = 1, 2, \cdots, n \text{ e } i \neq j \\
\sum_{i=0, i\neq j}^{n} y_{ij} &= 1 && \forall j = 0, 1, \cdots, n && (5.13)\\
\sum_{j=0, j\neq i}^{n} y_{ij} &= 1 && \forall i = 0, 1, \cdots, n && (5.14)\\
s_i + P_i + e_i &\geq E_i && \forall i = 1, 2, \cdots, n && (5.15)\\
s_i + P_i - t_i &\leq T_i && \forall i = 1, 2, \cdots, n && (5.16)\\
s_i &\geq 0 && \forall i = 0, 1, \cdots, n && (5.17)\\
e_i &\geq 0 && \forall i = 1, 2, \cdots, n && (5.18)\\
t_i &\geq 0 && \forall i = 1, 2, \cdots, n && (5.19)\\
y_{ij} &\in \{0, 1\} && \forall i, j = 0, 1, \cdots, n \text{ e } i \neq j && (5.20)
\end{aligned}
$$

Tal como descrito na Subseção 5.1.1 a função objetivo representada pela equação (5.11) tem como critério de otimização a minimização do custo total de antecipação e atraso de uma sequência de produção. As restrições (5.12) garantem que a tarefa i tenha tempo suficiente para terminar antes que a tarefa j sequenciada imediatamente após i comece a ser processada. As variáveis y_{ij} são restringidas pelas equações (5.13) e (5.14) que garantem que cada tarefa tenha somente uma tarefa imediatamente antecessora e uma tarefa imediatamente sucessora, respectivamente. Novamente, quando $y_{ij} = 1$, as restrições (5.12) são reduzidas a $s_j \geq s_i + S_{ij} + P_i$, ou seja, a tarefa j só pode ser iniciada após a tarefa predecessora i ser processada e a máquina ser preparada para a tarefa j. Já para $y_{ij} = 0$, as restrições (5.12) são reduzidas a $s_j \geq s_i + P_i - M$, e portanto desprezível, dado que M é uma constante com o valor arbitrariamente grande. Por final, as restrições (5.15) e (5.16) definem a antecipação e o atraso de cada tarefa respeitando a janela de entrega de cada tarefa e as restrições (5.17), (5.8), (5.19) e (5.20) definem o tipo das variáveis do problema.

5.1.3 MPLIM-IT

Nesta subseção é apresentado o MPLIM-IT (ROSA, 2009), um modelo de programação linear inteira mista indexado no tempo. Assim como em (de Paula, 2008), este modelo utiliza a discretização do tempo para modelar o PSUMAA-JP. Esta discretização é dada

por $H = \{h_0, h_1, h_2, \ldots, h_L\}$, como um conjunto de possíveis datas de início de processamento para a lista de tarefas a serem sequenciadas. Considera-se ainda e_i como o tempo de antecipação da tarefa i e t_i o tempo de atraso da tarefa i. As variáveis binárias x_{ih} definem a sequência de produção e são definidas como se segue:

$$x_{ih} = \begin{cases} 1, & \text{se a tarefa } i \text{ é sequenciada para iniciar na data } h; \\ 0, & \text{caso contrário.} \end{cases}$$

A formulação MPLIM-IT é definida seguir:

minimizar:

$$z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{5.21}$$

sujeito a:

$$x_{ih} + \sum_{u=h}^{\min(h+P_i+S_{ij}-1, H_L)} x_{ju} \leq 1 \qquad \forall i = 1, 2, \cdots, n \tag{5.22}$$
$$\forall j = 1, 2, \cdots, n$$
$$\forall\, h \in H \quad \text{e} \quad i \neq j$$

$$\sum_{h=H_0}^{H_L} x_{ih} = 1 \qquad \forall i = 1, 2, \cdots, n \tag{5.23}$$

$$\sum_{h=H_0}^{H_L} x_{ih} h + P_i + e_i \geq E_i \qquad \forall i = 1, 2, \cdots, n \tag{5.24}$$

$$\sum_{h=H_0}^{H_L} x_{ih} h + P_i - t_i \leq T_i \qquad \forall i = 1, 2, \cdots, n \tag{5.25}$$

$$e_i \geq 0 \qquad \forall i = 1, 2, \cdots, n \tag{5.26}$$

$$t_i \geq 0 \qquad \forall i = 1, 2, \cdots, n \tag{5.27}$$

$$x_{ih} \in \{0, 1\} \quad \forall i = 1, 2, \cdots, n \quad \text{e} \quad \forall\, h \in H \tag{5.28}$$

A função objetivo representada pela equação (5.21) tem como critério de otimização a minimização do custo total de antecipação e atraso de uma sequência de produção. As restrições (5.22) garantem que uma tarefa i tenha tempo suficiente para terminar antes que a tarefa j sequenciada imediatamente após i comece a ser processada e que a máquina seja preparada para o processamento da tarefa j. As restrições (5.23) garantem que cada tarefa seja executada uma única vez. As restrições (5.24) e (5.25) definem a antecipação e o atraso de cada tarefa respeitando a janela de entrega de cada tarefa. As restrições

(5.26), (5.27) e (5.28) definem o tipo das variáveis do problema.

De acordo com Rosa (2009), dada uma sequência ótima de tarefas, pressupõe-se que a primeira tarefa não ocorra antes de h_{inf} e que a última tarefa termine no máximo até h_{sup}. Sendo assim, pode-se tomar $H = \{h_{inf}, h_{inf}+1, h_{inf}+2, \ldots, h_{sup}\}$. Este modelo é muito sensível à cardinalidade de H; desta maneira, à medida que o intervalo $[h_{inf}, h_{sup}]$ diminuir, um número menor de variáveis utilizadas na discretização do tempo será necessário. Diferentemente das formulações MPLIM-G e MPLIM-BG, esta formulação é válida somente se os tempos de preparação do problema atenderem a desigualdade triangular definida pelas restrições 5.29:

$$\begin{aligned} S_{ik} \quad \leq \quad S_{ij} + S_{jk} + P_j \qquad & \forall i = 1, 2, \cdots, n \\ & \forall j = 1, 2, \cdots, n \\ & \forall k = 1, 2, \cdots, n, \\ & i \neq j,\ i \neq k \text{ e } j \neq k \end{aligned}$$

5.2 Modelagem Heurística

Apresenta-se, a seguir, a modelagem heurística do PSUMAA-JP. Na Seção 5.2.1 descreve-se o algoritmo heurístico proposto. As seções seguintes detalham seus módulos.

5.2.1 Algoritmo Proposto

Para resolver o PSUMAA-JP, propõe-se um algoritmo heurístico de três fases, nomeado GTSPR-M, o qual é apresentado no Algoritmo 9.

Algoritmo 9: GTSPR-M

Entrada: $N_{processors}$, Γ, *PrazoTABU*, *difElite*, *GRASPmax*, *MDRmax*, *TABUmax*
Saída: v

```
1 início
2 |  T ← ∅
3 |  CE ← ∅
4 |  v0 ← GRASP-M(N_processors, Γ, GRASPmax, MDRmax)
5 |  v1 ← TABU-M(N_processors, v0, PrazoTABU, T, difElite, CE, TABUmax, MDRmax)
6 |  v ← PR-M(N_processors, v1, CE)
7 |  Retorne v
8 fim
```

Na primeira fase (linha 4), constrói-se uma solução inicial (vide Seção 5.2.5). A seguir,

na linha 5, a solução resultante dessa fase é refinada por um algoritmo baseado em Busca Tabu (vide Seção 5.2.6). Como pós-otimização, linha 6, é aplicada a Reconexão por Caminhos (vide Seção 5.2.7).

5.2.2 Representação de uma Solução

Uma solução do PSUMAA-JP é representada por um vetor v de n elementos (tarefas), onde cada posição $i = 1, 2, \ldots, n$ de v indica qual é a i-ésima tarefa da sequência a ser processada. Desta maneira, na sequência $v = \{4, 3, 5, 6, 2, 1\}$, a tarefa 4 é a primeira a ser realizada e a tarefa 1, a última.

5.2.3 Vizinhança de uma Solução

A exploração do espaço de solução é feita utilizando-se três tipos de movimentos: troca da ordem de processamento de duas tarefas da sequência de produção, realocação de uma tarefa para outra posição na sequência de produção e realocação de um bloco de k tarefas, $2 \leq k \leq n-2$. Esses movimentos definem, respectivamente, as estruturas de vizinhança N^T, N^R e N^{Or}.

Considerando a vizinhança N^T e dado um conjunto com n tarefas, há $n(n-1)/2$ vizinhos possíveis para esta vizinhança. Como exemplo, considere a solução $v' = \{5, 3, 4, 6, 2, 1\}$, esta solução é um vizinho de v segundo a vizinhança N^T, pois a solução é obtida com a troca da tarefa 4 na primeira posição de v, com a tarefa 5, que está na terceira posição de v. A troca das tarefas é exibida na na Figura 12.

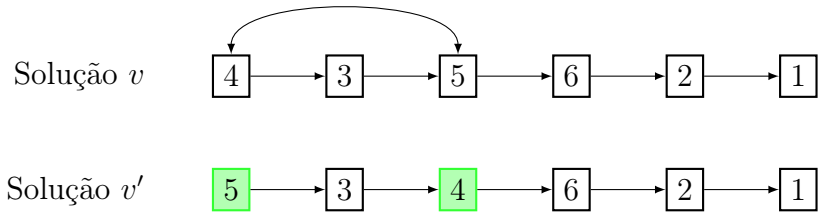


Figura 12: Movimento de troca - N^T

Já para a segunda estrutura de vizinhança, N^R, existem $(n-1)^2$ vizinhos possíveis. Por exemplo, a solução $v = \{3, 5, 6, 4, 2, 1\}$ é vizinha de v segundo a vizinhança N^R, visto que, esta solução é obtida através da realocação da tarefa 4 para depois da tarefa 6 na sequência de produção v. A realocação é mostrada na Figura 13.

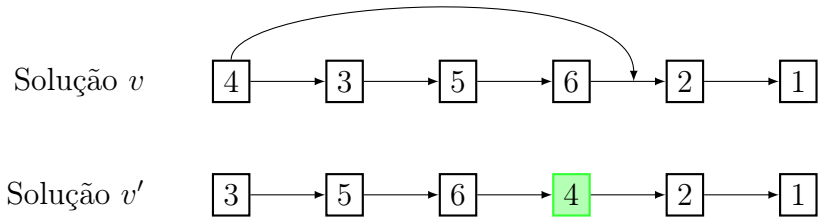


Figura 13: Movimento de realocação - N^R

Para a terceira e última estrutura de vizinhança, N^{Or}, há $(n-k-1)^2$ vizinhos possíveis. Por exemplo, a solução $v' = \{5, 4, 3, 6, 2, 1\}$ é vizinha de v segundo a vizinhança N^{Or}, pois é obtida pela realocação do bloco de tarefas 4 e 3, onde $k = 2$, para depois da tarefa 5 na sequência de produção v. A realocação do bloco é mostrada na Figura 14. Os movimentos de realocação segundo as vizinhanças N^R e N^{Or} podem ser feitos tanto para posições sucessoras quanto antecessoras àquela em que a tarefa ou bloco de tarefas se encontram na sequência de produção.

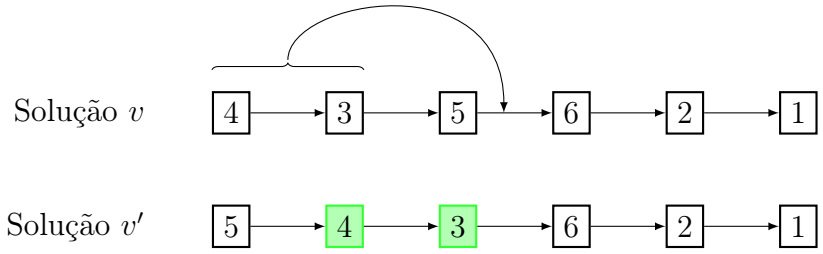


Figura 14: Movimento de realocação de um bloco - N^{Or}

5.2.4 Função de Avaliação

Uma solução v é avaliada pela função f a seguir, a qual deve ser minimizada:

$$f(v) = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{5.29}$$

em que e_i e t_i $(e_i, t_i \geq 0)$ indicam, respectivamente, o tempo de antecipação e atraso da tarefa i e α_i e β_i são as penalidades respectivas.

O procedimento PDDOIP (Gomes Jr. et al., 2007) é utilizado para determinar as datas ótimas para início do processamento das tarefas da sequência dada e, assim, permitir o cálculo dos valores e_i e t_i da função 5.29.

Os movimentos adotados não produzem soluções inviáveis; desta maneira, pode-se observar que a função de avaliação f é a própria função objetivo do PLIM (Vide 5.1).

5.2.5 Geração da Solução Inicial

A solução inicial é gerada pelo procedimento GRASP-M, descrito no Algoritmo 10. Esse método usa a heurística *Greedy Randomized Adaptive Search Procedures* – GRASP (FEO; RESENDE, 1995), tendo como método de busca local a Descida em Vizinhança variável (*Variable Neighborhood Search* – VND) (MLADENOVIĆ; HANSEN, 1997). Além disso, o método também aplica uma estratégia de distribuição do processamento, a qual é adotada para que o mesmo possa ser executado em modo paralelo, e assim tirar proveitos de uma arquitetura com vários núcleos de processamento.

Algoritmo 10: GRASP-M

```
   Entrada: N_processors, Γ, GRASPmax, MDRmax
   Saída: v*
1  início
2  |  NewGRASPmax ← GRASPmax/N_processors
3  |  f* ← ∞
4  |  para i=1 até N_processors faça
5  |  |  v_i ← NOVO-PROCESSO-GRASP(Γ, NewGRASPmax, MDRmax)
6  |  fim
7  |  Aguarde o término de todos os processos GRASP
8  |  para i=1 até N_processors faça
9  |  |  se (f(v_i) < f*) então
10 |  |  |  f* ← f(v_i)
11 |  |  |  v* ← v_i
12 |  |  fim
13 |  fim
14 |  Retorne v*
15 fim
```

No Algoritmo 10, $N_{processors}$ indica a quantidade de núcleos de processamento disponíveis, *GRASPmax* é o número máximo de iterações GRASP (Seção 2.1) e *MDRmax* o número máximo de iterações do método de busca local MDR (Seção 2.3). O procedimento retorna a melhor solução construída pelo método representada por v^*.

Como pode ser visto no Algoritmo 10, linha 2, um novo número máximo de iterações para o GRASP é calculado. Este parâmetro *NewGRASPmax* é dependente da quantidade processadores disponíveis ($N_{processors}$). O valor de *NewGRASPmax* é repassado para o procedimento NOVO-PROCESSO-GRASP, como pode ser visto na linha 5. Após a criação de $N_{processors}$ novos processos GRASP, estes são executados em paralelo e o valor da melhor solução v^* é atualizado com o menor valor encontrado por algum dos processos

disparados. Esta estratégia de distribuição de processamento visa melhorar o desempenho com relação à métrica tempo de resposta. O processamento antes executado *GRASPmax* vezes é dividido em $N_{processors}$ núcleos de processamento, o novo número de iterações é menor e os núcleos de processamento que poderiam estar ociosos também são utilizados para construir a solução inicial. A distribuição do processamento não implica em perda de qualidade da solução final, uma vez que esta divisão de processamento na verdade se trata de um balanceamento do número iterações executadas(*GRASPmax*) entre os núcleos de processamento(*NewGRASPmax*). De fato, todas as *GRASPmax* iterações GRASP serão executadas, mas elas serão distribuídas da forma mais equânime possível entre os $N_{processors}$ núcleos de processamento, reduzindo, assim, o tempo de processamento total. De acordo com Resende e Ribeiro (2005), a estratégia adotada é categorizada como *multiple-walk independent-thread*.

O procedimento executado no Algoritmo 10, linha 5, é o que efetivamente constrói uma solução. Seu funcionamento é descrito no Algoritmo 11. Neste algoritmo, Γ é um conjunto de parâmetros γ que definem o tamanho da Lista Restrita de Candidatos do GRASP (vide Seção 2.1), *NewGRASPmax* representa o número máximo de iterações GRASP executadas em um *thread*, *MDRmax* é o número máximo de iterações do método MDR (vide Seção 2.3). *EDD* e *TDD* são as estratégias guiadoras para a construção de uma solução inicial (vide Seção 2.2).

Algoritmo 11: NOVO-PROCESSO-GRASP

```
   Entrada: Γ, NewGRASPmax, MDRmax
   Saída: v_min
 1 início
 2     f_min ← ∞
 3     para i=1 até NewGRASPmax faça
 4         Escolha γ ∈ Γ
 5         Escolha E ∈ {EDD, TDD}
 6         v ← Constroi_Solucao(E, γ)
 7         v ← Refinamento_1(v, MDRmax)
 8         se (f(v) < f_min) então
 9             v_min ← v
10             f_min ← f(v)
11         fim
12     fim
13     v_min ← Refinamento_2(v_min, MDRmax)
14     Retorne v_min
15 fim
```

O procedimento GRASP, tal como foi proposto, constrói a solução iterativamente. Esta solução é obtida com o método situado na linha 6. A definição do procedimento é apresentada no Algoritmo 12. De acordo com ela, a estratégia adotada para guiar a

construção da solução é representada por E e o tamanho de LRC é dimensionado por γ. O método retorna v.

Algoritmo 12: CONSTROI-SOLUCAO

Entrada: E, γ
Saída: v

```
1  início
2      v ← ∅
3      Selecione as tarefas candidatas para o conjunto LC
4      enquanto (LC ≠ ∅) faça
5          se (E = EDD) então
6              E_min = min_{i ∈ LC} {E_i}
7              E_max = max_{i ∈ LC} {E_i}
8              LRC = {i ∈ LC | E_i ≤ E_min + γ × (E_max − E_min)}
9              Selecione aleatoriamente uma tarefa i ∈ LRC
10             v ← v ∪ {i}
11             Atualize o conjunto de tarefas candidatas LC
12         senão
13             T_min = min_{i ∈ LC} {T_i}
14             T_max = max_{i ∈ LC} {T_i}
15             LRC = {i ∈ LC | T_i ≤ T_min + γ × (T_max − T_min)}
16             Selecione aleatoriamente uma tarefa i ∈ LRC
17             v ← v ∪ {i}
18             Atualize o conjunto de tarefas candidatas LC
19         fim
20     fim
21     retorne v
22 fim
```

Passo a passo uma nova tarefa é adicionada a v; desta maneira, todas as tarefas ainda não sequenciadas formam a lista de possíveis candidatos (LC) a serem adicionados à solução. A cada tarefa de LC é atribuído um valor de classificação calculado segundo uma função gulosa. Em (BAKER; SCUDDER, 1990) são apresentadas, entre outras, duas estratégias para construir uma solução (linha 5 do Algoritmo 11): EDD (*Earliest Due Date*) e TDD (*Tardiest Due Date*). Na estratégia EDD as tarefas são ordenadas pela data de início da janela de entrega; sendo assim, a primeira tarefa da lista é aquela com a menor data de início ($E_{\min}$) e a última é aquela com a maior data ($E_{\max}$). Na estratégia TDD, as tarefas são ordenadas pela data de término da janela de entrega; a primeira tarefa da lista é aquela com a menor data de término ($T_{\min}$) e a última aquela com a maior data ($T_{\max}$). Com as tarefas devidamente classificadas, as mais bem classificadas em LC são selecionadas formando uma lista restrita de candidatos (LRC). A partir de

LRC, uma tarefa é selecionada aleatoriamente e adicionada a v. O tamanho de *LRC* é dimensionado pelo parâmetro $\gamma \in [0, 1]$ (Algoritmo 11, linha 4), assim como em (Gomes Jr. et al., 2007). Na estratégia EDD a lista *LRC* é formada por tarefas i cuja data de início E_i for menor ou igual à $E_{\min} + \gamma \times (E_{\max} - E_{\min})$, já para a estratégia TDD, a lista *LRC* é formada pelas tarefas i com data de término T_i menor ou igual a $T_{\min} + \gamma \times (T_{\max} - T_{\min})$. O procedimento encerra a fase de construção quando todas as tarefas forem sequenciadas e retorna v.

O processo de refinamento do método GRASP se divide em duas fases. A primeira delas é executada após a construção da solução (Algoritmo 11, linha 7), sendo feita por um procedimento de complexidade computacional mais barata, uma vez que são executadas somente descidas randômicas. A definição do procedimento REFINAMENTO$_1$ é mostrada no Algoritmo 13. Neste algoritmo, v representa a solução a ser refinada e *MDRmax* é o número máximo de iterações do método MDR (vide Seção 2.3).

A execução do método varia entre movimentos de realocação e troca. Desta forma, explora-se o espaço de soluções segundo as vizinhanças N^R e N^T, nesta ordem. Inicialmente a solução corrente passa por uma descida randômica com movimentos de realocação (linha 6). Para tanto, alguma tarefa J_i da sequência é sorteada aleatoriamente, assim como a posição j para onde ela será realocada. Se após a realocação, a solução gerada for melhor avaliada que a solução corrente segundo a função de avaliação, a solução é aceita e passa a ser a solução corrente. Mas decorridas *MDRmax* iterações sem melhora da solução corrente, o movimento de realocação é interrompido e sobre a solução corrente agora é aplicada uma descida randômica com movimentos de troca (linha 17). Assim, duas tarefas J_i e J_j são sorteadas aleatoriamente e trocadas de posição. Se a troca produzir uma solução melhor segundo a função de avaliação, a solução é aceita como solução corrente e o refinamento com movimentos de troca é interrompido, retornando-se ao refinamento usando realocação (linha 3 do Algoritmo 13). Mas, se novamente não houver uma melhora da solução em *MDRmax* iterações, o método encerra a execução.

Neste procedimento não há garantia que o método REFINAMENTO$_1$ retorne um ótimo local com relação às vizinhanças N^R e N^T, uma vez que o espaço de soluções é explorado de forma aleatória através descidas randômicas. Em consequência disto, ao final do método GRASP um novo refinamento é aplicado à melhor solução gerada (Algoritmo 11, linha 13), com vistas a se produzir ótimos locais com relação às vizinhanças adotadas. Esta segunda fase do refinamento é executada pelo método REFINAMENTO$_2$. Sua definição é mostrada no Algoritmo 14. Como pode ser visto no Algoritmo 14, a solução

Algoritmo 13: REFINAMENTO$_1$

```
   Entrada: v, MDRmax
   Saída: v
1  início
2      repita
3          solucaoSemMelhora ← verdadeiro
4          iteracao ← 0
5          enquanto (iteracao ≤ MDRmax) faça
6              Selecione um vizinho aleatório v' ∈ N^R(v)
7              se (f(v') < f(v)) então
8                  v ← v'
9                  solucaoSemMelhora ← falso
10                 iteracao ← 0
11             fim
12             iteracao ← iteracao + 1
13         fim
14         se (solucaoSemMelhora = verdadeiro) então
15             iteracao ← 0
16             enquanto (iteracao ≤ MDRmax) faça
17                 Selecione um vizinho aleatório v' ∈ N^T(v)
18                 se (f(v') < f(v)) então
19                     v ← v'
20                     Interrompa a troca e retorne para a realocação (linha 3)
21                 fim
22                 iteracao ← iteracao + 1
23             fim
24         fim
25     até (solucaoSemMelhora = verdadeiro)
26     retorne v
27 fim
```

que será refinada é representada por v e $MDRmax$ é o número máximo de iterações do método MDR (vide Seção 2.3).

O procedimento REFINAMENTO$_2$ explora o espaço de soluções através de movimentos de troca, realocação e movimentos OR (vide Seção 5.2.3). Esta busca é guiada por um conjunto de estratégias (linha 5). No início do procedimento a estratégia 1 é adotada; assim, a solução corrente é refinada pelo método de descida randômica com movimentos de realocação. A cada execução sem melhora da solução corrente, a estratégia utilizada é atualizada sequencialmente; desta forma, após a estratégia 1 a estratégia 2 é utilizada. Se a solução corrente for melhorada segundo a função de avaliação por alguma das estratégias, esta solução é aceita como a solução corrente e o método retorna à estratégia 1, repetindo este processo até que nenhuma das estratégias melhorem a solução corrente. Em

todas as estratégias que aplicam descidas randômicas a execução é interrompida quando não houver melhora em *MDRmax* iterações. Nas descidas com realocação de k tarefas contíguas, inicialmente k assume seu valor mínimo; não havendo melhora, o valor de k é incrementado progressivamente até atingir seu limite máximo. Ao final da execução do procedimento, um ótimo local com relação às vizinhanças N^R, N^T e N^{Or} é retornado.

Algoritmo 14: REFINAMENTO$_2$

```
   Entrada: v, MDRmax
   Saída: v
 1 início
 2     Estrategia ← 1
 3     f_min ← f(v)
 4     repita
 5         selecione Estrategia faça
 6             caso 1
 7                 v ← Descida Randômica em v com movimentos de Realocação
 8             caso 2
 9                 v ← Descida Randômica em v com movimentos de Troca
10             caso 3
11                 v ← Descida Randômica em v com movimentos OR de Realocação
                   de k tarefas onde {2 ≤ k ≤ 3}
12             caso 4
13                 v ← Descida Completa em v com movimentos de Realocação
14             caso 5
15                 v ← Descida Completa em v com movimentos de Troca
16             caso 6
17                 v ← Descida Completa em v com movimentos OR de Realocação de
                   k tarefas onde {2 ≤ k ≤ n − 2}
18
19         fim
20         se (f(v) < f_min) então
21             Estrategia ← 1
22             f_min ← f(v)
23         senão
24             Estrategia ← Estrategia + 1
25         fim
26     até (Estrategia > 6)
27     retorne v
28 fim
```

5.2.6 Busca Tabu

O procedimento Busca Tabu (GLOVER; LAGUNA, 1997), conhecido na literatura como *Tabu Search* – TS, inicia sua execução a partir da solução retornada pelo procedimento GRASP-M (vide Seção 5.2.5). O funcionamento deste método está de acordo com a Seção 2.5. A cada iteração o espaço de soluções é explorado alternando entre as vizinhanças N^T e N^R, nesta ordem. Assim, na primeira iteração são aplicados movimentos de troca, na segunda movimentos de realocação e assim por diante. Após todos os vizinhos serem analisados, é aceito o melhor vizinho que não seja tabu, ou se tabu, que atenda a condição de aspiração. Para atender a esta condição, o vizinho considerado tabu deve ser melhor avaliado que a melhor solução existente até então segundo a função de avaliação (Seção 5.2.4), isto é, adota-se a condição de aspiração por objetivo. O vizinho escolhido é, então, aceito como a solução corrente. Esta escolha não implica necessariamente em uma solução melhor que a solução corrente, uma vez que o melhor vizinho não tabu da solução corrente pode ser uma solução pior. Esta condição de piora permite que o método consiga escapar de um ótimo local e explorar soluções em áreas ainda não analisadas.

Um atributo do melhor vizinho da solução corrente segundo a vizinhança N^T (ou N^R) é armazenado em uma Matriz Tabu T, de dimensão $n \times n$, sendo n o número de tarefas. O método define de forma variável o tempo p em que uma solução permanecerá tabu, estando neste prazo proibido o retorno a esse vizinho. Esse prazo variável de proibição p é selecionado aleatoriamente no intervalo $[0,9 \times \mathit{PrazoTABU} \leq p \leq 1,1 \times \mathit{PrazoTABU}]$, sendo $\mathit{PrazoTABU}$ um parâmetro do método. Assim, escolhido um vizinho, o valor de p é escolhido aleatoriamente no intervalo, sendo armazenado na Matriz Tabu o valor p somado ao número da iteração corrente do método.

Para caracterizar os atributos das soluções selecionadas, considere a solução corrente v e o seu melhor vizinho v'. Se este vizinho for escolhido pela troca de duas tarefas J_i e J_j $(i < j)$, o atributo que caracteriza este vizinho é definido pelo par de tarefas (J_i, J_{i+1}) de v. Caso o vizinho seja escolhido pela realocação progressiva de uma tarefa J_i para a posição j com $i < j$, o atributo que caracteriza a solução também será representado pelo par de tarefas (J_i, J_{i+1}) de v. Entretanto, para a realocação regressiva em que $i > j$, dois casos são considerados. No primeiro, selecionada a posição j para onde a tarefa J_i será realocada e considerando $i < n$, então o atributo que representará a solução é dado pelo par de tarefas (J_i, J_{i+1}) de v. Para o segundo caso, a tarefa J_i é a última, isto é, $i = n$; desta maneira, o atributo será representado pelo par de tarefas (J_{i-1}, J_i) de v.

Assim, quando um vizinho escolhido é caracterizado pelo par de tarefas J_i e J_j, o

prazo que este elemento permanecerá tabu é armazenado na posição (J_i, J_j) da matriz T. Desta forma, um elemento é considerado tabu quando o tempo tabu armazenado na matriz é maior que o número da iteração corrente do procedimento Busca Tabu.

Para exemplificar o funcionamento da Matriz T, considere a sequência $v = \{2, 3, 4, 1, 5\}$ com $n = 5$. Logo, T é representada pela matriz 5×5 (Figura 15(a)). Considere também um vizinho $v' = \{2, 5, 4, 1, 3\}$, resultado da troca entre J_2 e J_5, ou seja, troca realizada entre as tarefas 3 e 5. Este vizinho é caracterizado pelo par de tarefas J_2 e J_3 de v; assim, na posição $(3, 4)$ da matriz é armazenado um valor P que representa o tempo tabu p da solução somado ao número da iteração corrente. A Figura 15(b) mostra a matriz após a troca.

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

(a) Matriz Tabu Inicial

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	P	0
4	0	0	0	0	0
5	0	0	0	0	0

(b) Matriz Tabu após a troca

Figura 15: Funcionamento da Matriz Tabu

Na realocação de tarefas, analisando primeiro a realocação progressiva, considere novamente $v = \{2, 3, 4, 1, 5\}$ e o vizinho $v'' = \{3, 4, 1, 2, 5\}$. Este vizinho é o resultado de uma realocação da tarefa J_1 (tarefa 2) para a posição $j = 4$. O atributo que caracteriza esta solução é representado pelo par J_1 e J_2 de v; desta maneira, a posição $(2, 3)$ será preenchida com P, tal como anteriormente. A matriz resultante é mostrada na Figura 16.

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	P	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Figura 16: Matriz Tabu - Realocação Progressiva

Analisando agora a realocação regressiva, considere o primeiro caso, no qual dada a solução corrente $v = \{2, 3, 4, 1, 5\}$ e o vizinho $v''' = \{4, 2, 3, 1, 5\}$ resultado da realocação da tarefa J_3 (tarefa 4) para a posição $j = 1$ com $J_i < J_n$. Desta maneira, a solução é caracterizada por J_3 e J_4 de v; assim, o valor P é armazenado na posição $(4, 1)$ (Figura

17(a)). Para o segundo caso, considere $v = \{2, 3, 4, 1, 5\}$ e o vizinho $v^{iv} = \{5, 2, 3, 4, 1\}$ resultado da realocação da tarefa J_5 (tarefa 5) para a posição $j = 1$ com $J_i = J_n$. O atributo que representará a solução será formado por J_4 e J_5 de v. Assim, será gravado na posição (1, 5) o valor de P (Figura 17(b)).

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	P	0	0	0	0
5	0	0	0	0	0

(a) Realocação Regressiva 1

	1	2	3	4	5
1	0	0	0	0	P
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

(b) Realocação Regressiva 2

Figura 17: Matriz Tabu - Realocação Regressiva

Tal como o procedimento de construção da solução inicial (Seção 5.2.5), este método também define uma estratégia de processamento que visa a execução em modo paralelo e, desta forma, aproveitar melhor uma arquitetura com vários núcleos de processamento. Durante a execução do procedimento e após a definição do melhor vizinho da solução corrente, se este vizinho for melhor avaliado que a melhor solução até então, um processo de refinamento é aplicado. Para o refinamento da solução é utilizado o procedimento REFINAMENTO$_1$ (Seção 5.2.5).

A estratégia para processamento paralelo deste procedimento define que um novo *thread* para processar o refinamento é disparado a cada melhora da melhor solução até então, e que a Busca Tabu continue sua execução. Assim, são executados em paralelo o procedimento Busca Tabu e um ou mais processos de refinamento. O número de núcleos de processamento disponíveis é definido por $N_{processors}$; sendo um núcleo reservado para a Busca Tabu e os demais para os procedimentos de refinamento. É adotada uma restrição de no máximo $N_{processors} - 1$ processos de refinamento estarem em execução simultânea com a Busca Tabu. O objetivo é evitar que um número excessivo de *threads* estejam concorrendo entre si, o que aumentaria o tempo gasto pelo sistema operacional com a comutação de contextos dos processos para agendamento do novo *thread* para execução (SILBERSCHATZ; GALVIN; GAGNE, 2004). De fato, não há garantia de que substituindo-se um processo de refinamento por outro haja melhora no procedimento.

Devido ao fato de vários *threads* estarem executando em paralelo, é definindo um mecanismo de sincronização que garante um estado consistente durante a execução do método. Para ilustrar este mecanismo, são descritos dois cenários a seguir.

5.2.6.1 Primeiro Cenário

Após a Busca Tabu começar a explorar um espaço de soluções de melhora, as soluções que melhorarem a melhor solução corrente passarão por um refinamento. Neste cenário não há *threads* em espera, sendo criados no máximo $N_{processors} - 1$ processos de refinamento executando em paralelo com a Busca Tabu. A sincronização entre os processos pode ser resumida da seguinte maneira:

1. A Busca Tabu consegue melhorar a melhor solução existente antes que todos os *threads* de refinamento em execução: Neste caso, não havendo melhora da solução pelos *threads*, não haverá nenhuma mudança no espaço de busca que deverá ser sincronizada com a Busca Tabu; portanto, os processos de refinamento terminarão o seu processamento e não influenciarão na execução da Busca Tabu.

2. Algum dos *threads* de refinamento consegue melhorar a melhor solução até então antes que a Busca Tabu: Neste caso, a Busca Tabu é informada sobre esta nova solução. Após esta sincronização, a Busca Tabu reinicializa sua Matriz Tabu, e recomeça sua execução a partir da nova solução retornada pelo procedimento de refinamento melhor sucedido. Caso um ótimo local tenha sido alcançado, os *threads* de refinamento restantes encerrarão normalmente sua execução, mas em nada afetarão o fluxo de execução da Busca Tabu. Isso decorre da premissa empírica que estes *threads* restantes poderão alcançar no máximo o mesmo ótimo, uma vez que a busca está sendo feita em uma mesma região do espaço de soluções. Entretanto, se a solução retornada não se tratar de um ótimo local e havendo alguma melhora por um procedimento de refinamento de algum outro *thread* em execução, a sincronização com a Busca Tabu se dará da mesma forma como descrito anteriormente.

5.2.6.2 Segundo Cenário

Este cenário se diferencia do primeiro, devido ao fato de haver *threads* de refinamento em espera. Assim, novamente após a Busca Tabu começar a explorar um espaço de soluções de melhora, as soluções que melhorarem a melhor solução até então passam por um refinamento. São criados mais do que $N_{processors} - 1$ *threads* de refinamento, e como já dito, vários *threads* de refinamento poderão ser criados, mas no máximo $N_{processors} - 1$ entrarão em execução simultânea com a Busca Tabu. Considera-se como *threads* em execução os que foram criados e disparados em algum momento, e *threads* em espera os que foram criados mas não disparados em função da restrição do número de *threads* em

execução. Os *threads* em espera aguardarão até que *threads* em execução finalizem para, então, iniciarem. Eles também poderão nem mesmo chegar a executar. A sincronização entre os processos pode ser resumida como segue:

1. A Busca Tabu consegue melhorar a melhor solução existente antes que todos os *threads* de refinamento em execução. Assim, não havendo melhora da solução pelos *threads*, não haverá nenhuma mudança no espaço de busca que necessite de uma sincronização com a Busca Tabu; portanto, o processo de refinamento não influenciará na execução da Busca Tabu. Enquanto a Busca Tabu não encontrar um ótimo local, os *threads* em espera poderão entrar em execução.

2. Algum dos *threads* de refinamento consegue melhorar a melhor solução corrente antes que a Busca Tabu. Neste caso, a Busca Tabu é informada sobre esta nova solução. Após esta sincronização, a Busca Tabu reinicializa sua Matriz Tabu, e recomeça sua execução a partir da nova solução retornada pelo método de refinamento. Caso um ótimo local tenha sido alcançado, os *threads* de refinamento restantes encerrarão normalmente sua execução, mas em nada afetarão o fluxo de execução da Busca Tabu, considerando a premissa estabelecida no primeiro cenário. Nesta situação, também os *threads* em espera são descartados. Entretanto, se a solução retornada não se tratar de um ótimo local e havendo alguma melhora por um procedimentos de refinamento de algum outro *thread*, a sincronização com a Busca Tabu se dará da mesma forma como descrito anteriormente. Enquanto a solução melhorada não for um ótimo local, os *threads* em espera poderão entrar em execução.

Os dois cenários resumem como a Busca Tabu e os métodos de refinamento manterão um estado consistente. A execução poderá alternar entre estes dois cenários. Desta maneira, quando a execução se enquadrar no primeiro cenário, e mais processos de refinamento forem criados, gerando assim, fila de *threads* em espera, o segundo cenário é aplicado. Por outro lado, quando a execução se enquadrar no segundo cenário, e os *threads* em execução terminarem, $N_{processors-1}$ *threads* em espera, no máximo, são acionados. Quando a fila de *threads* em espera terminar, novamente o primeiro cenário é aplicado.

O pseudocódigo do procedimento Busca Tabu é mostrado no Algoritmo 15. De acordo com o pseudocódigo, $N_{processors}$ indica a quantidade de núcleos de processamento disponíveis, $PrazoTABU$ o tempo tabu, T a matriz tabu, *difElite* e *CE* são parâmetros utilizados para a construção do Conjunto Elite que é utilizado no método Reconexão por Caminhos (Seção 5.2.7), $TABUmax$ é o número máximo de iterações tabu (Seção 2.5) sem me-

lhora e *MDRmax* o número máximo de iterações do método de busca local MDR (Seção 2.3). O procedimento retorna a melhor solução, representada por v^*. O pseudocódigo do procedimento NOVO-PROCESSO-REFINAMENTO$_1$ é exibido no Algoritmo 16.

Algoritmo 15: TABU-M

Entrada: $N_{processors}$, v^*, *PrazoTABU*, T, *difElite*, *CE*, *TABUmax*, *MDRmax*
Saída: v^*

1 **início**
2 $v_{\min} \leftarrow v^*$ `// Melhor solução corrente`
3 $v \leftarrow v^*$ `// Solução auxiliar`
4 *IterCorrente* $\leftarrow 0$ `// Iteração corrente`
5 *MelhorIter* $\leftarrow 0$ `// Iteração da melhor solução`
6 $T \leftarrow \emptyset$ `// Lista Tabu`
7 $CE \leftarrow \emptyset$ `// Conjunto Elite`
8 *NumProcsRefinamento* $\leftarrow 0$ `// Contador de processos de refinamento`
9 **enquanto** (*IterCorrente* $-$ *MelhorIter* $\leq$ *TABUmax*) **faça**
10 Seja $v' \leftarrow v \oplus m$ o melhor elemento de $V \subseteq N(v)$, sendo $N = N^T$ quando *IterCorrente* for par e $N = N^R$ se *IterCorrente* for ímpar, e tal que o movimento m não seja tabu, ou se tabu, $f(v') < f(v^*)$
11 Atualize a Matriz Tabu T com o atributo que representa a solução v'
12 $v \leftarrow v'$
13 **se** $f(v) < f(v^*)$ **então**
14 $v_{\min} \leftarrow v$
15 **se** *NumProcsRefinamento* $< N_{processors} - 1$ **então**
16 *NumProcsRefinamento* $\leftarrow$ *NumProcsRefinamento* $+ 1$
17 NOVO-PROCESSO-Refinamento$_1$(v^*, v, *MDRmax*)
18 **senão**
19 Entre com o processo de refinamento na fila de espera. Assim que outro processo em execução terminar, realize um sorteio entre os processos da fila, execute o processo sorteado e atualize o contador de processos.
20 **fim**
21 *MelhorIter* $\leftarrow$ *IterCorrente*
22 Atualize o Conjunto Elite *CE*
23 **fim**
`// Verifica se algum processo atualizou` v^*
24 **se** $(f(v^*) < f(v_{\min}))$ **então**
25 $v_{\min} \leftarrow v^*$
26 $T \leftarrow \emptyset$
27 Retorne para a linha 10
28 **fim**
29 *IterCorrente* $\leftarrow$ *IterCorrente* $+ 1$
30 **fim**
31 Aguarde o término de todos os processos de refinamento
32 **Retorne** v^*
33 **fim**

Algoritmo 16: NOVO-PROCESSO-REFINAMENTO$_1$

Entrada: v^*, v, $MDRmax$

início
  $v \leftarrow v^*$ `// Solução auxiliar`
  $v' \leftarrow \text{Refinamento}_1(v, MDRmax)$ `// Refinamento da solução`
  se $(f(v') < f(v^*))$ **então**
    $v^* \leftarrow v'$ `// Atualiza melhor solução da Busca Tabu`
  fim
fim

5.2.7 Reconexão por Caminhos

A Reconexão por Caminhos (*Path Relinking* – PR) proposta por (GLOVER, 1996), é uma estratégia de intensificação geralmente aplicada como pós-otimização ou refinamento de ótimos locais obtidos durante a busca. Dado um par de soluções, uma das soluções é definida como solução guia e a outra como solução base. O objetivo é partir da solução base e caminhar em direção à solução guia por meio da adição de atributos da solução guia na solução base (vide Seção 2.6).

Durante a exploração do espaço de busca por TS (Seção 5.2.6), um conjunto de soluções, denominado Conjunto Elite (CE), é construído. Esse conjunto é formado por soluções encontradas por TS durante a exploração do espaço de soluções que sejam de boa qualidade e heterogêneas. Para fazer parte de CE, cada solução candidata deve satisfazer a um dos dois seguintes critérios: 1) ser melhor que a melhor das |CE| soluções do conjunto elite; 2) ser melhor que a pior das |CE| soluções do conjunto elite e se diferenciar de todas elas em um determinado percentual dos atributos, definido por *difElite*. Um atributo é definido como alguma tarefa J_i da sequência. Para exemplificar, considere as sequências v_1 = {1, 2, 4, 5, 6, 3, 7, 9, 8, 10} e v_2 = {1, 2, 4, 5, 6, 9, 3, 10, 7, 8}. Elas têm 50% de atributos iguais, a saber, as tarefas $J_1 = 1$, $J_2 = 2$, $J_3 = 4$, $J_4 = 5$ e $J_5 = 6$. O objetivo desta estratégia é evitar a inclusão em CE de soluções muito parecidas. Estando o conjunto elite já formado e ordenado de forma crescente, isto é, a melhor solução ocupando a primeira posição do conjunto e a pior, a última, quando uma solução entra e o conjunto já está completo, a de pior avaliação sai.

No algoritmo GTSPR-M, a Reconexão por Caminhos é utilizada como estratégia de pós-otimização aplicada após a execução de TS. O procedimento PR é aplicado a todos os pares de soluções de CE, de forma bidirecional. Ele funciona como segue. Conhecidas as soluções guia e base, a cada iteração, uma tarefa J_i da solução guia é inserida na mesma posição i na solução base. Dessa maneira, a tarefa da solução base que foi substituída sai da posição que ocupava e assume a posição daquela que entrou. Ao final das iterações, ou seja, após a solução base receber todos os atributos da solução guia, o atributo inserido que produzir a melhor solução é fixado e sobre esta solução é aplicado um refinamento. Este refinamento utiliza descida completa com movimentos de realocação, retornando um ótimo local com relação à vizinhança N^R.

A Reconexão por Caminhos é feita pelo procedimento PR-M mostrado no Algoritmo 17. Neste algoritmo, $N_{processors}$ representa o número de núcleos de processamento e v^*, a melhor solução encontrada até então.

Algoritmo 17: PR-M

Entrada: $N_{processors}$, v^*, CE
Saída: v^*
1 **início**
2 $LPR \leftarrow \emptyset$
3 Atualize LPR com todos os pares de soluções de CE
4 Execute NOVO-PROCESSO-PR(v^*, v_i, v_j) para cada par de soluções v_i e v_j de LPR
5 Aguarde o término de todos os processos PR
6 **Retorne** v^*
7 **fim**

Para aplicar a Reconexão por Caminhos usando os diversos núcleos de processamento do computador, usa-se uma estratégia de distribuição de processamento como segue. Inicialmente, dado todos os pares de soluções de CE, uma lista LPR contendo esses pares é formada. A seguir, $N_{processors}$ elementos de LPR são aleatoriamente selecionados e cada qual avaliado pelo procedimento NOVO-PROCESSO-PR, descrito no Algoritmo 18, em um *thread*. A criação de vários *threads* permite que vários procedimentos NOVO-PROCESSO-PR sejam executados em modo paralelo, aproveitando, assim, todos os núcleos de processamento disponíveis. Podem estar em execução simultânea no máximo $N_{processors}$ *threads*, evitando assim, o tempo gasto com a comutação de contexto dos processos (vide (SILBERSCHATZ; GALVIN; GAGNE, 2004)). À medida que algum dos *threads* termine, outro elemento de LPR é selecionado e avaliado pelo procedimento NOVO-PROCESSO-PR. O procedimento PR-M encerra sua execução após a avaliação de todos os pares de soluções de CE e retorna a melhor solução encontrada.

O procedimento NOVO-PROCESSO-PR (Linha 4) é quem de fato aplica a Reconexão por Caminhos. Ele avalia as soluções de forma bidirecional, isto é, dadas duas soluções, caminha-se tanto da pior solução para a melhor (dita Reconexão por Caminhos Progressiva - *Forward Path Relinking*), quanto da melhor para a pior solução (dita Reconexão por Caminhos Regressiva - *Backward Path Relinking*). Determinadas as soluções guia e base, a cada iteração, uma tarefa J_i da solução guia é inserida na mesma posição i na solução base. Desta maneira, a tarefa da solução base que foi substituída sai da posição que ocupava e assume a posição daquela que entrou. Ao final das iterações, ou seja, após a solução base receber todos os atributos da solução guia, o atributo inserido que produzir a melhor solução é fixado e sobre esta solução é aplicado um refinamento. Este refinamento utiliza descida completa com movimentos de realocação, retornando um ótimo local com relação à vizinhança N^R.

Para exemplificar seu funcionamento, considere as soluções guia = $\{5, 3, 2, 1, 4, 6\}$ e base = $\{2, 6, 5, 1, 4, 3\}$. Os passos seguintes descrevem este funcionamento:

Passo 1: Tem como objetivo inserir, na solução base, cada uma das 6 tarefas da solução guia na ordem em que aparecem. Para tanto, procede-se como segue:

Passo 1.1) Inserir a tarefa 5 na primeira posição da solução base. Para isto, a tarefa 5 passa a ocupar a primeira posição da solução base, onde está a tarefa 2. Esta, por sua vez, sai da primeira posição e vai para a terceira posição, onde está a tarefa 5, resultando na sequência $\{5, 6, 2, 1, 4, 3\}$.

Passo 1.2) Inserir a tarefa 3 na segunda posição da solução base. Para tanto, a tarefa 3 passa a ocupar a segunda posição da solução base, onde está a tarefa 6. Esta, por sua vez, sai da segunda posição e vai para a última posição, onde está a tarefa 6, resultando na sequência $\{2, 3, 5, 1, 4, 6\}$.

Passo 1.3) Inserir a tarefa 2 na terceira posição da solução base e proceder como anteriormente.

Passo 1.4) Inserir a tarefa 1 na quarta posição da solução base e proceder como anteriormente.

Passo 1.5) Inserir a tarefa 4 na quinta posição da solução base e proceder como anteriormente.

Passo 1.6) Inserir a tarefa 6 na sexta posição da solução base e proceder como anteriormente.

A inserção compreendida entre os passos 1.1 a 1.6 que produzir a melhor solução, segundo a função de avaliação, é realizada. Assim, a tarefa inserida é então fixada e um refinamento é aplicado a solução. Este refinamento, como já dito, utiliza movimentos de realocação, mas as tarefas fixadas não são realocadas, assim como, nenhuma tarefa é realocada para a posição de uma tarefa fixada. Esta sequência de passos é executada até que a solução base chegue na solução guia, isto é, todos os atributos da solução guia sejam inseridos e fixados na solução base.

O procedimento NOVO-PROCESSO-PR está descrito no Algoritmo 18.

5.3 Considerações Finais

Neste capítulo foram apresentadas formulações de programação matemática encontradas na literatura, bem como apresentado um algoritmo paralelo híbrido de três fases. Na primeira fase, a solução inicial é construída por um procedimento baseado em GRASP.

Algoritmo 18: NOVO-PROCESSO-PR

Entrada: v^*, v_i, v_j

início
  $Corrente \leftarrow v_i$
  $Guia \leftarrow v_j$
  $PosicoesFixadas \leftarrow \emptyset$
  para $i = 1$ **até** *2* **faça**
    enquanto ($Corrente \neq Guia$) **faça**
      $melhorPosicao \leftarrow \emptyset$
      $melhorValor \leftarrow \infty$
      para $j = 1$ **até** *total de tarefas da solução guia* $\notin PosicoesFixadas$ **faça**
        $v \leftarrow Corrente \cup$ {tarefa da posição j da solução guia}
        se ($f(v) < melhorValor$) **então**
          $melhorPosicao \leftarrow j$
          $melhorValor \leftarrow f(v)$
        fim
      fim
      $v \leftarrow Corrente \cup$ {tarefa da posição $melhorPosicao$ da solução guia}
      $v' \leftarrow$ Descida Completa em v com movimentos de Realocação
      se ($f(v') < f(v^*)$) **então**
        $v^* \leftarrow v'$
      fim
      $Corrente \leftarrow Corrente \cup$ {tarefa da posição $melhorPosicao$ da solução guia}
      $PosicoesFixadas \leftarrow PosicoesFixadas \cup melhorPosicao$
    fim
    $PosicoesFixadas \leftarrow \emptyset$
    $Corrente \leftarrow s_j$
    $Guia \leftarrow v_i$
  fim
fim

Nesse procedimento, foi aplicada uma estratégia de paralelização visando distribuir o número total de iterações GRASP de forma o mais equânime possível entre os núcleos de processamento disponíveis. Após essa fase, a solução construída é refinada por um procedimento baseado em Busca Tabu. A estratégia de paralelização adotada nessa segunda fase, definiu que a Busca Tabu e o processo de refinamento disparado a cada melhora da solução global fossem executados de forma paralela. Nesse procedimento também foi implementado um mecanismo de sincronização entre os procedimentos Busca Tabu e refinamento. Destaca-se que a paralelização do procedimento Busca Tabu tem por objetivo acelerar a análise da vizinhança. No entanto, se as buscas locais realizadas não forem bem sucedidas, não há interferência na trajetória normal da Busca Tabu. Na terceira e última fase, um procedimento baseado em Reconexão por Caminhos foi proposto. Para a sua paralelização, novamente foi adotada uma estratégia visando balancear a execução entre os núcleos de processamento disponíveis. Nesse procedimento de paralelização foi definido que o processo de Reconexão por Caminhos deveria ser aplicado a diferentes pares de soluções de forma simultânea, com vistas à redução do tempo computacional.

6 RESULTADOS

A seguir, são apresentados na Seção 6.1 os problemas-teste utilizados para validar as formulações de programação matemática apresentadas, bem como o algoritmo heurístico proposto, todos descritos no Capítulo 5. Os parâmetros utilizados nesse algoritmo são mostrados na Seção 6.3. Os resultados computacionais dos modelos matemático e heurístico são apresentados e discutidos nas seções 6.2 e 6.4, respectivamente. Na Seção 6.4.4, são apresentados os resultados de um teste de probabilidade empírica.

6.1 Problemas-Teste

Para validar os modelos propostos, foram utilizados três conjuntos distintos de problemas-teste na execução dos experimentos computacionais. O primeiro conjunto foi criado por Gomes Jr. et al. (2007), tendo em vista a metodologia descrita em Wan e Yen (2002), Liaw (1999), Mazzini e Armentano (2001). O segundo e o terceiro conjuntos foram gerados por Rosa (2009), tendo por base a metodologia apresentada em Wan e Yen (2002), Rabadi, Mollaghasemi e Anagnostopoulos (2004). Esses conjuntos são descritos a seguir.

O primeiro conjunto considera problemas-teste com 8, 9, 10, 11, 12, 15, 20, 25, 30, 40, 50 e 75 tarefas. Neste conjunto, os tempos de processamento P_i das tarefas são números inteiros entre 1 e 100, enquanto os centros das janelas de entrega estão distribuídos no intervalo $[(1-TF-RDD/2)\times MS,\ (1-TF+RDD/2)\times MS]$, sendo MS o tempo total de processamento de todas as tarefas, TF o fator de atraso e RDD a variação relativa da janela de entrega. As janelas de entrega possuem tamanhos distribuídos no intervalo $[1, MS/n]$. Os valores de TF são 0,1; 0,2; 0,3 e 0,4, já os valores de RDD são 0,8; 1,0 e 1,2. Os custos por atraso da produção (β_i) são números inteiros no intervalo [20, 100]. Diante do fato que em situações reais o atraso é menos desejável que a antecipação, os custos por antecipação da produção (α_i) são k vezes o custo por atraso da mesma tarefa, sendo k um número real aleatório no intervalo [0, 1]. Os tempos de preparação da máquina (S_{ij}) são números inteiros no intervalo [0, 50], sendo os mesmos simétricos, ou seja, $S_{ij} = S_{ji}$.

Todas as distribuições são uniformes. Em vista dos diferentes valores para TF e RDD, foram gerados um total de 144 problemas-teste, sendo 12 para cada número de tarefas.

O segundo conjunto considera problemas-teste com 6, 7, 8, 9, 10, 11, 12, 13 , 14, 15, 16, 17, 18, 19 e 20 tarefas. Neste conjunto, os tempos de processamento (P_i) são números inteiros selecionados aleatoriamente dentro do intervalo $[1, 40]$. Os centros das janelas de entrega estão distribuídos no intervalo $[(1-FA-VRJ/2)\times TTP, (1-FA+VRJ/2)\times TTP]$, 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. O tamanho da janela de entrega é um valor inteiro dentro do intervalo $[0, TTP/n]$, onde n é o número de tarefas. Foram 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. Os custos por atraso (β_i) e antecipação (α_i) são números inteiros selecionados aleatoriamente dentro dos intervalos $[1, 10]$ e $[1, \beta_i]$, respectivamente. Os tempos de preparação da máquina (S_{ij}) são números inteiros pertencentes ao intervalo $[5, 15]$. Neste conjunto, os tempos de preparação satisfazem a desigualdade triangular, que é a condição para a aplicabilidade do modelo 5.1.3, descrito à página 55. Foram gerados um total de 240 problemas-teste, sendo 16 para cada número de tarefas.

No terceiro conjunto, os problemas-teste envolvem 6, 7, 8, 9, 10, 11, 12, 15, 20, 30, 40, 50, 75, 100 e 150 tarefas, sendo que para cada número de tarefas há 16 instâncias, resultando em um total de 240 instâncias. O tempo de processamento (P_i), custo por unidade de atraso (β_i) e custo por unidade de antecipação (α_i) são inteiros pertencentes aos intervalos $[1, 100]$, $[1, 10]$ e $[1, \beta_i]$, respectivamente. Os centros das janelas de entrega estão distribuídos no intervalo $[(1 - FA - VRJ/2) \times TTP, (1 - FA + VRJ/2) \times TTP]$, onde TTP é o tempo total de processamento de todas 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 dentro do intervalo $[0, TTP/n]$, onde n é o número de tarefas. Foram 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. Os tempos de preparação S_{ij} são números inteiros pertencentes ao intervalo $[5, 30]$. Este conjunto de problemas-teste é o mais genérico e, diferentemente do conjunto anterior, os tempos de preparação não satisfazem necessariamente a uma propriedade em particular.

6.2 Resultados dos Modelos Matemáticos

Na realização destes experimentos computacionais, os modelos matemáticos apresentados na Seção 5.1 foram implementados utilizando-se a ferramenta de modelagem AMPL

e resolvidos pelo software de otimização CPLEX, versão 9.1 da ILOG, com seus parâmetros padrão. Foi utilizado para os testes um computador *Pentium Core 2 Quad* (Q6600) 2,4 GHz (4 núcleos de processamento) com 4 GB de memória RAM e sistema operacional *Windows XP Service Pack 3*. A formulação de programação matemática MPLIM-IT foi aplicada somente na resolução do segundo conjunto de problemas-teste, pois o mesmo é o único que satisfaz a desigualdade triangular.

Para o primeiro conjunto de dados, foram resolvidos problemas-teste de 8 a 12 tarefas. Não foi possível a execução de testes com problemas acima de 12 tarefas, uma vez que o otimizador não foi capaz de resolver estes problemas, devido a estouro de memória. Os resultados encontrados são apresentados na Tabela 4. Nesta Tabela, a primeira coluna apresenta o número de tarefas do conjunto de problemas-teste, a segunda o tempo computacional (em segundos) gasto para se chegar à solução ótima, a terceira apresenta o percentual de soluções ótimas encontradas dentro do conjunto de problemas-teste com mesmo número de tarefas e a última apresenta o desvio (gap_i^{cplex}) das soluções em relação ao limite inferior calculado pelo otimizador. Considerou-se como critério de parada para obtenção de uma solução ótima, um tempo limite de 3600 segundos. Foi utilizada a equação (6.1) para calcular o desvio das soluções. De acordo com esta equação, f_i^{cplex} e f_i^{lim} representam a solução e o limite inferior calculados pelo o otimizador, respectivamente.

$$gap_i^{cplex} = \frac{f_i^{cplex} - f_i^{lim}}{f_i^{lim}} \tag{6.1}$$

Tabela 4: Resultados dos modelos MPLIM-G e MPLIM-BG no primeiro conjunto de problemas-teste

Tarefas	Tempo(s)[1]		Ótimos(%)		gap_i^{cplex}(%)	
	MPLIM-G	**MPLIM-BG**	**MPLIM-G**	**MPLIM-BG**	**MPLIM-G**	**MPLIM-BG**
8	1,5	1,76	100,00	100,00	0,00	0,00
9	30,35	34,07	100,00	100,00	0,00	0,00
10	67,2	73,4	100,00	100,00	0,00	0,00
11	1635,65	1678,57	75,00	83,00	11,6	9,67
12	1936,21	1985,75	58,00	50,00	26,22	29,77
Avg	**734,18**	**754,71**	**87,00**	**87,00**	**7,56**	**7,89**

[1] Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Pode se observar na Tabela 4, que a formulação MPLIM-G se mostrou ligeiramente superior que a formulação MPLIM-BG. Na média, o modelo MPLIM-G foi melhor tanto em tempo computacional quanto no desvio das soluções. Para problemas com até 10 tarefas, ambos os modelos foram capazes de encontrar todas as soluções ótimas, enquanto

que para problemas com 11 e 12 tarefas o MPLIM-G obteve respectivamente 75% e 58% dos ótimos e o MPLIM-BG, 83% e 50% dos ótimos. Além disso, o tempo computacional é bastante elevado, chegando à casa dos 30 minutos, em média, para problemas com 12 tarefas.

No segundo conjunto de problemas-teste, foram resolvidos problemas de 6 a 12 tarefas. Novamente foram considerados 3600 segundos como limite para obtenção da solução ótima, bem como a utilização da equação (6.1) como medida de desvio das soluções encontradas pelo otimizador. Para representar a solução encontrada pelo otimizador foi utilizada a variável f_i^{cplex} e para representar o limite inferior calculado, a variável f_i^{lim}. São exibidos na Tabela 5 os resultados destes experimentos.

Tabela 5: Resultados dos modelos MPLIM-G, MPLIM-BG e MPLIM-IT no segundo conjunto de problemas-teste

Tarefas	Tempo(s)[1]			Ótimos(%)			gap_i^{cplex}(%)		
	MPLIM-G	**MPLIM-BG**	**MPLIM-IT**	**MPLIM-G**	**MPLIM-BG**	**MPLIM-IT**	**MPLIM-G**	**MPLIM-BG**	**MPLIM-I**
6	0,07	0,08	4,05	100,00	100,00	100,00	0,00	0,00	0,00
7	0,78	0,86	9,33	100,00	100,00	100,00	0,00	0,00	0,00
8	8,17	8,1	14,88	100,00	100,00	100,00	0,00	0,00	0,00
9	80,29	84,12	31,86	100,00	100,00	100,00	0,00	0,00	0,00
10	745,97	783,45	77,67	100,00	100,00	100,00	0,00	0,00	0,00
11	2455,4	2557,49	174	44,00	44,00	100,00	30,93	33,31	0,00
12	3380,16	3378,9	447,54	6,00	6,00	94,00	60,86	61,18	5,14
Avg	**952,98**	**973,28**	**108,48**	**79,00**	**79,00**	**99,00**	**13,11**	**13,5**	**0,99**

[1] Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Pela Tabela 5, observa-se que a formulação MPLIM-IT foi a de melhor desempenho, sendo capaz de encontrar um maior número de soluções ótimas com um menor tempo computacional e um desvio inferior às outras formulações. Observa-se, também, que a formulação MPLIM-G foi melhor que a formulação MPLIM-BG com relação ao tempo computacional. Entretanto, a formulação MPLIM-BG obteve um desvio médio inferior. As formulações MPLIM-G e MPLIM-BG obtiveram a mesma porcentagem de soluções ótimas.

Para o último conjunto de problemas-teste, os experimentos foram aplicados somente a problemas com até 12 tarefas. Assim como os experimentos anteriores, foi adotado o tempo limite 3600 segundos para a resolução dos problemas. Para o cálculo do desvio das soluções retornadas pelo CPLEX, foi utilizada a equação (6.1) e as variáveis f_i^{cplex} e f_i^{lim}, para representarem a solução e o limite inferior definidos pelo otimizador. A Tabela 6 resume estes resultados.

Nos testes com o terceiro conjunto de dados, o modelo MPLIM-BG se mostrou melhor tanto em tempo computacional quanto no desvio das soluções finais. Para problemas com

Tabela 6: Resultados dos modelos MPLIM-G e MPLIM-BG no terceiro conjunto de problemas-teste

Tarefas	Tempo(s)[1]		Ótimos(%)		gap_i^{cplex}(%)	
	MPLIM-G	**MPLIM-BG**	**MPLIM-G**	**MPLIM-BG**	**MPLIM-G**	**MPLIM-BG**
6	0,09	0,09	100,00	100,00	0,00	0,00
7	0,67	0,68	100,00	100,00	0,00	0,00
8	5,96	5,9	100,00	100,00	0,00	0,00
9	69,26	70,06	100,00	100,00	0,00	0,00
10	617,52	642,75	100,00	100,00	0,00	0,00
11	2220,23	2401,72	44,00	38,00	28,60	27,84
12	2796,18	2791,74	25,00	25,00	53,58	52,58
Avg	**815,7**	**844,7**	**81,00**	**80,00**	**11,74**	**11,49**

[1] Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

até 10 tarefas, todas as soluções ótimas foram encontradas pelos modelos. Nos problemas com 11 tarefas, o MPLIM-G foi melhor, obtendo 44% das soluções ótimas. Já para problemas com 12 tarefas, ambos os modelos empataram em relação a este quesito. O tempo computacional novamente se mostrou elevado, sendo necessário em torno de 46 minutos, para os modelos MPLIM-G e MPLIM-BG resolverem problemas com 12 tarefas.

6.3 Parâmetros do Algoritmo Heurístico

O algoritmo heurístico proposto neste trabalho foi desenvolvido com a linguagem C++, utilizando-se o IDE *NetBeans* 6.7 e o compilador GCC 4.3.3. Todos os experimentos computacionais foram executados no mesmo computador em que foram realizados os experimentos com as formulações de programação matemática, isto é, um computador *Pentium Core 2 Quad* (Q6600) 2,4 GHz (4 núcleos de processamento) com 4 GB de memória RAM e sistema operacional *Ubuntu 9.04*.

Os parâmetros do algoritmo heurístico foram os mesmos adotados em Souza et al. (2008), os quais são, a seguir, discutidos. Para o processo de calibração, foi utilizado um problema-teste de 25 tarefas e um ciclo de 10 execuções.

Na primeira fase do algoritmo, o método GRASP (Subseção 5.2.5) utilizou para o conjunto Γ os mesmos valores γ adotados em Gomes Jr. et al. (2007). Com relação ao número máximo de iterações *GRASPmax*, foram realizados testes com valores entre 1 e 30, sendo adotado para o número máximo de iterações um valor igual a 20. O parâmetro *MDRmax* foi calculado em função do número de tarefas n, sendo utilizado um valor q entre 1 e 10, onde $MDRmax = q \times n$. Nesta calibração foi adotado $q = 7$, o que resultou

em $MDRmax = 7 \times n$.

Foram calibrados para o método Busca Tabu (Subseção 5.2.6), o número máximo de iterações $TABUmax$ sem melhora e o prazo tabu padrão para um elemento, $PrazoTABU$. Assim como no método GRASP, os parâmetros $TABUmax$ e $PrazoTABU$ foram calculados com base em n, sendo adotados $TABUmax = q \times n$ e $PrazoTABU = q \times n$, com $q \in [1, 10]$. Foram escolhidos $q = 4$ e $q = 2$ para os parâmetros $TABUmax$ e $PrazoTABU$, respectivamente.

Para a terceira e última fase, foram calibrados para o método Reconexão por Caminhos (Subseção 5.2.7), o tamanho máximo do Conjunto Elite e o percentual mínimo de diversificação entre as soluções do Conjunto Elite, ambos representados por $|CE|$ e *difElite*, respectivamente. Para o tamanho máximo do Conjunto Eliete foi adotado um valor igual a 5. Já para *difElite*, foi utilizado um percentual mínimo de diversificação igual a 40%.

Um estudo detalhado sobre o processo de calibração dos parâmetros adotados neste trabalho está descrito em Penna (2009). Como o presente trabalho explorou o recurso *multicore* da máquina de teste, foi utilizado o parâmetro k para representar o número de núcleos de processamento disponíveis, sendo adotado $k = 4$. A seguir, é resumido na Tabela 7 o conjunto de parâmetros utilizados no algoritmo desenvolvido.

Tabela 7: Parâmetros do Algoritmo GTSPR-M

Parâmetro	Valores
Γ	{0; 0,02; 0,04; 0,12; 0,14}
$GRASPmax$	20
$MRDmax$	$7 \times n$
$TABUmax$	$4 \times n$
$PrazoTABU$	$2 \times n$
$\|CE\|$	5
difElite	0,4
Núcleos de processamento	4

6.4 Resultados do Algoritmo Heurístico

Estes experimentos computacionais estão divididos em três baterias de testes. Na primeira, Subseção 6.4.1, é utilizado o primeiro conjunto de problemas-teste, enquanto na segunda bateria (Subseção 6.4.2), o segundo conjunto. Por fim, na terceira bateria

(Subseção 6.4.3) é utilizado o terceiro conjunto.

Com relação à comparação dos tempos computacionais, ressalta-se que somente nas comparações com a versão sequencial do algoritmo proposto por Souza et al. (2008) e com os modelos de formulação matemática da Seção 5.1, os experimentos computacionais foram executados em máquinas com o mesmo poder computacional. Assim, de forma a se fazer uma comparação mais justa, os tempos computacionais dos outros autores foram ajustados. Os ajustes foram realizados reduzindo-se uma certa porcentagem dos tempos computacionais dos algoritmos desses autores. Estes ajustes, são na verdade aproximações baseadas em informações de *benchmarks* disponíveis nos sítios www.spec.org e www.tomshardware.com. Ambos os sítios têm as informações dos tempos gastos pelos processadores em questão na execução de vários programas.

Na primeira bateria de testes, o algoritmo GILS-VND-RT desenvolvido por Gomes Jr. et al. (2007) foi ajustado com uma redução de 30% nos tempos computacionais. Este valor foi usado no trabalho de Penna (2009) e mantido no presente trabalho. Já para o algoritmo AGA1 (RIBEIRO, 2009), foi realizada uma redução de 10% do tempo computacional. O cálculo das reduções foi baseado em informações disponíveis nos seguintes endereços:

- http://www.tomshardware.com/charts/cpu-charts-2006/compare,38.html?prod[204]=on-&prod[153]=on
- http://www.spec.org/cpu2006/results/res2007q3/cpu2006-20070723-01550.html
- http://www.spec.org/osg/cpu2000/results/res2003q1/cpu2000-20030224-01964.html
- http://www.spec.org/cpu2006/results/res2007q1/cpu2006-20070205-00388.html

Para a segunda e terceira baterias de testes, o algoritmo AGA1 teve novamente o seu tempo reduzido em 10%. A aplicação desse percentual foi baseada em informações disponíveis em:

- http://www.spec.org/cpu2006/results/res2007q3/cpu2006-20070723-01550.html
- http://www.spec.org/cpu2006/results/res2007q1/cpu2006-20070205-00388.html

Nas comparações realizadas com o algoritmo GPV desenvolvido por Rosa (2009) não foi realizado nenhum ajuste do tempo computacional, mesmo sendo o processador utilizado para testar o GPV superior ao da máquina em que o GTSPR-M foi testado. De

fato, a máquina em que o GPV foi testado é cerca de 20% mais rápida. Os ajustes dos tempos computacionais dos algoritmos com os quais o algoritmo GTSPR-M foi comparado, foram realizados somente nos casos em que o processador utilizado nos experimentos deste trabalho se mostrou superior. As informações sobre o desempenho dos processadores estão disponíveis nos sítios a seguir, conforme experimentos realizados no endereço www.spec.org:

- http://www.spec.org/cpu2006/results/res2007q3/cpu2006-20070723-01550.html
- http://www.spec.org/cpu2006/results/res2008q2/cpu2006-20080317-03707.html

6.4.1 Primeira Bateria de Testes

Nesta bateria de testes é utilizado o primeiro conjunto de problemas-teste descrito na Seção 6.1. O algoritmo GTSPR-M, ou simplesmente GTSPR-M, descrito na Seção 5.2, foi aplicado a todas as instâncias de problemas com 8, 9, 10, 11, 12, 15, 20, 25, 30, 40, 50 e 75 tarefas.

Os resultados retornados estão sintetizados na Tabela 8. Nesta Tabela, é apresentada a contribuição de cada fase do algoritmo GTSPR-M no esforço de minimização da função de avaliação descrita na Subseção 5.2.4. A coluna "Fase GRASP-VND" diz respeito à fase de obtenção da solução inicial, a coluna "Fase TS" à fase de refinamento por Busca Tabu e a coluna "Fase PR", à fase de pós-otimização por Reconexão por Caminhos. Em cada uma das fases é exibido quanto o algoritmo proposto melhorou as melhores soluções conhecidas (Equação (6.2)). Nesta medida, quanto maior o valor, melhor. Também é exibido o desvio médio das soluções encontradas em relação às melhores soluções conhecidas (Equação (6.3)). Nesta medida, quanto menor o valor, mais robusto é o algoritmo. Para cada problema-teste i do grupo, $f_i^{GTSPR\text{-}M^*}$ e $\bar{f}_i^{GTSPR\text{-}M}$ representam o melhor valor e valor médio encontrados pelo algoritmo em cada fase, respectivamente. A representação da melhor solução conhecida é dada por f_i^*. Por fim, também é mostrado em cada fase, o tempo médio de processamento acumulado, em segundos.

$$imp_i^{best} = \frac{f_i^* - f_i^{GTSPR\text{-}M^*}}{f_i^*} \tag{6.2}$$

$$gap_i^{avg} = \frac{\bar{f}_i^{GTSPR\text{-}M} - f_i^*}{f_i^*} \tag{6.3}$$

Tabela 8: Resultados das fases do algoritmo GTSPR-M na primeira bateria de testes

	Fase GRASP-VND			Fase TS			Fase PR		
Tar.	imp^{best} %	$\overline{gap}^{avg}$ %	tempo[(1)] (s)	imp^{best} %	$\overline{gap}^{avg}$ %	tempo[(1)] (s)	imp^{best} %	$\overline{gap}^{avg}$ %	Tempo[(1)] (s)
8	0,00	0,00	0,03	0,00	0,00	0,04	0,00	0,00	0,05
9	0,00	0,00	0,04	0,00	0,00	0,05	0,00	0,00	0,07
10	0,00	0,00	0,05	0,00	0,00	0,07	0,00	0,00	0,11
11	0,00	0,10	0,07	0,00	0,01	0,10	0,00	0,00	0,17
12	0,00	0,08	0,09	0,00	0,00	0,14	0,00	0,00	0,24
15	0,00	1,31	0,15	0,00	0,43	0,30	0,00	0,32	0,49
20	0,00	1,41	0,38	0,00	0,37	1,09	0,00	0,25	1,71
25	-0,60	2,30	0,95	0,00	1,06	3,33	0,00	0,84	5,23
30	-0,64	3,77	2,26	-0,09	1,59	9,62	0,00	1,30	14,60
40	-0,60	5,66	8,00	0,02	2,98	37,80	0,04	2,43	56,36
50	-2,35	8,36	24,83	-0,47	4,26	145,42	-0,30	3,95	208,14
75	-5,04	11,58	212,75	-1,08	6,69	1378,09	0,10	6,16	1922,23
Avg	**-0,77**	**2,88**	**20,80**	**-0,14**	**1,45**	**131,34**	**-0,01**	**1,29**	**184,12**

(1) Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Pode-se observar na Tabela 8 que, para problemas com até 20 tarefas, apenas a fase GRASP-VND do algoritmo, foi suficiente para encontrar as melhores soluções com um desvio médio de no máximo 1,41%. Os desvios médios foram reduzidos gradativamente nas outras fases. Na fase TS, o desvio chega a no máximo 0,43%, enquanto que na fase PR a variabilidade das soluções finais foi reduzida para um máximo de 0,32%.

Para problemas de 25 a 50 tarefas, observa-se que a fase GRASP-VND não foi capaz de alcançar as melhores soluções, sendo que a variabilidade média das soluções chegou a 8,36%. Contudo, na fase TS houve uma melhoria das melhores soluções. O desvio médio das soluções também foi reduzido nesta fase, chegando na média em até 4,26%. Na última fase, a pós-otimização reduziu a variabilidade média das soluções para 3,95%.

Para os problemas com 75 tarefas, a fase PR melhora em 0,10% as melhores soluções conhecidas. A respeito da variabilidade média das soluções, nota-se que na fase GRASP-VND uma variabilidade média de 11,58%, passando a 6,69% na fase TS e na fase PR para 6,16%.

É exibida, na Tabela 9, a comparação dos resultados do algoritmo proposto com os dos modelos matemáticos MPLIM-G e MPLIM-BG. A seguir, nas tabelas 10, 11, 12 e 13, o algoritmo GTSPR-M é comparado com os algoritmos heurísticos GILS-VND-RT, GTSPR, AGA1 e GPV, propostos nos trabalhos de Gomes Jr. et al. (2007), Souza et al. (2008), Ribeiro (2009), Rosa (2009), respectivamente. Como os resultados do algoritmo GTSPR publicado em Penna (2009) são superiores àqueles publicados em Souza et al.

(2008), apenas os resultados do primeiro trabalho são apresentados.

Foram utilizados na comparação da Tabela 9, os resultados dos modelos matemáticos exibidos na Tabela 4 e os resultados do algoritmo heurístico apresentados na Tabela 8. A Equação 6.1 foi utilizada para a medida de desvio das melhores soluções conhecidas para os modelos matemáticos; já para o algoritmo heurístico, foi utilizada a Equação (6.4). Nesta equação, $f_i^{GTSPR\text{-}M^*}$ representa o melhor valor obtido por GTSPR-M e f_i^* a melhor solução conhecida até então.

$$gap_i^{best} = \frac{f_i^{GTSPR\text{-}M^*} - f_i^*}{f_i^*} \tag{6.4}$$

Tabela 9: Resultados GTSPR-M × MPLIM-G × MPLIM-BG

Tarefas	Tempo(s)[1]			Melhores Soluções(%)			gap_i^{best}(%)		
	GTSPR-M	**MPLIM-G**	**MPLIM-BG**	**GTSPR-M**	**MPLIM-G**	**MPLIM-BG**	**GTSPR-M**	**MPLIM-G**	**MPLIM-**
8	0,05	1,5	1,76	100,00	100,00	100,00	0,00	0,00	0,00
9	0,07	30,35	34,07	100,00	100,00	100,00	0,00	0,00	0,00
10	0,11	67,2	73,4	100,00	100,00	100,00	0,00	0,00	0,00
11	0,17	1635,65	1678,57	100,00	75,00	83,00	0,00	11,6	9,67
12	0,24	1936,21	1985,75	100,00	58,00	50,00	0,00	26,22	29,77
Avg	**0,13**	**734,18**	**754,71**	**100,00**	**87,00**	**87,00**	**0,00**	**7,56**	**7,89**

[1] Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Observa-se na Tabela 9, uma redução do tempo computacional pelo algoritmo GTSPR-M. Ressalta-se, ainda, que para o conjunto de problemas-teste de 8 a 12 tarefas, o algoritmo proposto foi capaz de alcançar as melhores soluções conhecidas em todos os grupos de tarefas, enquanto que os modelos matemáticos obtiveram, na média, um total de 87% das melhores soluções conhecidas. O desvio das melhores soluções conhecidas foi nulo para o algoritmo proposto, ao passo que utilizando-se os modelos MPLIM-G e MPLIM-BG chegou-se a um desvio médio de 7,56% e 7,89%, respectivamente.

Na Tabela 10, a seguir, são apresentados os resultados obtidos pelo algoritmo GILS-VND-RT, de Gomes Jr. et al. (2007), os quais são comparados com os do algoritmo proposto neste trabalho. A Equação 6.5 foi utilizada para calcular a porcentagem de melhora das melhores soluções obtidas pelo algoritmo GILS-VND-RT. Nesta equação, para o grupo de tarefas i, $f_i^{GJr.^*}$ representa a melhor solução do GILS-VND-RT e $f_i^{GTSPR\text{-}M^*}$ a melhor solução do GTSPR-M.

$$imp^{Gjr.} = \frac{f_i^{GJr.^*} - f_i^{GTSPR\text{-}M^*}}{f_i^{GJr.^*}} \tag{6.5}$$

Tabela 10: Resultados GTSPR-M × GILS-VND-RT

Tarefas	Tempo(s)		$imp^{Gjr.}$	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	GILS-VND-RT$^{(*)}$	GTSPR-M	GTSPR-M	GILS-VND-RT
8	0,05	0,03	0,00	0,00	0,03
9	0,07	0,06	0,00	0,00	0,06
10	0,11	0,1	0,00	0,00	0,02
11	0,17	0,18	0,00	0,00	0,12
12	0,24	0,26	0,00	0,00	0,21
15	0,49	0,84	0,00	0,32	1,47
20	1,71	3,91	0,00	0,25	1,65
25	5,23	11,96	0,00	0,84	2,32
30	14,6	36,06	0,19	1,3	3,34
40	56,36	140,21	0,47	2,43	4,7
50	208,14	443,05	1,03	3,95	7,29
75	1922,23	1231,27	6,94	6,16	19,38
Avg	**184,12**	**155,66**	**0,72**	**1,27**	**3,38**

$^{(*)}$ Tempo Computacional Corrigido em 30%

De acordo com a Tabela 10, observa-se que o método GILS-VND-RT obteve melhores resultados para problemas com 8 a 12 tarefas em relação ao tempo computacional. Entretanto, a variabilidade média das soluções do algoritmo GTSPR-M foi menor, superando o algoritmo GILS-VND-RT em até 100%. Para problemas de 15 a 50 tarefas, o algoritmo GTSPR-M foi melhor tanto no tempo computacional quanto na variabilidade média das soluções, sendo que a melhora no tempo ficou entre 41% e 60%, enquanto que a redução na variabilidade ficou entre 44% e 85%. Para problemas com 75 tarefas, o algoritmo GILS-VND-RT utilizou um conjunto de parâmetros de modo a se reduzir o esforço computacional e obteve um menor tempo computacional. Todavia, a variação média das soluções finais do método GILS-VND-RT para 75 tarefas chegou a 19,38%, enquanto que para o algoritmo GTSPR-M, esta variação chegou a no máximo 6,16%, implicando em um melhora de 68% pelo algoritmo proposto. O algoritmo GTSPR-M foi capaz, além disso, de superar as melhores soluções do GILS-VND-RT em até 6,94%. Comparando o algoritmo GILS-VND-RT somente com a "Fase GRASP-VND" do GTSPR-M, percebe-se que o algoritmo proposto obteve uma variabilidade média das soluções finais de 2,88%, enquanto que no algoritmo GILS-VND-RT, a variabilidade final foi de 3,38%. Além de uma menor variabilidade, o tempo computacional do GTSPR-M também foi menor, o que implicou em uma redução de 87% no tempo médio final.

A Tabela 11 exibe a comparação do algoritmo GTSPR-M com a sua versão sequencial, o algoritmo GTSPR. Os resultados do GTSPR foram obtidos de Penna (2009). Para avaliar a melhora das soluções retornadas pela versão sequencial do algoritmo proposto, foi utilizada a Equação 6.6. A variável $f_i^{Pc^*}$ representa, para cada conjunto de tarefas, a melhor solução do algoritmo GTSPR, e $f_i^{GTSPR\text{-}M^*}$ a melhor do algoritmo GTSPR-M.

$$imp^{Pc} = \frac{f_i^{Pc^*} - f_i^{GTSPR\text{-}M^*}}{f_i^{Pc^*}} \tag{6.6}$$

Tabela 11: Resultados GTSPR-M × GTSPR

Tarefas	Tempo(s)[1]		imp^{Pc}	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	GTSPR	GTSPR-M	GTSPR-M	GTSPR
8	0,05	0,06	0	0	0
9	0,07	0,09	0	0	0
10	0,11	0,15	0	0	0
11	0,17	0,25	0	0	0
12	0,24	0,37	0	0	0
15	0,49	1,13	0	0,32	0,47
20	1,71	4,93	0	0,25	0,64
25	5,23	14,9	0	0,84	1,09
30	14,6	39,93	0	1,3	1,68
40	56,36	190,61	0,12	2,43	3,37
50	208,14	630,77	0,02	3,95	4,95
75	1922,23	6308,74	0,72	6,16	7,6
Avg	**184,12**	**599,33**	**0,07**	**1,27**	**1,65**

[1] Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Na comparação apresentada na Tabela 11, nota-se a superioridade do algoritmo GTSPR-M na melhoria do tempo computacional em todas as instâncias. Nos problemas-teste com 40 e 75 tarefas, esta melhora chegou a 70%. Ainda com relação aos tempos computacionais, o algoritmo proposto foi capaz de reduzir em 69% o tempo médio computacional final. Com respeito às melhores soluções do algoritmo GTSPR, o GTSPR-M, melhorou em 0,07% as melhores soluções da sua versão sequencial. Verifica-se, também, que o algoritmo proposto teve uma variabilidade menor que a do algoritmo GTSPR. Para a instância com 20 tarefas, a redução na variabilidade chegou a 60%.

As comparações dos resultados do algoritmo GTSPR-M com os do algoritmo AGA1 (RIBEIRO, 2009) são apresentadas na Tabela 12. Nesta tabela, a coluna imp^{Fb} representa quanto o algoritmo GTSPR-M foi superior às melhores soluções do algoritmo AGA1. Para tal avaliação, foi utilizada a Equação 6.7, em que $f_i^{Fb^*}$ e $f_i^{GTSPR\text{-}M^*}$ representam as melhores soluções dos algoritmos AGA1 e GTSPR-M, respectivamente.

$$imp^{Fb} = \frac{f_i^{Fb^*} - f_i^{GTSPR\text{-}M^*}}{f_i^{Fb^*}} \tag{6.7}$$

Tabela 12: Resultados GTSPR-M × AGA1

Tarefas	Tempo(s)		imp^{Fb}	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	AGA1$^{(*)}$	GTSPR-M	GTSPR-M	AGA1
8	0,05	0,84	0	0	0
9	0,07	1,14	0	0	0,15
10	0,11	1,44	0	0	0,24
11	0,17	1,99	0	0	0,03
12	0,24	2,53	0	0	0,07
15	0,49	5,42	0	0,32	0,76
20	1,71	18,54	0	0,25	0,73
25	5,23	41,14	0	0,84	1,02
30	14,6	100,86	0	1,3	1,6
40	56,36	302,29	0,04	2,43	2,45
50	208,14	806,49	-0,3	3,95	4,08
75	1922,23	1804,54	0,88	6,16	7,76
Avg	**184,12**	**257,27**	**0,05**	**1,27**	**1,57**

$^{(*)}$ Tempo Computacional Corrigido em 10%

Com respeito ao tempo computacional, nota-se pela Tabela 12, que o GTSPR-M foi melhor nos problemas com 8 a 50 tarefas. Verificou-se melhorias de 90% a 95% nas instâncias de 8 e 20 tarefas. Esta melhoria permaneceu entre 74% e 87% nas instâncias com 25 a 50 tarefas. Para problemas com 75 tarefas, o AGA1 obteve um tempo menor. Contudo, ressalta-se que para problemas com 75 tarefas, o AGA1 utilizou um conjunto especial de parâmetros visando a redução do esforço computacional. Entretanto, a média final dos tempos computacionais do GTSPR-M foi 28% inferior a do AGA1. Na variabilidade das soluções finais, novamente, o GTSPR-M foi melhor nos problemas com 9 a 11 tarefas, chegando esta melhoria a 100%. Já para 75 tarefas, esta melhoria foi de 21%. Com relação à avaliação das melhores soluções do AGA1, verifica-se, na média, que o algoritmo proposto foi capaz de melhorar em 0,05% as melhores soluções.

Na última comparação desta bateria de testes, os resultados do algoritmo proposto são comparados aos do algoritmo GPV (ROSA, 2009). A Tabela 13 exibe estes resultados. A

Equação 6.8 foi utilizada para avaliar a melhoria proporcionada pelo algoritmo proposto tendo em vista as melhores soluções do GPV. Para representar as melhores soluções do GPV e GTSPR-M foram utilizadas as variáveis $f_i^{Br^*}$ e $f_i^{GTSPR\text{-}M^*}$, respectivamente.

$$imp^{Br} = \frac{f_i^{Br^*} - f_i^{GTSPR\text{-}M^*}}{f_i^{Br^*}} \tag{6.8}$$

Tabela 13: Resultados GTSPR-M × GPV

Tarefas	Tempo(s)		imp^{Br}	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	GPV	GTSPR-M	GTSPR-M	GPV
8	0,05	0,11	0	0	0
9	0,07	0,17	0	0	0
10	0,11	0,24	0	0	0
11	0,17	0,46	0	0	0,06
12	0,24	0,74	0	0	0,02
15	0,49	1,47	0	0,32	0,51
20	1,71	5,19	0	0,25	1,49
25	5,23	14,22	0	0,84	1,53
30	14,6	30,84	0,04	1,3	2,41
40	56,36	111,52	1,09	2,43	3,93
50	208,14	309,03	1,31	3,95	6,48
75	1922,23	787,9	5,5	6,16	13,64
Avg	**184,12**	**105,16**	**0,66**	**1,27**	**2,51**

Nas comparações da Tabela 13, verifica-se que para problemas com até 50 tarefas, o GTSPR-M obteve os melhores tempos, chegando esta melhoria a 67% nos problemas-teste envolvendo 12 e 20 tarefas. No entanto, para os problemas-teste com 75 tarefas, com a utilização de parâmetros diferenciados para reduzir o esforço computacional, o GPV obteve um tempo computacional menor. Com relação aos melhores resultados obtidos, observa-se que o algoritmo proposto foi capaz de superar o algoritmo GPV nos problemas-teste de 30 a 75 tarefas, sendo que neste último grupo, esta melhora chegou a 5,5%. Nos demais problemas-teste, não houve melhoria, possivelmente devido ao fato de que as melhores soluções encontradas por ambos os algoritmos eram ótimas. Na variabilidade média das soluções finais, ambos os algoritmos alcançaram um desvio nulo para os problemas com 8 a 10 tarefas. Para os demais problemas, o GTSPR-M obteve uma variabilidade menor, sendo que nos problemas com 75 tarefas, a redução da variabilidade foi de 55%. Utilizando somente a fase "Fase GRASP-VND" para comparar com o resultado final do GPV, observa-se uma redução substancial no tempo computacional, uma vez que esta fase corresponde a cerca de 20% do tempo demandado pelo GPV e a produção de uma solução

com valor médio apenas 15% maior que a do GPV.

A Figura 18 exibe a relação entre os piores desvios das soluções finais com os desvios médios dessas soluções no primeiro conjunto de problemas-teste. Para cada problema-teste com n tarefas, o pior *gap* é dado como o maior desvio dentre todas as instâncias e o *gap* médio como a média dos desvios.

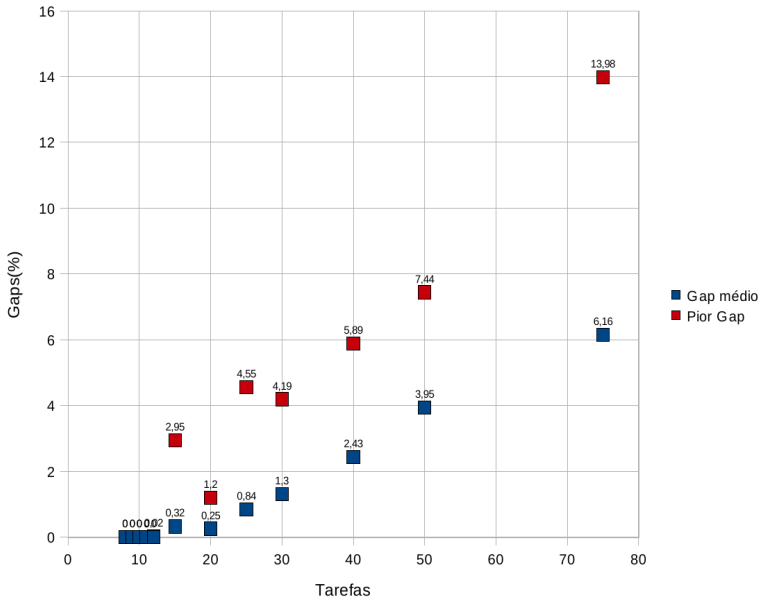


Figura 18: *Gap* Médio × Pior *Gap* - Primeiro Conjunto

Nota-se, pela Figura 18, que para problemas com até 12 tarefas, ambos os desvios foram nulos. Nos problemas com mais de 12 tarefas, a maior diferença entre o pior *gap* e o *gap* médio foi de 7% para problemas com 75 tarefas. Para os demais problemas, a diferença entre os desvios permaneceu abaixo de 3,5%.

6.4.2 Segunda Bateria de Testes

Nesta bateria de testes, foi utilizado o segundo conjunto de problemas-teste descrito na Seção 6.1. O algoritmo GTSPR-M, foi aplicado a todas as instâncias de problemas com 6, 7, 8, 9, 10, 11, 12, 13 , 14, 15, 16, 17, 18, 19 e 20 tarefas. Cada um desses problemas-teste, foi resolvido 20 vezes. Não houveram alterações dos parâmetros durante a execução dos experimentos computacionais. Foram novamente utilizadas as equações (6.2), (6.3) e (6.4), como medidas de desempenho para determinar a porcentagem de melhora da melhor solução conhecida, desvio médio das soluções e desvio da melhor solução conhecida, respectivamente. Nestas equações, $f_i^{GTSPR\text{-}M^*}$ e $f_i^{GTSPR\text{-}M}$ representam respectivamente o melhor valor e valor médio encontrados pelo algoritmo proposto. A

representação da melhor solução conhecida é dada por f_i^*.

Na Tabela 14, é exibida a contribuição de cada fase do algoritmo GTSPR-M na resolução do segundo conjunto de problemas-teste. Tal como na Subseção 6.4.1, a coluna "Fase GRASP-VND" diz respeito à fase de obtenção da solução inicial, a coluna "Fase TS", à fase de refinamento por Busca Tabu e a coluna "Fase PR", à fase de pós-otimização. Em cada uma dessas colunas, foi calculada a devida contribuição do método, o desvio médio das soluções finais e o tempo médio de processamento acumulado, em segundos.

Tabela 14: Resultados das fases do algoritmo GTSPR-M na segunda bateria de testes

	Fase GRASP-VND			Fase TS			Fase PR		
Tar.	imp^{best}	$\overline{gap}^{avg}$	tempo[(1)]	imp^{best}	$\overline{gap}^{avg}$	tempo[(1)]	imp^{best}	$\overline{gap}^{avg}$	Tempo[(1)]
	%	%	(s)	%	%	(s)	%	%	(s)
6	0,00	0,00	0,01	0,00	0,00	0,01	0,00	0,00	0,01
7	0,00	0,00	0,02	0,00	0,00	0,02	0,00	0,00	0,03
8	0,00	0,00	0,03	0,00	0,00	0,04	0,00	0,00	0,05
9	0,00	0,00	0,04	0,00	0,00	0,05	0,00	0,00	0,07
10	0,00	0,00	0,05	0,00	0,00	0,07	0,00	0,00	0,11
11	0,00	0,04	0,07	0,00	0,00	0,10	0,00	0,00	0,16
12	0,00	0,00	0,09	0,00	0,00	0,14	0,00	0,00	0,24
13	0,00	0,02	0,10	0,00	0,00	0,18	0,00	0,00	0,30
14	0,00	0,04	0,12	0,00	0,03	0,22	0,00	0,03	0,38
15	0,00	0,04	0,15	0,00	0,02	0,30	0,00	0,02	0,50
16	0,00	0,07	0,18	0,00	0,04	0,43	0,00	0,02	0,68
17	0,00	0,09	0,22	0,00	0,07	0,53	0,00	0,06	0,79
18	0,00	0,06	0,27	0,00	0,02	0,69	0,00	0,01	1,01
19	0,00	0,07	0,33	0,00	0,01	0,88	0,00	0,01	1,31
20	0,00	0,15	0,41	0,00	0,08	1,18	0,00	0,07	1,83
Avg	**0,00**	**0,04**	**0,14**	**0,00**	**0,02**	**0,32**	**0,00**	**0,01**	**0,50**

(1) Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Verifica-se na Tabela 14, que para problemas com até 10 tarefas, o desvio médio das soluções finais foi nulo. No entanto, este desvio aumenta para problemas com mais de 10 tarefas, chegando a 0,15% na primeira fase. Na fase TS, há uma redução deste valor, resultando em um desvio máximo de 0,08% e 0,02% na média. Na última fase, o algoritmo novamente reduz o desvio, sendo que este cai para um máximo de 0,07%. A média final do desvio das soluções finais foi de 0,01% na última fase.

Nesta bateria de testes, não houve melhora das melhores soluções, uma vez que na coluna imp^{best} todos os valores são nulos nas três fases. Isto se deve ao fato de que possivelmente todas as melhores soluções conhecidas são ótimas. O tempo computacional dos experimentos computacionais se mostrou baixo em todos os grupos i de tarefas, sendo que o maior tempo foi de apenas 1,83 segundos para a instância com 20 tarefas.

A seguir, na Tabela 15, os resultados dos modelos matemáticos MPLIM-G, MPLIM-BG e MPLIM-IT (Tabela 5), são comparados com os do algoritmo proposto (Tabela 14). Foram utilizadas as equações (6.1) e (6.4), para calcular os desvios das melhores soluções conhecidas para os modelos matemáticos e do algoritmo heurístico, respectivamente.

Tabela 15: Resultados GTSPR-M × MPLIM-G × MPLIM-BG × MPLIM-IT

Tarefas	Tempo(s)[1]				Melhores Soluções(%)				gap_i^{best}(%)		
	GTSPR-M	**PLIM-G**	**PLIM-BG**	**PLIM-IT**	**GTSPR-M**	**PLIM-G**	**PLIM-BG**	**PLIM-IT**	**GTSPR-M**	**PLIM-G**	**PLIM-BG** P
6	0,01	0,07	0,08	4,05	100,00	100,00	100,00	100,00	0,00	0,00	0,00
7	0,03	0,78	0,86	9,33	100,00	100,00	100,00	100,00	0,00	0,00	0,00
8	0,05	8,17	8,1	14,88	100,00	100,00	100,00	100,00	0,00	0,00	0,00
9	0,07	80,29	84,12	31,86	100,00	100,00	100,00	100,00	0,00	0,00	0,00
10	0,11	745,97	783,45	77,67	100,00	100,00	100,00	100,00	0,00	0,00	0,00
11	0,16	2455,4	2557,49	174	100,00	44,00	44,00	100,00	0,00	30,93	33,31
12	0,24	3380,16	3378,9	447,54	100,00	6,00	6,00	94,00	0,00	60,86	61,18
Avg	**0,1**	**952,98**	**973,28**	**108,48**	**100,00**	**79,00**	**79,00**	**99,00**	**0,00**	**13,11**	**13,5**

[1] Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Com relação à Tabela 15, verifica-se que o algoritmo GTSPR-M foi melhor que todos os modelos matemáticos em todas medidas de comparação. Observa-se, ainda, que o algoritmo heurístico proposto foi capaz de encontrar todas as melhores soluções conhecidas, sendo algumas delas com a otimalidade provada. A primeira fase do algoritmo foi capaz de encontrar todas as melhores soluções conhecidas com um desvio máximo de 0,04%, na média. Nas fases seguintes, este desvio foi reduzido a 0%. Com relação ao tempo computacional, o modelo MPLIM-IT gastou cerca de 7 minutos para resolver os problemas com 12 tarefas, e o algoritmo proposto menos de 1 segundo.

Na Tabela 16, são comparados os resultados do algoritmo AGA1 (RIBEIRO, 2009) com os resultados do algoritmo proposto (Tabela 14). Para avaliar a melhora das melhores soluções do AGA1, foi reutilizada a Equação 6.7.

Observa-se que o algoritmo GTSPR-M possui tempos computacionais menores que os do AGA 1 em todos os problemas-teste. Pela Tabela 16, nota-se que a redução no tempo chega a 98% para problemas com 6 tarefas e permanece entre 93% e 97% para os problemas com 7 a 16 tarefas e problemas com 18 a 20 tarefas. Não há melhoria das melhores soluções, uma vez que a coluna imp^{Fb} é nula para todos os problemas-teste. Com respeito à variabilidade média das soluções, novamente o GTSPR-M foi melhor, obtendo uma redução média aproximada de 64% na variabilidade média final.

A seguir, o algoritmo GPV é comparado com o GTSPR-M. O resultados desta compa-

Tabela 16: Resultados GTSPR-M × AGA1

Tarefas	Tempo(s)		imp^{Fb}	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	**AGA1**$^{(*)}$	**GTSPR-M**	**GTSPR-M**	**AGA1**
6	0,01	0,60	0,00	0,00	0,00
7	0,03	0,78	0,00	0,00	0,00
8	0,05	1,02	0,00	0,00	0,00
9	0,07	1,32	0,00	0,00	0,00
10	0,11	1,60	0,00	0,00	0,00
11	0,16	2,19	0,00	0,00	0,00
12	0,24	3,49	0,00	0,00	0,00
13	0,30	4,38	0,00	0,00	0,00
14	0,38	5,18	0,00	0,03	0,00
15	0,50	7,65	0,00	0,02	0,02
16	0,68	10,47	0,00	0,02	0,06
17	0,79	1,33	0,00	0,06	0,04
18	1,01	13,71	0,00	0,01	0,42
19	1,31	19,94	0,00	0,01	0,04
20	1,83	27,29	0,00	0,07	0,02
Avg	**0,50**	**6,73**	**0,00**	**0,01**	**0,04**

$^{(*)}$ Tempo Computacional Corrigido em 10%

ração são apresentados na Tabela 17. Na coluna imp^{Br} é apresentando quanto o algoritmo proposto foi superior às melhores soluções do GPV. Para tal avaliação, foi reutilizada a Equação 6.8.

De acordo com os resultados dos algoritmos GTSPR-M e GPV, conclui-se que o algoritmo proposto foi superior nas medidas de desempenho aplicadas nas comparações apresentadas na Tabela 17. Em relação aos tempos computacionais, o GTSPR-M obteve tempos menores em todos os problemas-teste, chegando esta redução a até 83% em problemas-teste com 8 tarefas. O algoritmo proposto alcançou todas as melhores soluções do GPV. Tal como na primeira bateria de testes, a variabilidade das soluções finais do algoritmo GTSPR-M novamente foi menor que a do GPV. Na média, o GTSPR-M chegou a uma variabilidade de 0,01% e o GPV a 0,03%.

A relação entre os piores desvios e desvios médios das soluções finais no segundo conjunto de problemas-teste é apresentada na Figura 19.

Para problemas com até 13 tarefas, observa-se pela Figura 19 que o pior *gap* e o *gap* médio foram nulos. Nos demais problemas-teste, a diferença entre os desvios ficou abaixo de 0,53%.

Tabela 17: Resultados GTSPR-M × GPV

Tarefas	Tempo(s)		imp^{Br}	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	GPV	GTSPR-M	GTSPR-M	GPV
6	0,01	0,04	0,00	0,00	0,00
7	0,03	0,17	0,00	0,00	0,00
8	0,05	0,29	0,00	0,00	0,00
9	0,07	0,34	0,00	0,00	0,00
10	0,11	0,41	0,00	0,00	0,00
11	0,16	0,32	0,00	0,00	0,04
12	0,24	0,49	0,00	0,00	0,00
13	0,30	0,69	0,00	0,00	0,01
14	0,38	0,91	0,00	0,03	0,04
15	0,50	1,25	0,00	0,02	0,04
16	0,68	2,36	0,00	0,02	0,14
17	0,79	2,34	0,00	0,06	0,04
18	1,01	3,03	0,00	0,01	0,05
19	1,31	3,80	0,00	0,01	0,01
20	1,83	5,26	0,00	0,07	0,14
Avg	**0,50**	**1,45**	**0,00**	**0,01**	**0,03**

6.4.3 Terceira Bateria de Testes

Nesta bateria foi utilizado o terceiro conjunto de problemas-teste descrito na Seção 6.1. O algoritmo GTSPR-M resolveu todas as instâncias de problemas com 6, 7, 8, 9, 10, 11, 12, 15, 20, 30, 40, 50, 75 e 100 tarefas. Para problemas-teste com até 75 tarefas, foram realizadas 20 repetições. Já para problemas com 100 tarefas, foram realizadas apenas 10 repetições, devido ao grande esforço computacional gasto nestes experimentos. Apesar deste conjunto envolver grupos de problemas-teste com 150 tarefas, não foram realizados experimentos com esse grupo. Assim como na Subseção 6.4.2, as equações (6.2), (6.3), (6.4) e (6.1), foram aplicadas para avaliar a porcentagem de melhora da melhor solução conhecida, desvio médio das soluções finais e desvio da melhor solução heurística e matemática, respectivamente.

A Tabela 18 resume o desempenho de cada uma das três fases do algoritmo GTSPR-M.

Com relação à Tabela 18, observa-se na primeira fase do algoritmo GTSPR-MULTICORE, que ele encontrou as melhores soluções conhecidas para problemas com até 30 tarefas. Com relação ao desvio médio das soluções, a fase GRASP-VND obteve uma média de 0,60%, sendo que o maior desvio das soluções finais foi de 2,59% em problemas-teste de 75 tarefas.

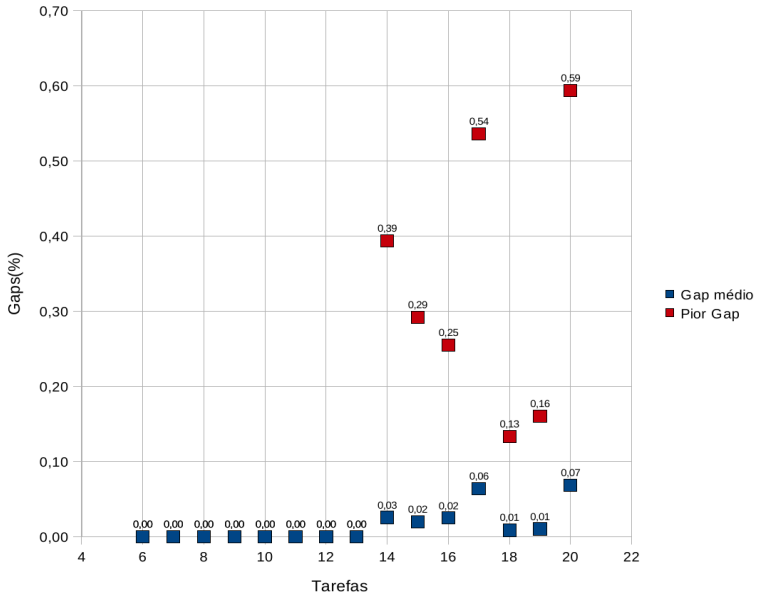


Figura 19: *Gap* Médio × Pior *Gap* - Segundo Conjunto

Na fase TS, várias das melhores soluções conhecidas foram encontradas e houve também uma melhora das melhores soluções para os problemas com 75 e 100 tarefas. Nos problemas com 75 tarefas, esta melhora chegou a 0,23%. O desvio médio das soluções finais, foi reduzido para 0,41%, na média. Ainda com relação a este desvio, nota-se que para problemas com até 15 tarefas, ele se tornou nulo. Já em problemas com mais de 15 tarefas, o desvio chegou ao percentual máximo de 1,87% em problemas com 75 tarefas.

A pós-otimização, realizada na terceira e última fase, foi novamente capaz de melhorar algumas das melhores soluções e reduzir o desvio médio das soluções finais. Na média, as melhores soluções foram melhoradas em 0,03% e o desvio reduzido a 0,39%.

Observa-se que a fase TS foi a que mais demandou tempo computacional, gastando na média 753,51 segundos. A fase PR, demorou em média 164,83 segundos para conclusão dos experimentos e a fase GRASP-VND, 129,28 segundos.

A seguir, os resultados do algoritmo proposto (Tabela 18), são comparados com os resultados dos modelos matemáticos MPLIM-G e MPLIM-BG (Tabela 6).

Conclui-se pela Tabela 19, que o algoritmo GTSPR-M foi capaz de encontrar todas as melhores soluções conhecidas com um desvio máximo das soluções finais de 0,07%, na média. Observa-se, ainda, que o tempo computacional do algoritmo proposto foi muito menor do que aqueles encontrados pelo otimizador usando-se os modelos matemáticos. Para problemas com 12 tarefas, foram necessários apenas 0,23 segundos para a sua conclusão, ao passo que nos modelos matemáticos, gastou-se cerca de 46 minutos para resolver

Tabela 18: Resultados das fases do algoritmo GTSPR-M na terceira bateria de testes

	Fase GRASP-VND			Fase TS			Fase PR		
Tar.	imp^{best} %	$\overline{gap}^{avg}$ %	tempo[(1)] (s)	imp^{best} %	$\overline{gap}^{avg}$ %	tempo[(1)] (s)	imp^{best} %	$\overline{gap}^{avg}$ %	Tempo[(1)] (s)
6	0,00	0,00	0,01	0,00	0,00	0,01	0,00	0,00	0,01
7	0,00	0,00	0,02	0,00	0,00	0,02	0,00	0,00	0,03
8	0,00	0,00	0,03	0,00	0,00	0,04	0,00	0,00	0,05
9	0,00	0,01	0,04	0,00	0,00	0,05	0,00	0,00	0,07
10	0,00	0,07	0,05	0,00	0,00	0,07	0,00	0,00	0,11
11	0,00	0,00	0,07	0,00	0,00	0,09	0,00	0,00	0,16
12	0,00	0,00	0,09	0,00	0,00	0,14	0,00	0,00	0,23
15	0,00	0,03	0,14	0,00	0,00	0,28	0,00	0,00	0,47
20	0,00	0,27	0,35	0,00	0,13	1,00	0,00	0,09	1,56
30	0,00	0,71	2,17	0,00	0,39	8,80	0,00	0,30	13,44
40	-0,14	1,13	9,11	-0,01	0,66	44,17	-0,01	0,60	64,58
50	-0,34	1,60	30,66	-0,04	1,04	165,59	-0,01	0,97	221,60
75	-0,17	2,59	296,69	0,23	1,87	1897,13	0,24	1,83	2295,03
100	-0,29	2,00	1470,53	0,17	1,65	10241,87	0,21	1,62	12069,13
Avg	**-0,07**	**0,60**	**129,28**	**0,03**	**0,41**	**882,80**	**0,03**	**0,39**	**1047,63**

(1) Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

o mesmo problema. Com a utilização dos modelos matemáticos não foi possível encontrar todas as melhores soluções. Além disso, houve um desvio médio de 11% nos valores das soluções finais.

A comparação dos resultados dos algoritmos GTSPR-M e AGA1 (RIBEIRO, 2009), é apresentada na Tabela 20. Tal como nas comparações anteriores com o algoritmo AGA1, a Equação 6.7 foi utilizada na avaliação proposta pela coluna imp^{Fb}.

Nota-se que o algoritmo GTSPR-M obteve tempos computacionais menores que os

Tabela 19: Resultados GTSPR-M × MPLIM-G × MPLIM-BG

Tarefas	Tempo(s)[(1)]			Melhores Soluções(%)			gap_i^{best}(%)		
	GTSPR-M	**MPLIM-G**	**MPLIM-BG**	**GTSPR-M**	**MPLIM-G**	**MPLIM-BG**	**GTSPR-M**	**MPLIM-G**	**MPLIM-**
6	0,01	0,09	0,09	100,00	100,00	100,00	0,00	0,00	0,00
7	0,03	0,67	0,68	100,00	100,00	100,00	0,00	0,00	0,00
8	0,05	5,96	5,90	100,00	100,00	100,00	0,00	0,00	0,00
9	0,07	69,26	70,06	100,00	100,00	100,00	0,00	0,00	0,00
10	0,11	617,52	642,75	100,00	100,00	100,00	0,00	0,00	0,00
11	0,16	2220,23	2401,72	100,00	44,00	38,00	0,00	28,60	27,84
12	0,23	2796,18	2791,74	100,00	25,00	25,00	0,00	53,58	52,58
Avg	**0,09**	**815,70**	**844,70**	**100,00**	**81,29**	**80,43**	**0,00**	**11,74**	**11,49**

(1) Tempo de CPU, em um PC Intel Core 2 Quad (Q6600) 2,40 GHz com 4 GB de RAM

Tabela 20: Comparação de resultados GTSPR-M × AGA1

Tarefas	Tempo(s)		imp^{Fb}	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	**AGA1**$^{(*)}$	**GTSPR-M**	**GTSPR-M**	**AGA1**
06	0,01	0,76	0,00	0,00	0,00
07	0,03	0,76	0,00	0,00	0,00
08	0,05	1,03	0,00	0,00	0,00
09	0,07	1,35	0,00	0,00	0,00
10	0,11	1,61	0,00	0,00	0,00
11	0,16	2,20	0,00	0,00	0,00
12	0,23	3,32	0,00	0,00	0,00
15	0,47	7,49	0,00	0,00	0,01
20	1,56	23,88	0,00	0,09	0,30
30	13,44	172,51	0,15	0,30	1,36
40	64,58	801,67	0,15	0,60	1,17
50	221,60	1575,11	0,29	0,97	1,50
75	2295,03	4978,02	4,49	1,83	6,33
100	12069,13	22299,79	2,44	1,62	3,07
Avg	**1047,61**	**2133,54**	**0,54**	**0,39**	**0,98**

$^{(*)}$ Tempo Computacional Corrigido em 10%

demandados pelo AGA1 em todas os problemas-teste. A redução desses tempos chegou a 99% em problemas-teste com 6 tarefas e permaneceu entre 92% e 97% para os problemas de 7 a 40 tarefas. Com relação à média final dos tempos, o algoritmo proposto foi cerca de 51% mais rápido que o AGA1.

Observa-se, na coluna imp^{Fb}, que o GTSPR-M foi capaz de melhorar as melhores soluções do AGA1 para problemas com 30 a 100 tarefas, superando este último em 0,54%, na média.

No desvio médio das soluções finais, percebe-se uma redução da variabilidade para problemas com 15 a 100 tarefas. Esta redução chega a 100% em problemas com 15 tarefas. Para problemas com 6 a 12 tarefas, ambos os métodos obtiveram um desvio nulo. O desvio médio final do algoritmo proposto foi cerca de 61% melhor que o do AGA1.

Na última comparação desta bateria, são apresentados na Tabela 21, os resultados dos algoritmos GTSPR-M e GPV. Para avaliar a superioridade do algoritmo proposto com relação às melhores soluções do algoritmo GPV, foi utilizada a Equação 6.8.

Pelos resultados apresentados na Tabela 21, o algoritmo proposto melhorou algumas das melhores soluções do GPV, bem como obteve uma redução na variabilidade das soluções finais. Nos problemas com 6 a 20 tarefas, os dois algoritmos encontraram todas as melhores soluções. Para os problemas com mais de 30 tarefas, o GTSPR-M foi capaz de

Tabela 21: Resultados GTSPR-M × GPV

Tarefas	Tempo(s)		imp^{Br}	$\overline{gap}^{avg}(\%)$	
	GTSPR-M	GPV	GTSPR-M	GTSPR-M	GPV
06	0,01	0,04	0,00	0,00	0,00
07	0,03	0,07	0,00	0,00	0,00
08	0,05	0,30	0,00	0,00	0,00
09	0,07	0,37	0,00	0,00	0,00
10	0,11	0,45	0,00	0,00	0,20
11	0,16	0,40	0,00	0,00	0,01
12	0,23	0,52	0,00	0,00	0,00
15	0,47	1,26	0,00	0,00	0,01
20	1,56	4,52	0,00	0,09	0,10
30	13,44	36,06	0,01	0,30	0,54
40	64,58	132,94	0,05	0,60	0,80
50	221,60	440,49	0,05	0,97	1,33
75	2295,03	1187,76	0,38	1,83	3,01
100	12069,13	4612,79	0,42	1,62	2,77
Avg	**1047,61**	**458,43**	**0,06**	**0,39**	**0,63**

melhorar várias das melhores soluções GPV.

Nas comparações da variabilidade das soluções finais, o GTSPR-M foi capaz de obter desvios inferiores na maior parte dos problemas-teste. Para problemas com 7, 9, 10, 11 e 15 tarefas, esta redução chegou a 100%.

Com relação aos tempos computacionais, o GTSPR-M obteve tempos menores para os problemas de 6 a 50 tarefas. Entretanto, para problemas com 75 e 100 tarefas, o GPV obteve tempos melhores, o que resultou em uma média final dos tempos computacionais menor. Em uma comparação com a primeira fase do algoritmo proposto ("Fase GRASP-VND"), percebe-se que o GTSPR-M obteve tempos menores em todos os grupos de tarefas, com uma redução no tempo médio final de 72%, e um desvio médio das soluções finais menor que a do GPV.

Os piores desvios e os desvios médios das soluções finais no terceiro conjunto de problemas-teste são relacionados na Figura 20.

Pela Figura 20 é possível observar que para problemas com até 15 tarefas, ambos os desvios foram 0%. Para o restante dos problemas-teste, a maior diferença entre os desvios foi de 3,29% para 40 tarefas e este desvio permaneceu abaixo de 3% para os demais.

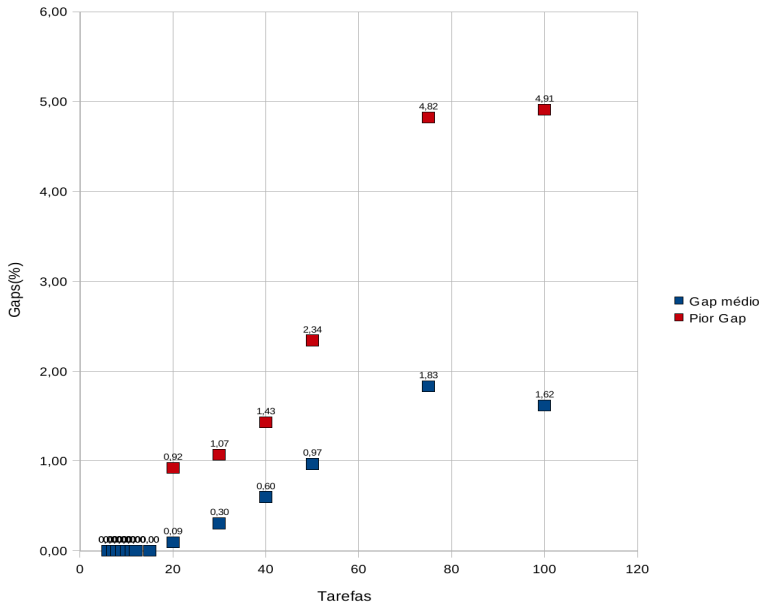


Figura 20: *Gap* Médio × Pior *Gap* - Terceiro Conjunto

6.4.4 Teste de Probabilidade Empírica

Devido ao fato de que nem todos os algoritmos puderam ser executados na mesma máquina, foi proposto também para a comparação dos resultados um teste de probabilidade empírica, baseado no trabalho desenvolvido em Aiex, Resende e Ribeiro (2002). Para execução dos experimentos, foi utilizado um problema-teste com 40 tarefas do primeiro conjunto de problemas descrito na Seção 6.1, tendo como alvo a melhor solução conhecida. O algoritmo foi executado 100 vezes, sendo que em cada rodada ele era interrompido após alcançar o alvo. Não foram permitidos tempos de execução repetidos; assim, os tempos repetidos foram descartados e uma nova execução foi agendada. Após determinados os tempos para as 100 execuções, estes foram ordenados de forma crescente e para cada tempo t_i, foi associada uma probabilidade $p_i = \frac{i-0,5}{100}$. Os resultados do gráfico $t_i \times p_i$ são apresentados na Figura 21. Neste gráfico, foram apresentados, também, os resultados do trabalho de Rosa (2009), relativos aos algoritmos GPV, AGA1, GILS-VND-RT e GTSPR.

Observa-se na Figura 21, que o algoritmo GTSPR-M, proposto no presente trabalho, demandou menos tempo que todos os outros algoritmos para garantir a melhor solução. De acordo com o gráfico, ele necessita de aproximadamente 95 segundos para retornar a melhor a solução, enquanto que o GPV necessita de 210 segundos, o GILS-VND-RT e o GTSPR, aproximadamente 310 segundos e o AGA1, 510 segundos. Verifica-se também, que os algoritmos GPV, GILS-VND-RT e GTSPR obtiveram um desempenho parecido nos instantes iniciais. O algoritmo AGA1 demandou mais tempo que todos os outros

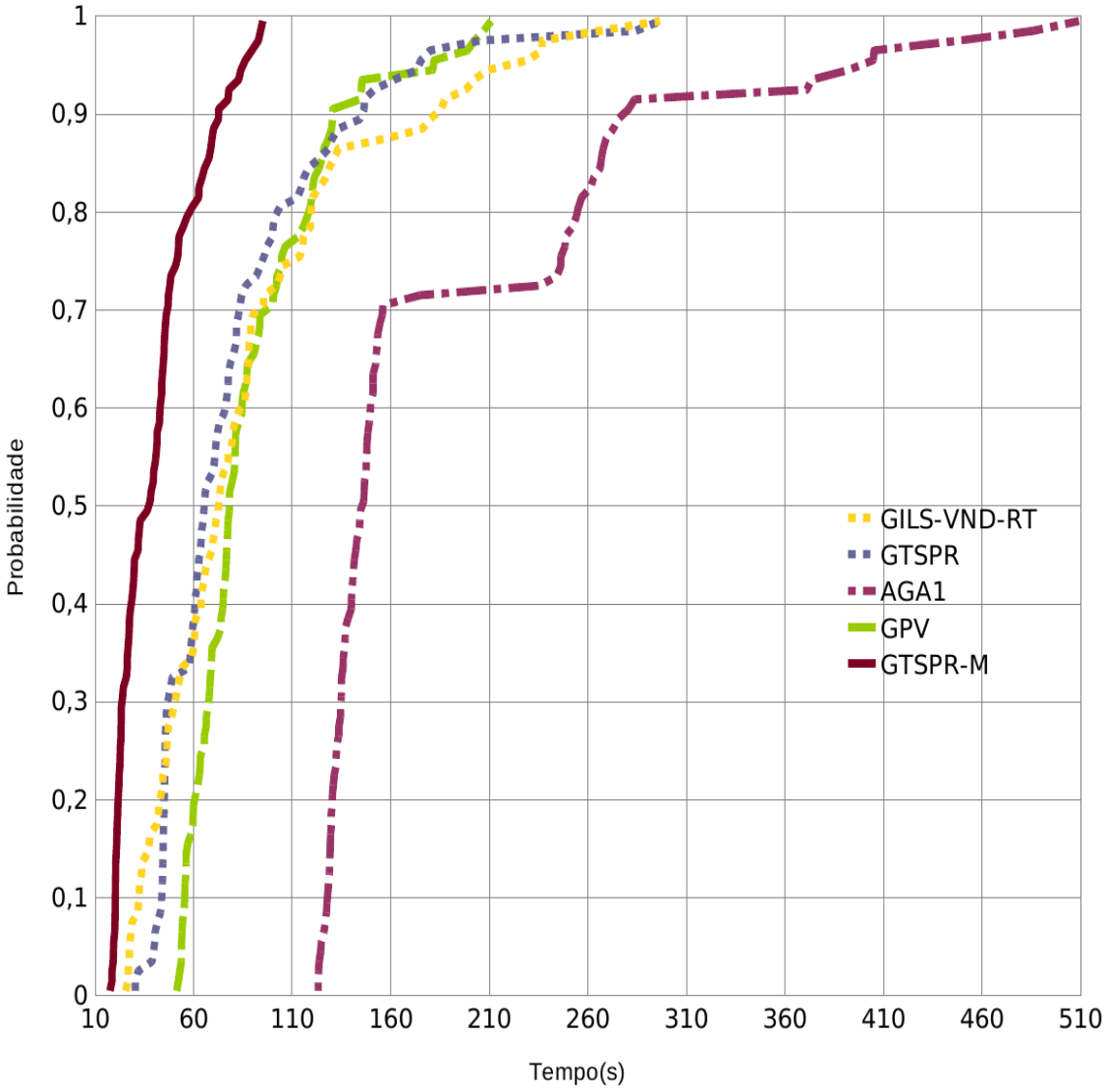


Figura 21: Teste de Probabilidade Empírica

algoritmos.

6.5 Considerações Finais

Neste capítulo, o algoritmo GTSPR-M foi testado e seus resultados comparados com quatro outros algoritmos da literatura, bem como com os resultados de três formulações de programação matemática existentes.

Os experimentos foram feitos considerando-se três conjuntos de problemas-teste da literatura e eles foram divididos em três baterias de testes.

Na primeira bateria, os resultados do algoritmo proposto foram comparados com

aqueles produzidos pela aplicação do otimizador CPLEX usando os modelos de formulação matemática MPLIM-G e MPLIM-BG. Verificou-se que para problemas-teste com até 12 tarefas, as melhores soluções conhecidas foram alcançadas em tempos computacionais muito menores e com um desvio médio das soluções nulo. Verificou-se, ainda, que para problemas com até 20 tarefas, somente a fase GRASP foi suficiente para encontrar as melhores soluções conhecidas. Na segunda comparação, foi utilizado o algoritmo GILS-VND-RT (Gomes Jr. et al., 2007). Observou-se que nesta comparação, o algoritmo GTSPR-M obteve tempos computacionais menores para instâncias de 15 a 50 tarefas. Na variabilidade média das soluções, o algoritmo proposto foi melhor em todas a instâncias, chegando a melhoria de até 100%. As melhores soluções do GILS-VND-RT foram superadas em 0,72% na média. Numa outra comparação com o algoritmo GILS-VND-RT, foi utilizada a primeira fase do algoritmo proposto, a qual foi capaz de superar o GILS-VND-RT em todas as avaliações aplicadas. Na comparação com a sua versão sequencial, apresentada em Souza et al. (2008), o algoritmo GTSPR-M se mostrou melhor, tanto em relação aos tempos computacionais, quanto na variabilidade média das soluções em todos os problemas-teste. Esta melhoria resultou em redução do tempo computacional de até 70%. O desvio das soluções finais foi reduzido em até 61% e os valores das melhores soluções do GTSPR foram superados em 0,07%, na média. Comparado com o algoritmo AGA1, proposto por Ribeiro (2009), o algoritmo GTSPR-M também se mostrou melhor no tempo computacional, variabilidade média das soluções finais e capacidade de encontrar melhores soluções finais. Esta superioridade na redução dos tempos computacionais chegou a 94%. Já na variabilidade média das soluções finais, a redução chegou a 90%. Na última comparação desta bateria de testes, foi utilizado o algoritmo GPV, desenvolvido em Rosa (2009). Os resultados demostraram que para problemas com até 50 tarefas, o tempo computacional do GTSPR-M foi menor, com redução no tempo de até 68%. Com respeito à variabilidade média das soluções finais e à capacidade de obter soluções finais melhores, o algoritmo proposto também foi melhor. Uma outra avaliação realizada foi com a "Fase GRASP-VND" do GTSPR-M. Nesta avaliação, a primeira fase do algoritmo proposto superou o GPV no tempo computacional e obteve uma variação das soluções finais menor que a obtida pelo GPV.

A seguir, o algoritmo GTSPR-M foi testado com o segundo conjunto de problemas-teste. Na comparação com os modelos matemáticos MPLIM-G, MPLIM-BG e MPLIM-IT para problemas de 6 a 12 tarefas, o algoritmo proposto superou todos os modelos, melhorando tanto o tempo computacional, quanto o desvio das melhores soluções. Com uma grande superioridade em relação à redução dos tempos computacionais, o GTSPR-

M foi capaz de encontrar as melhores soluções conhecidas apenas na fase GRASP-VND e com desvio médio das soluções finais de 0,03%. Com relação às comparações com o algoritmo AGA1, o algoritmo proposto também se mostrou melhor, obtendo melhoria de até 97% no tempo computacional. Na variabilidade média das soluções finais, novamente o GTSPR-M se mostrou melhor. Comparado ao algoritmo GPV, o algoritmo GTSPR-M, obteve melhores resultados nas medidas de desempenho aplicadas. A melhoria nos tempos computacionais chegou a cerca de 80%. No desvio médio das soluções finais, o algoritmo proposto teve, na média, um desvio de 0,01%, enquanto que no GPV este desvio foi mais alto, no caso, 0,03%.

Na última bateria de testes, inicialmente o GTSPR-M foi comparado com os modelos MPLIM-G e MPLIM-BG, e assim como nas comparações anteriores, ele superou todos os modelos matemáticos. O GTSPR-M encontrou todas as melhores soluções conhecidas para problemas com até 12 tarefas com um desvio nulo. O tempo computacional do algoritmo heurístico foi muito inferior ao do otimizador CPLEX usando os modelos matemáticos. Enquanto os resultados dos modelos matemáticos eram obtidos após cerca de 46 minutos em problemas com 12 tarefas, no GTSPR-M era necessário menos de um segundo para realizar a mesma tarefa. Na comparação seguinte, foi utilizado o algoritmo AGA1. Nesta comparação, o algoritmo GTSPR-M melhorou os tempos computacionais em até 96%. Em relação às melhores soluções do AGA1, o algoritmo proposto o superou em 0,54%. A variabilidade média das soluções finais foi menor no algoritmo GTSPR-M, que obteve 0,39%, contra 0,98% do algoritmo AGA1. Na última avaliação, o GTSPR-M foi comparado com o algoritmo GPV. O algoritmo GTSPR-M requereu tempos menores para problemas-teste com até 50 tarefas; entretanto, demandou um tempo computacional final médio maior do que o algoritmo GPV. Com relação à variabilidade das soluções finais e capacidade de gerar soluções finais melhores, o algoritmo proposto, tal como em todas as comparações anteriores, foi melhor. Na comparação com a primeira fase do algoritmo GTSPR-M, o algoritmo proposto foi capaz de reduzir o tempo computacional médio em 72%, na média, e se mostrou capaz de reduzir o desvio das soluções finais em relação ao GPV.

7 Conclusões e Trabalhos Futuros

Este trabalho teve seu foco na resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção, considerando janelas de entrega e tempo de preparação da máquina dependente da sequência de produção (PSUMAA-JP).

Foi proposto um algoritmo paralelo híbrido de três fases, denominado GTSPR-M. Esse algoritmo combina os procedimentos heurísticos GRASP (*Greedy Randomized Adaptive Search Procedures*) e Descida em Vizinhança Variável (*Variable Neighborhood Descent* – VND) para geração de solução inicial; Busca Tabu (*Tabu Search* – TS) para refinamento e Reconexão por Caminhos (*Path Relinking* – PR) como mecanismo de pós-otimização.

Além disso, o algoritmo GTSPR-M explora a capacidade de multiprocessamento paralelo da geração atual de computadores. Para tanto, a paralelização é feita por meios de *Threads*. O algoritmo foi executado em uma máquina dotada de um processador de 4 núcleos operando no Sistema SMP (*Symmetric Multiprocessing*).

A solução inicial do GTSPR-M foi gerada pelo procedimento GRASP. Nesse procedimento, foram utilizados os modelos de agendamento EDD (*Earliest Due Date*) e TDD (*Tardiest Due Date*) para guiarem a construção das soluções. Para o refinamento das soluções nessa fase, foi utilizado o procedimento VND, o qual explora o espaço de soluções por meio de movimentos de troca, realocação de uma tarefa e realocação de um bloco de tarefas. A seguir, a solução inicial construída foi refinada pelo procedimento TS, o qual utiliza movimentos de troca e realocação de uma tarefa para explorar o espaço de soluções. Durante a execução desse procedimento, foi construído um conjunto de boas soluções, denominado Conjunto Elite. O algoritmo GTSPR-M considerou, ainda, uma fase de pós-otimização. Essa terceira e última fase consistia na aplicação da Reconexão por Caminhos a todos os pares de soluções do Conjunto Elite.

Para cada solução gerada por GTSPR-M, foram determinadas as datas ótimas de conclusão das tarefas por meio de um procedimento de tempo polinomial da literatura.

Para testar o algoritmo proposto foram utilizados três conjuntos de problemas-teste encontrados na literatura. Os resultados obtidos pelo algoritmo GTSPR-M foram comparados com aqueles alcançados por quatro outros algoritmos da literatura, bem como com aqueles encontrados pela aplicação de um otimizador de mercado, o CPLEX, a três formulações de programação matemática para o problema.

As formulações de programação matemática foram aplicadas apenas a problemas-teste de até 12 tarefas, uma vez que acima disso o otimizador não era capaz de resolver os problemas por estouro de memória devido ao número elevado de variáveis.

Com base nos resultados encontrados, conclui-se que o algoritmo GTSPR-M foi capaz de alcançar todas as soluções ótimas conhecidas. Além disso, a variabilidade das soluções finais com otimalidade provada foi nula e o tempo computacional demandado foi inferior a um quarto de segundo, enquanto que o otimizador de mercado requereu até cerca de 56 minutos de processamento para resolver problemas de até 12 tarefas.

Adicionalmente, na comparação com os algoritmos heurísticos, o GTSPR-M também se mostrou mais robusto que os demais algoritmos dessa classe. Isso se deve ao fato de que ele foi capaz de alcançar as melhores soluções existentes na maioria dos casos, além de produzir soluções finais com a menor variabilidade. Além disso, o GTSPR-M foi o algoritmo que demandou o menor tempo de processamento na maioria das comparações.

Estes resultados validam, pois, a utilização do GTSPR-M na resolução do PSUMAA-JP, e evidenciam a contribuição do processamento paralelo na geração de soluções de melhor qualidade em menor tempo computacional.

Como contribuição principal do presente trabalho, destaca-se o desenvolvimento e implementação de uma estratégia de paralelização para o algoritmo GTSPR, proposto em Souza et al. (2008).

Finalmente, como trabalhos futuros, propõem-se os seguintes itens para estudo:

1. Aplicar periodicamente a estratégia de Reconexão por Caminhos durante a exploração da busca, e não somente como mecanismo de pós-otimização;

2. Estudar propriedades que possam ser aplicadas ao problema, de forma a reduzir o espaço de busca e, consequentemente, o tempo de processamento do algoritmo.

3. Estudar novas estratégias de paralelização dos métodos, de forma a se obter ganhos de desempenho tanto em tempo computacional quanto em qualidade das soluções.

Esta proposta decorreu da verificação de que a segunda fase do algoritmo GTSPR-M, a qual envolvia a Busca Tabu, foi a que demandou o maior tempo computacional. Para reduzir esse tempo propõe-se uma paralelização da análise da vizinhança, isto é, a cada iteração a vizinhança seria particionada entre os diversos núcleos de processamento disponíveis, ficando a cargo de cada núcleo a análise somente de sua parte. Com esse procedimento seria possível reduzir significativamente o tempo de processamento do procedimento de Busca Tabu.

4. Expandir o mecanismo de paralelização para várias máquinas conectadas em uma rede e não somente entre os núcleos de processamento de uma mesma máquina.

Referências

AIEX, R. M.; RESENDE, M. G. C.; RIBEIRO, C. C. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, v. 8, p. 343–373, 2002.

ALLAHVERDI, A.; GUPTA, J. N. D.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. *OMEGA*, v. 27, p. 219–239, 1999.

ALLAHVERDI, A. et al. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, v. 187, n. 3, p. 985–1032, 2008.

BAKER, K. R.; SCUDDER, G. D. Sequencing with earliness and tardiness penalties: A review. *Operations Research*, v. 38, p. 22–36, 1990.

BILGE, .; KURTULAN, M.; KIRAC, F. A tabu search algorithm for the single machine total weighted tardiness problem. *European Journal of Operational Research*, v. 176, p. 1423–1435, 2007.

BISKUP, D.; FELDMANN, M. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers and Operations Research*, v. 28, p. 787–801, 2001.

CAMPOS, G. D.; YOSHIZAKI, H. T. Y.; BELFIORE, P. Algoritmo genético e computação paralela para problemas de roteirização de veículos com janelas de tempo e entregas fracionadas. *Gestão e Produção*, v. 13, p. 271–281, 2006.

CHANG, P. C. A branch and bound approach for single machine scheduling with earliness and tardiness penalties. *Computers & Mathematics with Applications*, v. 37, n. 10, p. 133–144, 1999.

CLARKE, G.; WRIGHT, J. W. Scheduling of vehicles from a central depot to a number of delivery points. In: *Operations Research*. [S.l.: s.n.], 1964. v. 12, p. 568–581.

COELHO, I. M.; RIBAS, S.; SOUZA, M. J. F. Um algoritmo baseado em grasp, iterated local search para a otimização do planejamento operacional de lavra. In: *Anais do XI Encontro de Modelagem Computacional*. Volta Redonda: [s.n.], 2008.

de Paula, M. R. *Heurísticas para a minimização dos atrasos em seqüenciamento de máquinas paralelas com tempos de preparação dependentes da seqüência*. Dissertação (Dissertação de mestrado) — Departamento de Ciência da Computação, UFMG, Belo Horizonte, 2008.

DU, J.; LEUNG, J. Y. T. Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research*, v. 15, p. 483–495, 1990.

FELDMANN, M.; BISKUP, D. Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers and Industrial Engineering*, v. 44, p. 307–323, 2003.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133, 1995.

França Filho, M. F. *GRASP e Busca Tabu aplicados a problemas de programação de tarefas em máquinas paralelas*. Dissertação (Dissertação de mestrado) — Departamento de Engenharia de Sistemas, UNICAMP, Campinas, 2007.

GARCIA, V. J. et al. Algoritmo memético paralelo aplicado a problemas de sequenciamento em máquina simples. In: *Anais do XXXIII Simpósio Brasileiro de Pesquisa Operacional – XXXIII SBPO*. Campos do Jordão: [s.n.], 2001. p. 871–981.

GLOVER, F. Computing tools for modeling, optimization and simulation: Interfaces in computer science and operations research. In: _____. [S.l.]: Kluwer Academic Publishers, 1996. cap. Tabu search and adaptive memory programming: Advances, applications and challenges, p. 1–75.

GLOVER, F.; LAGUNA, M. *Tabu Search*. Boston: Kluwer Academic Publishers, 1997.

Gomes Jr., A. C. et al. Um método heurístico híbrido para a resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. In: *Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional – SBPO*. Fortaleza: [s.n.], 2007. p. 1649–1660.

GORDON, V.; PROTH, J.-M.; CHU, C. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, v. 139, p. 1–25, 2002.

GUPTA, S. R.; SMITH, J. S. Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, v. 175, p. 722–739, 2006.

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

HINO, C. M.; RONCONI, D. P.; MENDES, A. B. Minimizing earliness and tardiness penalties in a single-machine problem with a common due date. *European Journal of Operational Research*, v. 160, p. 190–201, 2005.

JACKSON, J. R. *Scheduling a production line to minimize maximum tardiness*. Management Science Research Project, 1955.

JAMES, R. J. W. Using tabu search to solve the common due date early/tardy machine scheduling problem. *Computers and Operations Research*, v. 24, n. 3, p. 199–208, 1997.

KIM, S. C.; BOBROWSKI, P. M. Impact os sequence dependent setup time on job shop scheduling performance. v. 32, n. 7, p. 1503–1520, 1994.

LEE, C. Y.; CHOI, J. Y. A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. *Computers and Operations Research*, v. 22, p. 857–869, 1995.

LI, F. Single machine earliness and tardiness scheduling. *European Journal of Operational Research*, v. 96, p. 546–558, 1997.

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

MAZZINI, R.; ARMENTANO, V. A. A heuristic for single machine scheduling with early and tardy costs. *European Journal of Operational Research*, v. 128, p. 129–146, 2001.

MLADENOVIĆ, N.; HANSEN, P. Variable Neighborhood Search. *Computers and Operations Research*, v. 24, p. 1097–1100, 1997.

MORAGA, R. J. et al. Solving the vehicle routing problem using the meta-raps approach. In: *International Conference on Computers and Industrial Engineering*. Montreal: [s.n.], 2001. p. 1–6.

MUHAMMAD, R. B. A parallel local search algorithm for euclidean steiner tree problem. In: *Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06)*. [S.l.: s.n.], 2006. p. 157–164.

PANWALKAR, S. S.; DUDEK, R. A.; SMITH, M. L. Sequencing research and the industrial scheduling problem. In: SE, E. (Ed.). *Symposium on the Theory of Scheduling and its Applications*. Springer: Berlin: [s.n.], 1973. p. 29–38.

PENNA, P. H. V. *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 (Dissertação de mestrado) — Programa de Pós-Graduação em Engenharia Mineral, Escola de Minas, UFOP, Ouro Preto, 2009.

PITANGA, M. *Computação em Cluster: O Estado da Arte da Computação*. Primeira edição. Rio de Janeiro: Brasport, 2003.

RABADI, G.; MOLLAGHASEMI, M.; ANAGNOSTOPOULOS, G. C. A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. *Computers and Operations Research*, v. 31, p. 1727–1751, 2004.

RESENDE, M. G. C.; RIBEIRO, C. C. Parallel metaheuristics: a new class of algorithms. In: _____. [S.l.]: John Wiley and Sons, 2005. cap. 14, *Parallel Greedy Randomized Adaptive Search Procedures*, p. 315–344.

RIBAS, S. et al. Parallel iterated local search aplicado ao planejamento operacional de lavra. In: *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional – XLI SBPO*. [S.l.: s.n.], 2009.

RIBAS, S. et al. Grasp, tabu search and path relinking for solving total earliness/tardiness single machine scheduling problem with distinct due windows and sequence-dependent setups. In: *Anais do IX Congresso Brasileiro de Redes Neurais / Inteligência Computacional – IX CBRN*. Ouro Preto: [s.n.], 2009.

RIBEIRO, C. C.; UCHOA, E.; WERNECK, R. F. A hybrid grasp with perturbations for the steiner problem in graphs. *INFORMSJC*, v. 14, p. 228–246, 2002.

RIBEIRO, F. F. *Um algoritmo genético adaptivo para a resolução do problema de sequenciamento em um máquina com penalização por antecipação e atraso da produção*. Dissertação (Dissertação de mestrado) — Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, Belo Horizonte, 2009.

ROSA, B. F. *Heurísticas para o problema de seqüenciamento em uma máquina com penalidades por antecipação e atraso da produção*. Dissertação (Dissertação de mestrado) — Programa de Pós-Graduação em Modelagem Matemática e Computacional, CEFET-MG, Belo Horizonte, 2009.

ROSSETI, I. C. M. *Estratégias sequenciais e paralelas de GRASP com reconexão por caminhos para o problema de síntese de redes a 2-caminhos*. Tese (Doutorado em Informática) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2003.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de Sistemas Operacionais*. 6. ed. Rio de Janeiro: LTC, 2004.

SOUZA, M. J. F. et al. GRASP, Tabu Search and Path Relinking for solving total earliness/tardiness single machine scheduling problem with distinct due windows and sequence-dependent setups. In: *Proceedings of the XXIX CILAMCE*. Maceió: [s.n.], 2008. v. 1, p. 1–15.

STANDERSKI, N. Aplicação de algoritmos genéticos paralelos a problemas de grande escala VMI - vendor managed inventory. In: *Anais do XXXV Simpósio Brasileiro de Pesquisa Operacional – XXXV SBPO*. Natal: [s.n.], 2003.

TANENBAUM, A. S.; WOODHULL, A. S. *Sistemas Operacionais, Projeto e Implementação*. Segunda edição. Porto Alegre: Bookman, 2002.

WAN, G.; YEN, B. P. C. Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *European Journal of Operational Research*, v. 142, p. 271–281, 2002.

YING, K.-C. Minimizing earliness-tardiness penalties for common due date single-machine scheduling problems by a recovering beam search algorithm. *Computers and Industrial Engineering*, v. 55, p. 494 – 502, 2008.