PAULO LÚCIO DE OLIVEIRA JÚNIOR

METAHEURÍSTICAS PARA A MINIMIZAÇÃO DO ATRASO TOTAL NO PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS COM DIVISÃO DE TAREFAS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de Magister Scientiae.

VIÇOSA

MINAS GERAIS - BRASIL

2013

Ficha catalográfica preparada pela Seção de Catalogação e Classificação da Biblioteca Central da UFV

T
O48m
2013

Oliveira Júnior, Paulo Lúcio de, 1983-
Metaheurísticas para a minimização do atraso total no problema de sequenciamento em máquinas paralelas com divisão de tarefas / Paulo Lúcio de Oliveira Júnior. – Viçosa, MG, 2013.
vi, 81 f. : il. (algumas color.) ; 29 cm.

Inclui apêndice.
Orientador: José Elias Cláudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 64-69.

1. Pesquisa operacional. 2. Otimização combinatória. 3. Solução de problemas. 4. Heurística. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22. ed. 001.424

PAULO LÚCIO DE OLIVEIRA JÚNIOR

METAHEURÍSTICAS PARA A MINIMIZAÇÃO DO ATRASO TOTAL NO PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS COM DIVISÃO DE TAREFAS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 28 de fevereiro de 2013

Gustavo Peixoto Silva

Luciana Brugiolo Gonçalves

José Elias Cláudio Arroyo
(Orientador)

Agradecimentos

À minha família pelo apoio incondicional e constante; aos meus amigos; ao meu orientador que me possibilitou essa oportunidade única para que eu possa fazer este trabalho; ao Departamento de Informática com seus professores por me darem uma base educacional universitária e me possibilitar a vencer as etapas e chegar onde eu estou hoje. Nomes, em especial: Mônica de Mello Antunes, Pedro Henrique Antunes Oliveira, José Lopes Barbosa, Oberlan Cristo Romão, Harlem Maurício e Vinícius Vilar Jacob.

Resumo

OLIVEIRA JÚNIOR, Paulo Lúcio de, M.Sc., Universidade Federal de Viçosa, fevereiro de 2013. **Metaheurísticas para a minimização do atraso total no problema de sequenciamento em máquinas paralelas com divisão de tarefas** Orientador: José Elias Cláudio Arroyo.

Este trabalho aborda o problema de escalonar n tarefas independentes em m máquinas paralelas idênticas com o objetivo de minimizar o atraso total das tarefas. Assume-se que uma tarefa possa ser dividida em sub-tarefas e estas possam ser processadas independentemente nas máquinas paralelas idênticas. Este problema é considerado NP-difícil, o que significa que, encontrar a solução ótima para este problema, levará um tempo computacional não aceitável. Por tal razão, métodos alternativos, como heurísticas, são utilizados para que boas soluções sejam obtidas em tempo razoável. Algumas heurísticas baseadas nas metaheurísticas GRASP (*Greed Randomized Adaptive Search procedure*) e no Algoritmo Genético são propostas. Além disso, três regras de dominâncias são utilizadas para melhorar as soluções. São comparados os resultados destes algoritmos com os resultados de outros dois algoritmos propostos na literatura.

Abstract

OLIVEIRA JÚNIOR, Paulo Lúcio de, M.Sc., Universidade Federal de Viçosa, February, 2013. **Metaheuristics to minimize total tardiness on scheduling of sub-jobs in parallel machines with splitting jobs.** Advisor: José Elias Cláudio Arroyo.

This work focuses on the problem of scheduling n independently jobs on m identical parallel machines with the objective of minimizing the total tardiness. It is assumed that a job can be split in sub-jobs and they can be processed independently in the identical machines. This problem is considered NP-Hard, what means that, finding an optimal solution will take an unacceptable computational time. For such reason, alternative methods, as heuristics, are used for good solutions to be gotten in reasonable time. Some heuristics based on metaheuristics GRASP (*Greed Randomized Adaptive Search Procedure*) and on Genetic Algorithm are proposed. Futhermore, three dominance rules are used in order to improve the solutions. The results of these algorithms are compared with the results of two others proposed in the literature.

Sumário

1 Introdução

Este trabalho apresenta propostas para a solução do Problema de Escalonamento em Máquinas Paralelas (*Parallel Machine Scheduling*, PMS) com a propriedade de divisão de tarefas. O problema consiste em achar uma sequência de sub-tarefas em cada máquina para que sejam processadas com o objetivo de minimizar o atraso total. Embora problemas com datas de entrega tenham sido estudados em muitos trabalhos, não existe muitos com o objetivo de minimizar o atraso total (Shim e Kim, 2008).

Pode-se considerar o escalonamento de máquinas paralelas como um processo de dois passos. No primeiro, tem-se que determinar quais tarefas serão alocadas em quais máquinas. No segundo, tem que determinar a sequência das tarefas em cada máquina (Pinedo et al, 2002).

Segundo Shim e Kim (2008), "nas indústrias que fazem uso de máquinas paralelas, cada tarefa pode ser considerada como uma ordem de produção para processar um tipo de produto em uma quantidade especificada, para uma determinada data de entrega. Em problemas de escalonamento para estas indústrias, tarefas podem ser divididas em um número de sub-tarefas que podem ser processadas independentemente em duas ou mais máquinas paralelas do mesmo tipo".

Este problema é aplicado, por exemplo, em computadores que utilizam vários processadores iguais, sendo que eles representariam as máquinas e os processos dos programas representariam as tarefas.

Este problema também tem aplicação prática em um sistema de manufatura de placa de circuito impresso (em inglês, PCB - *Printed Circuit Board*), formulado por Kim et al (2004).

O problema de Escalonamento em Máquinas Paralelas com a propriedade de divisão de tarefas é considerado NP-difícil (Xing e Zhang, 2000). Pelo fato de não ser fácil a obtenção soluções ótimas para problemas de atraso em máquinas paralelas de tamanho prático, pesquisadores tem focado no desenvolvimento de algoritmos heurísticos (Kim et al, 2004).

Embora a divisão de tarefas reduza o tempo de finalização de certas tarefas, ela aumenta a frequência dos tempos de preparação e atrasos em outras tarefas. Em outras palavras, quanto mais sub-tarefas nas quais uma tarefa é dividida, menor é o tempo da tarefa para ela ser processada, já que estas sub-tarefas podem ser processadas simultaneamente em diferentes máquinas. Entretanto, maior é o tempo de preparação requerido para essas sub-tarefas (Kim et al, 2004). Portanto, no problema de escalonamento considerado neste trabalho, é importante achar um número apropriado de sub-tarefas nas quais cada tarefa será dividida.

No capítulo 2, são dadas a descrição do problema e as definições gerais, é

explicado o modelo matemático, é definida a função objetivo, o modelo matemático e, por fim, as regras de dominância a que as soluções são aplicadas.

No capítulo 3, é feita a revisão bibliográfica, destacando as principais fontes para o desenvolvimento deste trabalho.

No capítulo 4, as heurísticas propostas, baseadas no GRASP e no Algoritmo Genético, são explicadas. Duas heurísticas propostas são baseadas no GRASP reativo, sendo que a diferença entre elas está na busca local, que será explicada com detalhes no capítulo 4. Outra heurística é uma versão melhorada do GRASP Reativo que foi denominada GPR, que utiliza *Path Relinking* para obter melhores soluções. A outra heurística proposta é a AGB que é a versão básica do Algoritmo Genético, com seus métodos de mutação e *cross-over*. Por fim, outras duas heurísticas, baseadas no Algoritmo Genético, foram propostas. A primeira versão utiliza o método de Perturbação, que é um modo de modificar levemente uma solução, seguido de Busca Local. A segunda versão utiliza o método de *Path-Relinking*.

No capítulo 5, é realizada a calibração dos algoritmos e o detalhamento dos parâmetros utilizados nas heurísticas.

No capítulo 6, seguem as comparações das heurísticas propostas com as da literatura.

Nos dois últimos capítulos, são mostradas as conclusões e os trabalhos futuros.

2 Descrição do problema

Neste capítulo é explicado, primeiramente, o problema de sequenciamento de tarefas em máquinas paralelas de uma forma geral. Depois, na seção 2.2, é abordada a definição do problema junto das notações utilizadas neste trabalho, a função objetivo, a representação da solução e, por fim, é apresentado um exemplo de solução com os tempos de preparação, processamento, datas de entrega. Na seção 2.3, é descrito o modelo matemático do problema. E, finalmente, na seção 2.4, são descritas as regras de dominância utilizadas neste trabalho.

2.1 Problema de sequenciamento de tarefas em máquinas paralelas

McNaughton (1959) introduziu o problema de sequenciamento de tarefas em máquinas paralelas. A partir de seu trabalho, vários pesquisadores começaram a realizar pesquisas sobre o tema.

O problema de sequenciamento de tarefas em máquinas paralelas consiste em saber quais tarefas são processadas em quais máquinas e, também, qual a ordem das tarefas nas máquinas, de forma a otimizar um ou mais critérios.

Cheng e Sin (1990) classificaram esse problema em três categorias, de acordo com as máquinas:
- Máquinas paralelas idênticas, em que os tempos de processamento de cada tarefa são iguais para todas as máquinas;
- Máquinas paralelas uniformes, em que os tempos de processamento das tarefas nas máquinas variam proporcionalmente;
- Máquinas paralelas não-relacionadas, em que os tempos de processamento de cada tarefa nas máquinas são diferentes, isto é, não havendo relação entre eles.

2.2 Definição do problema de sequenciamento em máquinas paralelas com divisão de tarefas

Neste trabalho, é abordado o problema de escalonar n tarefas independentes em m máquinas paralelas idênticas com o objetivo de minimizar o atraso total das tarefas, levando em consideração a propriedade de divisão de tarefas.

Alguns termos relacionados ao problema de escalonamento devem ser descritos para melhor esclarecimento do problema:

- Tarefa: Um lote de produção composto de um número de unidades-tarefa.
- Unidade-Tarefa: Menor unidade de processo para uma tarefa; unidades-tarefa são idênticas no sentido de que seus tempos de processamento e preparação e suas datas de entrega são idênticos.
- Sub-tarefa: Um conjunto de unidades-tarefa processadas em uma máquina.

Na propriedade de divisão de tarefas, cada tarefa pode ser dividida em partes e processadas independentemente em quaisquer máquinas ao mesmo tempo. Estas sub-tarefas podem ser processadas simultaneamente em máquinas diferentes. Assume-se que uma operação de preparação é requerida antes de uma sub-tarefa ser processada em uma máquina. Neste contexto, preparação significa que a tarefa precisa de alguma atividade prévia, começando a ser realmente processada só depois desta atividade. Uma atividade de preparação pode ser limpar a máquina para processar a tarefa, fazer alguns ajustes na máquina para suportar a tarefa em questão, entre outras. Os tempos de preparação são independentes da sequência das sub-tarefas.

É assumido que:
- Todas as tarefas estão disponíveis no tempo zero.
- Tempos de processamento de uma unidade-tarefa em todas as máquinas são iguais, desde que as máquinas sejam idênticas.
- Cada máquina pode processar somente uma sub-tarefa na hora.
- Cada sub-tarefa pode ser processada por somente uma máquina.

As notações usadas neste trabalho são dadas:

m = Número de máquinas paralelas.

n = Número de tarefas.

i = Índice para máquinas, $i = 1, 2, \ldots, m$.

j = Índice para tarefas, $j = 1, 2, \ldots, n$.

k = Índice para posições de sub-tarefas da tarefa j na máquina i, com $k = 1, 2, \ldots, u$.

u_j = Número de unidades-tarefa de cada sub-tarefa da tarefa j ($u_j = u$, $\forall\, j$), com $j = 1, 2, \ldots, n$.

p_j = Tempo de processamento de cada unidade-tarefa da tarefa j, com $j = 1, 2, \ldots, n$.

s_j = Tempo de preparação independente de sequência da tarefa j, com $j = 1, 2, \ldots, n$.

d_j = Data de entrega da tarefa j, com $j = 1, 2, \ldots, n$.

M = Uma constante com o valor da soma dos tempos de processamento e preparação de todas as tarefas.

C_j = tempo de finalização da última sub-tarefa da tarefa j.

$C_j(\delta)$ = tempo de finalização da última sub-tarefa da tarefa j na sequência (possivelmente parcial) δ.

2.2.1 Função objetivo

O atraso de uma tarefa é definido como $T_j = \max\{0, C_j - d_j\}$, em que C_j e d_j são o tempo de finalização e a data de entrega da tarefa j, respectivamente. O tempo de finalização de uma tarefa é o tempo quando todas as sub-tarefas desta tarefa são finalizadas. A função objetivo é o atraso total das tarefas. O atraso para uma tarefa será igual ao maior valor entre todos os valores de atraso das sub-tarefas de uma mesma tarefa. Em outras palavras, o atraso de uma tarefa é calculado baseado no tempo no qual todas as sub-tarefas são completadas (Sariçiçek e Çelik, 2011). A função objetivo é calculada pela expressão:

$$f = \sum_{j=1}^{n} T_j$$

2.2.2 A representação da solução

Neste trabalho, toda vez que for mostrado um exemplo de solução, com a finalidade de compreensão visual, ela será representada como uma lista de vetores, com cada vetor sendo uma sequência de tarefas em uma máquina. Essa lista está no formato $\{M_1, M_2, \ldots, M_m\}$. Um exemplo pode ser $M_1 = \{1, 1, 4, 3, 3\}$, $M_2 = \{2, 2, 1\}$ e $M_3 = \{4, 4, 4, 3\}$, com M_1, M_2 e M_3 sendo a sequência da primeira, segunda e terceira máquinas, respectivamente e $n = 4$, $m = 3$ e $k = 3$. Neste caso, a solução seria $s = \{\{1, 1, 4, 3, 3\}, \{2, 2, 1\}, \{4, 4, 4, 3\}\}$.

Um exemplo gráfico pode ser mostrado na Figura 2.1. Nela, há 3 máquinas paralelas e 4 tipos de tarefa. Antes de cada sub-tarefa, existe o tempo de preparação. Para cada grupo de sub-tarefa, só é necessário um tempo de preparação. Por este motivo, é uma vantagem agrupar unidades-tarefa para precisar de somente um tempo de preparação. Entretanto, também é uma vantagem separar as unidades-tarefa e permitir que cada máquina processe uma unidade-tarefa. Neste caso, cada unidade-tarefa precisa de um tempo de preparação, porém, elas são processadas ao mesmo tempo. Mais detalhes sobre isso são dados na seção sobre regras de dominância. Além disso, são mostrados, também, os tempos de finalização de cada tarefa. Vale notar que, como foi dito na seção 2.2.1, o tempo de finalização de uma tarefa é o tempo em que a última sub-tarefa desta tarefa termina. Outro ponto a ser notado é que, em cada máquina, todas as unidades-tarefa de uma determinada tarefa estão sempre agrupadas.

Para deixar este exemplo mais inteligível, valores são colocados para os tempos de preparação, processamento e data de entrega de cada tarefa. A Tabela 2.1 contém, na primeira coluna, as tarefas numeradas de 1 até 4.

Tarefa	s_j	p_j	d_j	u_j
1	5	4	10	3
2	2	6	14	2
3	6	3	28	3
4	10	11	12	4

Tabela 2.1: Dados de entrada correspondentes às tarefas.

Na segunda coluna desta mesma Tabela, estão os tempos de preparação, mostrados como "s_j", enquanto na terceira coluna estão os tempos de processamento, apresentados como "p_j" e, por fim, na quarta e quinta colunas, respectivamente, as datas de entrega das tarefas, mostradas como "d_j" e o número de unidades-tarefa.

Na Figura 2.1 nota-se a sequência das tarefas 1, 4 e 3 na máquina M_1. Na máquina M_2 tem-se a sequência das tarefas 2 e 1 e, por fim, a sequência das tarefas 4 e 3 na máquina M_3.

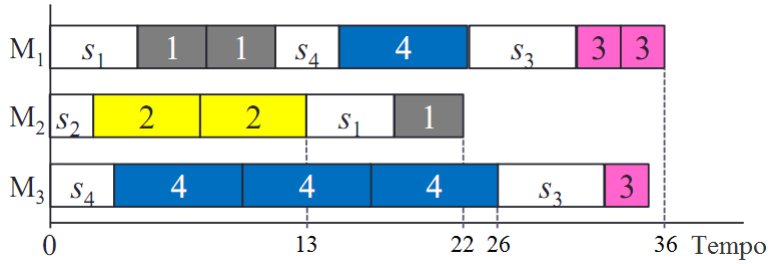


Figura 2.1: Exemplo gráfico de uma solução com 3 máquinas e 4 tarefas.

Pela Figura 2.1, nota-se que $C_1 = 22$, $C_2 = 13$, $C_3 = 36$ e $C_4 = 26$, que são os tempos de conclusão das tarefas 1, 2, 3 e 4, respectivamente. Considerando as datas de entrega "d" da Tabela 2.1, tem-se que $T_1 = max\{0, 22-10\} = 12$, $T_2 = max\{0, 13-14\} = 0$, $T_3 = max\{0, 36-28\} = 8$ e $T_4 = max\{0, 26-12\} = 14$. É importante notar que a tarefa 2 termina antes de sua data de entrega, por isso o seu atraso é considerado 0. Após estes cálculos, é possível calcular a função objetivo que é dado por $\sum_{j=1}^{4} T_j = 34$.

As soluções foram codificadas como uma lista de vetores de tuplas, sendo que cada vetor de tupla representa uma máquina com suas sub-tarefas e a quantidade de unidade-tarefa das sub-tarefas. Uma solução que tem m máquinas e n tarefas foi codificada do seguinte modo:

Máquina 1:
$[j_1, uj_1], [j_2, uj_2], \ldots, [j_n, uj_n]$

Máquina 2:
$[j_1, uj_1], [j_2, uj_2], \ldots, [j_n, uj_n]$
.
.
.
Máquina m:
$[j_1, uj_1], [j_2, uj_2], \ldots, [j_n, uj_n]$

Sendo que $[j_1, uj_1], [j_2, uj_2], \ldots, [j_n, uj_n]$ representam, respectivamente, a sub-tarefa na primeira posição da máquina em questão e o número de unidades-tarefa daquela sub-tarefa, a sub-tarefa na segunda posição da máquina em questão e o número de unidades-tarefa daquela sub-tarefa e assim por diante, e, finalmente, a sub-tarefa na última posição da máquina em questão e o número de unidades-tarefa daquela sub-tarefa.

O exemplo da Figura 2.1 pode ser expresso (seguindo o modelo, como as soluções foram codificadas) de tal maneira:
Máquina 1:
$[1, 2], [4, 1], [3, 2]$

Máquina 2:
$[2, 2], [1, 1]$

Máquina 3:
$[4, 3], [3, 1]$

2.3 Formulação matemática

Neste trabalho é utilizado o modelo matemático proposto por Sariçiçek e Çelik (2011) para resolver instâncias pequenas do problema. A seguir apresenta-se esse modelo.

2.3.1 Variáveis de decisão

$x_{ijk} = 1$, se a sub-tarefa da tarefa j é processada na máquina i na k-ésima posição; 0, caso contrário.

y_{ijk} = o número de unidades-tarefa da sub-tarefa da tarefa j que é processada na máquina i na k-ésima posição.

T_{ijk} = atraso da sub-tarefa da tarefa j processada na máquina i na k-ésima posição.

2.3.2 Modelo matemático

O atraso total de uma tarefa é computado quando todas as sub-tarefas (desta tarefa) são completadas. Esta afirmação é mostrada pela expressão entre chaves na função objetivo:

$Minimizar \sum_{j=1}^{n} \{Maximize\ T_{ijk}\}, 1 \leq i \leq m, 1 \leq k \leq n$ (1)

Já que a equação da função objetivo, (1), é não linear, uma nova variável, digamos G_j, é definida ($G_j \geq T_{ijk}$) para que (1) torne-se linear.
A equação 2 é a função objetivo, isto é, minimizar o atraso total:

Minimize $\sum_{j=1}^{n} G_j$, (2)
sujeito à

$\sum_{j=1}^{n} x_{ijk} \leq 1,\ k = 1, \ldots, n;\ i = 1, \ldots, m.$ (3)

$\sum_{k=1}^{n} x_{ijk} \leq 1,\ j = 1, \ldots, n;\ i = 1, \ldots, m.$ (4)

$X_{ijk} \leq y_{ijk} \leq u_j x_{ijk},\ i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, n.$ (5)

$\sum_{i=1}^{m} \sum_{k=1}^{n} y_{ijk} = u_j,\ j = 1, \ldots,$ n. (6)

$T_{ijk} \geq \sum_{s=1}^{k} \sum_{l=1}^{n} (p_l y_{ils} + s_l x_{ils})\text{-}d_j\text{-M}(1 - X_{ijk}), i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, n.$ (7)

$T_{ijk} \geq 0,\ i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, n.$ (8)

$X_{ijk} \in \{0,1\},\ i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, n.$ (9)

$Y_{ijk} \geq 0$, inteiro $i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, n.$ (10)

$G_j \geq T_{ijk},\ i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, n.$ (11)

$G_j \geq 0,\ j = 1, \ldots, n.$ (12)

No modelo acima, a função objetivo na restrição (2) envolve calcular o atraso total das tarefas e, em seguida, minimizá-lo. Restrição (3) possibilita o escalonamento da sub-tarefa de no máximo uma tarefa na posição

de qualquer máquina enquanto a restrição (4) certifica que a sub-tarefa de uma tarefa é colocada, no máximo, uma das posições de quaisquer máquinas disponíveis. A restrição (5) estabelece uma conexão entre as variáveis de decisão. Restrição (6) indica que o total de unidades-tarefa de uma tarefa em todas as máquinas e posições é igual ao número de unidades-tarefa daquela tarefa. Unindo a isto, por meio desta restrição, variáveis de decisão são atribuídas indiretamente. Restrição (7) expressa o atraso de uma sub-tarefa de uma tarefa em uma máquina e uma posição. Este atraso é igual à diferença entre a data de entrega de uma sub-tarefa da tarefa em questão mostrada pela primeira expressão no lado direito da restrição e a data de entrega na segunda expressão. A terceira expressão, por outro lado, torna esta restrição ativa ou não de acordo com os valores da variável de decisão x_{ijk}. Se uma sub-tarefa da tarefa j não foi atribuída à posição k da máquina i, nesta expressão x_{ijk} resulta no valor de M. Pelo fato de M ser positivo e um valor muito grande, a restrição em questão se torna nula. Caso contrário ($x_{xijk} = 1$), esta expressão resulta no valor zero e, portanto, o lado direito expressa o atraso em questão. Por outro lado, as restrição restantes representam os valores a serem atribuídos para as variáveis de decisão (Sariçiçek e Çelik, 2011).

2.4 Regras de dominância

Durante a execução de alguma metaheurística para o problema abordado neste trabalho, normalmente muitas soluções são geradas. Várias delas poderiam ser evitadas se algumas regras fossem adotadas na hora de gerar as soluções. Com isso são evitados cálculos desnecessários. As soluções que, eventualmente, seriam evitadas são ditas dominadas (por outras soluções, e estas são ditas dominantes). Uma solução parcial δ é dita dominada por outra solução parcial δ', se há uma solução completa em δ' que seja melhor do que solução completas obtidas de δ (Shim e Kim, 2008). Uma solução s que tenha alguma solução δ dominada por uma solução δ' de outra solução s' é dita dominada por esta solução s'. As regras de dominância propostas por Shim e Kim (2008) são as seguintes:

Regra 1. Considere uma solução δ na qual a sub-tarefa j_1 e a sub-tarefa j_2 da mesma tarefa são atribuídas à mesma máquina e a sub-tarefa j_1 é completada antes do que a sub-tarefa j_2. Então, δ é dominada por outra solução, δ', na qual a solução de sub-tarefas é idêntica àquela em δ, exceto que a sub-tarefa j_1 é movida para a posição imediatamente antes da sub-tarefa j_2 e, a partir daí, estas duas sub-tarefas são unidas a uma única sub-tarefa.

A regra seguinte é relacionada a sub-tarefas atribuídas a máquinas diferentes.

Regra 2. Considere uma solução δ na qual duas sub-tarefas, digamos sub-tarefas j_1 e j_2, da mesma tarefa são atribuídas a máquinas diferentes e o tempo de finalização da sub-tarefa j_1 é menor do que o da sub-tarefa j_2. Se existe uma sub-tarefa, digamos j_0, de outra tarefa que é colocada depois de j_1 na máquina à qual a sub-tarefa j_1 foi atribuída e cujo tempo de finalização não é maior do que o tempo de finalização da sub-tarefa j_2 em δ, então δ é dominado por uma solução δ', na qual solução de sub-tarefas nas máquinas são idênticas àquelas em δ, exceto que a sub-tarefa j_1 é movida para a posição imediatamente depois da sub-tarefa j_0. A Figura 2.2 ilustra esta regra de dominância.

A regra seguinte é relacionada à atribuição de sub-tarefas a máquinas. Denote $\delta j \wedge k$ como uma solução parcial obtida ao unir a sub-tarefa j ao final da solução δ na máquina k.

Regra 3. Considere uma solução δ, na qual uma das sub-tarefas (suponha sub-tarefa j_1) de uma tarefa, digamos tarefa i, já está atribuída à máquina

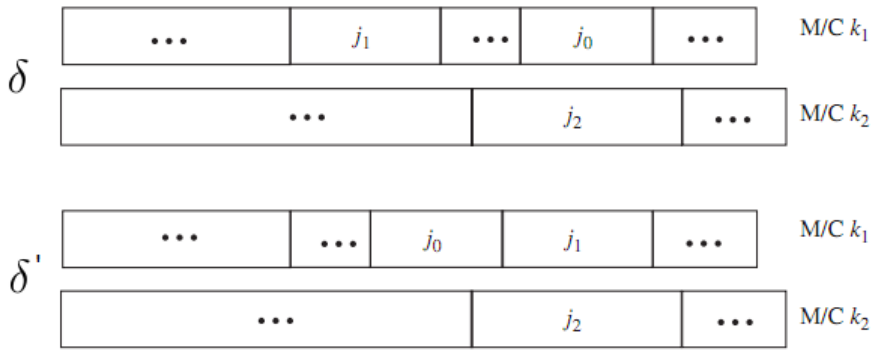


Figura 2.2: Segunda regra de dominância.

k_1 e nenhuma das sub-tarefas dessa tarefa está atribuída à máquina k_2 ($k_1 \neq k_2$). Se $C_{j2}(\delta j_2 \wedge k_1) \leq C_{j2}(\delta j_2 \wedge k_2)$ e $C^M_{k2}(\delta) \leq C^M_{k1}(\delta)$, então a solução parcial $\delta j_2 \wedge k_2$ domina a solução $\delta j_2 \wedge k_1$ (veja figura 2.3).

Na regra 3, $C_{j2}(\delta j_2 \wedge k_2)$ é o tempo de finalização de j_2 na solução δ, estando j_2 na última posição da máquina k_i, e $C^M_{k2}(\delta)$ é o tempo de finalização da última tarefa da máquina k_i.

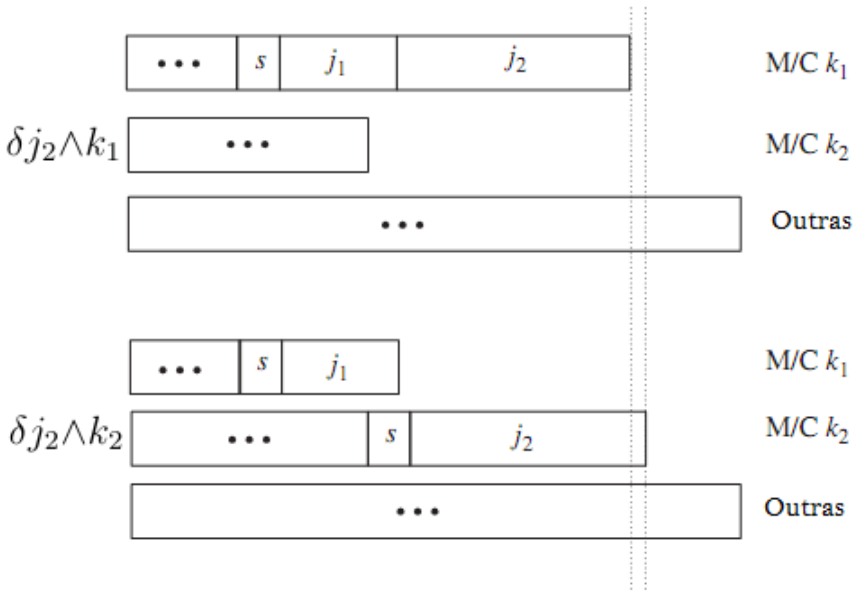


Figura 2.3: Terceira regra de dominância.

3 Revisão bibliográfica

Neste capítulo são apresentadas várias referências bibliográficas ligadas ao problema abordado. Na seção 3.1, serão mostradas as referências de Máquinas Paralelas. Na seção 3.2, serão mostradas as referências de Máquinas Paralelas com divisão de tarefas e, na seção 3.3, serão explicadas as heurísticas de Sariçiçek e Çelik (2011).

3.1 Máquinas Paralelas

Como dito na seção 2.1, McNaughton (1959) introduziu o conceito de máquinas paralelas. Em seu trabalho, ele define conceitos sobre escalonamento de única máquina e várias máquinas. Com relação a esta última definição, ele diferencia máquinas parecidas, exatamente parecidas e diferentes. O conceito de divisão de tarefas também é definido em seu trabalho.

Azizoglu e Kirca (1998) resolveram o problema de minimizar o atraso total no problema de escalonamento em máquinas paralelas idênticas. Os autores propuseram um algoritmo *branch and bound* junto com um esquema de calcular o *lower bound*. Segundo os autores, soluções ótimas podem ser encontradas para problemas com até 15 tarefas.

Mazzini et al (1998) estudam a aplicação de metaheurística populacional em problemas de programação da produção. O problema abordado consiste na minimização do atraso total na programação de tarefas em máquinas paralelas idênticas onde se consideram tempos de preparação de máquina dependentes da sequência de processamento e datas de entregas distintas.

Min e Cheng (1999) apresentaram uma versão, baseada em código de máquina, do Algoritmo Genético com o objetivo de minimizar o *makespan* (máximo tempo de finalização) em máquinas paralelas idênticas. A versão desta metaheurística foi comparada com o método *Simulated Annealing*. De acordo com os autores, a versão do Algoritmo Genético foi melhor do que o método de *Simulated Annealing*.

Müller et al (2002) propõem um novo algoritmo de busca local em conexão com um esquema de vizinhança que usa estrutura de intervalos e o conceito de eficiência das máquinas para cada tarefa. O algoritmo proposto, denominado Mutat, foi criado para resolver o problema de escalonamento de máquinas paralelas não relacionadas com o objetivo de minimizar o *makespan*. O Mutat foi comparado com o algoritmo heurístico proposto por Piersma e Dijk (1996). De acordo com os autores, a nova abordagem encontra soluções que superam, em qualidade e tempo computacional, o melhor algoritmo de busca local encontrado na literatura para este problema.

Mendes et al (2002) resolveram o problema de escalonamento de máquinas

paralelas idênticas com os tempos de preparação dependente de sequência com a finalidade de minimizar o *makespan*. A ideia dos autores foi comparar o desempenho de duas metaheurísticas. A primeira é uma heurística baseada na Busca Tabu e a segunda é uma abordagem Memética, que é chamado pelos autores de MA, que combina um método baseado em população com procedimentos de busca local. De acordo com os autores, os dois algoritmos propostos obtiveram desempenhos comparáveis. Ainda segundo Mendes et al (2002), MA obteve valores muito próximos da Busca Tabu.

Kim et al (2002) resolveram o problema de escalonamento de máquinas paralelas não-relacionadas com tempos de preparação dependente de sequência utilizando o algoritmo *Simulated Annealing* com o objetivo de minimizar o atraso total. A metaheurística desenvolvida foi comparada a um método de descida que utiliza a mesma ideia de perturbação e os movimentos propostos, e, além dela, é comparada a uma versão convencional do *Simulated Annealing*, que realiza movimentos de trocas e inserções de itens. Após a análise dos resultados, a versão que implementou o *Simulated Annealing* com a ideia de perturbações nas soluções foi considerada a melhor pelos autores.

França Filho (2007) desenvolveu metaheurísticas para tratar de dois problemas: o primeiro é o escalonamento em máquinas paralelas idênticas, com o objetivo de minimizar a soma ponderada de custos de atraso; o segundo é o escalonamento de máquinas paralelas não relacionadas, com o objetivo de minimizar a soma ponderada de custos de avanço e de atraso. Em ambos, as transições entre tarefas requerem tempos de preparação dependentes da sequência de processamento. Os problemas são resolvidos por meio de duas heurísticas, GRASP e Busca Tabu. Memória de longo prazo é empregada para melhorar o desempenho das duas metaheurísticas. No GRASP, soluções elite influenciam a fase construtiva. Na Busca Tabu, estratégias de diversificação e de intensificação fazem uso direto das soluções de elite e também de frequências de residência. Como pós-otimização, nas duas metaheurísticas, realizam-se reconexões de caminhos entre as soluções elite. Os autores mostraram, para o primeiro problema, que não há dominância de uma heurística sobre a outra. Com relação ao segundo problema, há uma vantagem das heurísticas da Busca Tabu em relação às do GRASP.

De Paula (2008) propôs um algoritmo Non-Delayed Relax-and-Cut a partir de uma relaxação lagrangeana de uma formulação indexada no tempo para resolver o problema de escalonamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes de sequência com o objetivo de minimizar a soma dos atrasos ponderados das tarefas. De acordo com os autores, foram obtidas, em tempo razoável, boas soluções para instâncias com até seis máquinas e 200 tarefas, e limites inferiores para instâncias com até seis máquinas e 80 tarefas. Os limites inferiores foram particularmente

bons para instâncias fáceis, provando a otimalidade de algumas soluções e provendo gaps estreitos para outras. Para as instâncias mais difíceis, os limites inferiores obtidos não foram tão bons, mas ainda significativos.

Ruiz e Vallada (2009) propuseram duas versões do Algoritmo Genético que inclui uma busca local rápida e uma busca local melhorada com um operador de *crossover* com o objetivo de minimizar o *makespan*. Estas versões foram comparadas com os melhores métodos disponíveis na literatura e, segundo os autores, as versões do Algoritmo Genético superaram todos os outros métodos.

Ruiz e Fanjul-Peyro (2011) estudaram o problema de escalonamento em máquinas paralelas não-relacionadas com o objetivo de minimizar o *makespan*. Segundo os autores, este é um tipo de problema que tem sido frequentemente estudado na literatura científica devido às suas aplicações em potencial. Eles propuseram um conjunto de metaheurísticas baseadas em uma redução de tamanho e testaram com métodos considerado, por eles, estado-da-arte atuais. De acordo com os autores, na maioria dos casos, os algoritmos de redução de tamanho propostos produziram resultados, estatisticamente, melhores.

3.2 Máquinas Paralelas com divisão de tarefas

Serafini (1996) e Xing e Zhang (2000) estudaram casos no qual cada tarefa pode ser dividida arbitrariamente (em sub-tarefas de unidades contínuas, não discretas) e processadas independentemente em máquinas paralelas uniformes ou não relacionadas. Serafini (1996) criou algoritmos heurísticos com o objetivo de minimizar o máximo atraso ponderado e Xing e Zhang (2000) propuseram um algoritmo heurístico com o objetivo de minimizar o *makespan*.

Kim et al (2004) estudaram casos, nos quais, cada tarefa pode ser dividida em um número discreto de sub-tarefas e elas podem ser processadas em máquinas paralelas independentemente. Eles sugeriram uma heurística de duas fases para o problema de escalonamento em máquinas paralelas com a propriedade de divisão de tarefas com o objetivo de minimizar o atraso total.

Tahar et al (2006) apresentaram um método baseado na programação linear para escalonamento de máquinas paralelas idênticas com propriedade de divisão de tarefas e os tempos de preparação dependentes de sequência.

Shim e Kim (2008) sugeriram um algoritmo *Branch and Bound* para problema de escalonamento em máquinas paralelas com divisão de tarefas com o objetivo de minimizar o atraso total. Eles desenvolveram regras de dominância e limites inferiores e usaram-nos em seu algoritmo *Branch and Bound*. Os resultados dos seus experimentos computacionais mostraram que

o algoritmo sugerido conseguiu achar a solução ótima para problemas de até 4 máquinas e 12 tarefas (e cinco máquinas e oito tarefas) em uma quantidade razoável de tempo de CPU. De acordo com esse trabalho, pode-se criar heurísticas que resultam em muito boas soluções para problemas de grande porte dentro de um tempo computacional razoavelmente curto.

3.3 Heurísticas *Simulated Annealing* e Busca Tabu de Sariçiçek e Çelik (2011) para o problema de Máquinas Paralelas com Divisão de Tarefas

Nesta seção serão explicadas as heurísticas propostas por Sariçiçek e Çelik (2011). A primeira a ser explicada é chamada de *Simulated Annealing*, referida como SA e a outra é a Busca Tabu, referida como TS.

Como solução inicial, Sariçiçek e Çelik (2011) utilizaram a regra EDD (Earliest Due Date), que é uma regra comumente empregada para produzir soluções iniciais.

Na geração de vizinhos, uma estrutura híbrida, consistindo de trocas e inserções foi adaptada. De um modo geral, um movimento de inserção identifica duas tarefas particulares e coloca a primeira tarefa no lugar que, diretamente, precede o lugar da segunda. Um movimento de troca, por outro lado, coloca cada tarefa no lugar, previamente ocupada pela outra, não importando se são consideradas máquinas iguais ou diferentes. A inserção é realizada sobre unidades-tarefa da mesma tarefa em duas máquinas diferentes. As trocas de tarefas, entretanto, são realizadas somente entre tarefas na mesma máquina. O procedimento de lista de candidato foi definido como um procedimento que é usado para formar um sub-grupo de uma vizinhança inteira e o tamanho deste sub-grupo é controlado. De um modo geral, quanto maior a vizinhança é, menor a probabilidade de se obter um mínimo local (Glover e Laguna, 1997).

Já que a vizinhança gerada por movimentos de inserção é maior do que a gerada por trocas de tarefas, a estratégia da lista de candidatos é realizada sobre movimentos de inserção. Portanto, duas estratégias de lista de candidatos são utilizadas (uma após a outra) nos dois algoritmos, *Simulated Annealing* e Busca Tabu, de forma que a dificuldade de calcular todos os vizinhos é minimizada por um lado, e soluções de qualidade são obtidas, por um outro lado. Uma delas, como ditas no estudo por Bilge et al (2004), é uma estratégia de lista de candidatos que é baseada em escolher a máquina que mais contribui para o atraso total calculando o atraso de cada máquina e envolve aplicar ação de inserção desta máquina para todas as outras máquinas. E a outra é, como mencionada no estudo por Kim et al (2004) dentre as

regras para escolher uma máquina, uma estratégia de lista de candidatos é baseada na escolha da máquina com a menor carga de trabalho (soma total dos tempos de preparação e processamento da máquina em questão) e, na qual, são realizados movimentos de inserção de outras máquinas para esta máquina (Sariçiçek e Çelik 2011).

3.3.1 *Simulated Annealing* - SA

Uma das heurísticas implementadas por Sariçiçek e Çelik (2011), que é usada para comparar com as heurísticas do trabalho, foi baseada no *Simulated Annealing*.

Segundo Baykasoglu, 2003, "o *Simulated Annealing* é um processo de busca que tem sua origem nos campos da física e ciência dos materiais. Foi desenvolvido primeiramente como um modelo de simulação para descrever o processo de aquecimento físico de matéria condensada. Este método tem a capacidade de saltar de ótimos locais para otimização global. Esta capacidade é conseguida aceitando, com uma probabilidade, soluções vizinhas que são piores do que a solução atual. A probabilidade de aceitação é determinada por um parâmetro de controle que diminui durante o procedimento do método".

O procedimento do *Simulated Annealing* passa por um número de iterações. A cada iteração do procedimento, há uma solução atual S_k, inicialmente gerada através do procedimento *GreedyRandomized* com o parâmetro 0,3, que serão explicados detalhadamente na subseção 4.2.1, e, também, a melhor solução encontrada até o momento, S_0. Sejam $G(S_k)$ e $G(S_0)$ os valores das funções objetivas de S_k e S_0, respectivamente. Note que $G(S_k) \geq G(S_0)$ O valor da melhor solução obtida até o momento, $G(S_0)$, é geralmente referenciada como o critério de aspiração. A cada iteração k, uma busca por uma nova solução é conduzida dentro da vizinhança de S_k. A solução candidata S_k é selecionada da vizinhança. Se $G(S_c) < G(S_k)$, um movimento é feito atribuindo $S_{k+1} = S_c$ e, se $G(S_c) < G(S_0)$, então também é atribuído igual a S_c. Entretanto, se $G(S_c) \geq G(S_k)$ um movimento é feito em S_c com probabilidade: $P(S_k, S_c) = exp(\frac{G(S_k) - G(S_c)}{ß_k})$. Com a probabilidade de 1 - $P(S_k, S_c)$, a solução S_c é rejeitada em favor da solução atual, atribuindo $S_{k+1} = S_k$. ß é o parâmetro de controle referido como a temperatura de esfriamento (Baykasoglu, 2003). Geralmente $ß_k$ é escolhido como a^k para um a escolhido aleatoriamente entre 0 e 1. No procedimento do *Simulated Annealing*, movimentos para soluções piores são permitidos. A razão para permitir estes movimentos é dar oportunidade ao procedimento de fugir de mínimos locais e achar uma solução melhor mais tarde. Já que $ß_k$ diminui

com k, a probabilidade de aceitação para um movimento de não-melhoria é mais baixa em iterações mais avançadas do processo de busca. A definição da probabilidade de aceitação também garante que, se um vizinho é significativamente pior, sua probabilidade de aceitação é muito baixa e um movimento é improvável de ser feito (Pinedo et al, 2002).

A descrição do algoritmo do *Simulated Annealing* é a que se segue no Algoritmo 3.1 (Sariçiçek e Çelik, 2011).

Algoritmo 3.1: Pseudocódigo do *Simulated Annealing*, segundo Sariçiçek e Çelik (2011).

1: **SimulatedAnnealing**(N):
2: $K \leftarrow 1$ e selecione ß$_1$;
3: $S_1 \leftarrow$ *GreedyRandomized(0,3)*;
4: $S_0 \leftarrow S_1$;
5: Selecione uma solução candidata S_c da vizinhança de S_k;
6: **if** $f(S_0) < f(S_k)$ *e* $f(S_k) < f(S_k)$ **then**
7: $S_{k+1} \leftarrow S_c$;
8: vá para o passo 24;
9: **end if**
10: **if** $f(S_c) < f(S_0)$ **then**
11: $S_{k+1} \leftarrow S_c$;
12: $S_0 \leftarrow S_{k+1}$;
13: vá para o passo 24;
14: **end if**
15: **if** $f(S_c) \geq f(S_k)$ **then**
16: $U_k \leftarrow$ número aleatório entre 0 e 1;
17: **if** $U_k \leq P(S_k, S_c)$ **then**
18: $S_{k+1} \leftarrow S_c$;
19: **else**
20: $S_{k+1} \leftarrow S_k$;
21: vá para o passo 24;
22: **end if**
23: **end if**
24: Selecione ß$_{k+1} \leq$ ß$_k$;
25: $k \leftarrow k + 1$;
26: **if** $k = N$ **then**
27: retorne S_0;
28: **else**
29: vá para o passo 5;
30: **end if**

3.3.2 Busca Tabu - TS

A Busca Tabu é uma metaheurística proposta por Glover (1989). Ela pode ser descrita como uma técnica de busca local e melhoramento de uma heurística bem conhecida do tipo *Hill Climbing* (ou em português, "Subida da Colina") (Hatami et al, 2010).

O método da Busca Tabu é, de muitas maneiras, parecido ao metódo do

Simulated Annealing em que também move de uma solução à outra, possivelmente pior do que a primeira. Para cada solução, uma vizinhança é definida como no *Simulated Annealing*. A busca na vizinhança para um candidato em potencial é um problema de *design*. Como no *Simulated Annealing*, isto pode ser feito aleatoriamente ou de um modo organizado. A diferença básica entre a busca tabu e o *Simulated Annealing* está no mecanismo que é usado para aprovar a solução candidata. Na Busca Tabu o mecanismo não é probabilístico, mas de natureza determinística. Em qualquer estágio do processo uma lista tabu de mutações, cujo procedimento não é permitido a fazer, é mantido. Uma mutação na lista tabu pode ser, por exemplo, um par de tarefas que não podem ser trocadas de posição. A lista tabu tem um número fixo de entradas (normalmente entre 5 e 9) que depende da aplicação (Pinedo et al, 2002). Cada movimento que é feito por uma certa mutação na solução corrente, a mutação reversa é colocada no topo da lista tabu; todas as outras entradas na lista tabu são jogadas para baixo uma posição e a última entrada da lista é deletada. A mutação reversa é posta na lista tabu para evitar de retornar a um mínimo local que foi visitado anteriormente. Na verdade, às vezes uma mutação reversa que é tabu poderia, de fato, ter levado a uma nova solução, não visitada antes que fosse melhor do que qualquer uma gerada até o momento. Isso pode acontecer quando a mutação está perto do fim da lista tabu e um número de movimento já foi feito desde que a mutação foi posta na lista. Portanto, se o número de entradas na lista tabu é muito pequeno, ciclos podem ocorrer; se é muito grande, a busca pode ser indevidamente limitada (Pinedo et al, 2002).

A descrição do algoritmo é a que se segue no Algoritmo 3.2 (Sariçiçek e Çelik, 2011):

Algoritmo 3.2: Pseudocódigo da Busca Tabu, segundo Sariçiçek e Çelik (2011).

1: **BuscaTabu**(N):
2: $k \leftarrow 1$;
3: $S_1 \leftarrow$ *GreedyRandomized(0,3)*;
4: $S_0 \leftarrow S_1$;
5: Selecione uma solução candidata S_c da vizinhança de S_k;
6: **if** movimento $S_k \rightarrow S_c$ for proibido por uma mutação na lista tabu **then**
7: $S_{k+1} \leftarrow S_k$;
8: vá para o passo 20;
9: **end if**
10: **if** movimento $S_k \rightarrow S_c$ não for proibido por uma mutação na lista tabu **then**
11: $S_{k+1} \leftarrow S_c$;
12: **end if**
13: coloque a mutação reversa no topo da lista tabu;
14: jogue todas as entradas na lista tabu uma posição abaixo;
15: delete a entrada no final da lista tabu;
16: **if** $G(S_c) < G(S_0)$ **then**
17: $S_0 = S_c$;
18: **end if**
19: vá para o passo 20;
20: $k = k + 1$;
21: **if** $k = N$ **then**
22: retorne S_0;
23: **else**
24: vá para o passo 4;
25: **end if**

4 Heurísticas propostas

Neste capítulo, são descritas as versões das heurísticas propostas: GRASP Reativo, GRASP Reativo com *Path-Relinking*, Algoritmo Genético Básico e Algoritmo Genético com *Path-Relinking*.

4.1 Heurística GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma heurística de múltiplos reinícios, proposto por Feo e Resende (1995), onde, em cada iteração, uma solução é construída para ser usada como uma solução inicial para a fase de busca local. Busca local é um procedimento que tenta melhorar a solucção corrente. Uma solução vizinha é obtida realizando um movimento na solução corrente. Se não há nenhuma solução melhor na vizinhança, a solução atual é declarada como mínimo local e a busca para. O melhor mínimo local encontrado em todas as iterações do GRASP é a saída como a solução (Resende et al, 2010).

GRASP foi aplicado com sucesso para resolver muitos problemas de otimização (Festa e Resende, 2009). Seu pseudocódigo é mostrado no Algoritmo 4.1.

Algoritmo 4.1: Pseudocódigo do GRASP básico.

```
1:  GRASP(α):
2:  s* ← GreedyRandomized(α);
3:  while critério de parada não for satisfeito do
4:     s' ← GreedyRandomized(α);
5:     s' ← BuscaLocal(s');
6:     if f(s') < f(s*) then
7:        s* ← s';
8:     end if
9:  end while
10: retorne s*;
```

O Algoritmo 4.1 possui um parâmetro de entrada α, que controla o nível de aleatoriedade, e é utilizado na etapa de construção, que é o método *Greed Randomized* das linhas 2 e 4. Na seção 4.2.1, este método será explicado com detalhes. A solução que é retornada do método de construção é utilizado como parâmetro para a busca local na linha 5. O algoritmo mantém uma solução corrente s^* que é considerada a melhor solução encontrada para ser retornada na linha 10 e esta solução é atualizada toda vez que a solução retornada da busca local seja melhor do que ela (s^*).

4.2 GRASP Reativo - GR

O parâmetro α, do algoritmo construtivo, é o único parâmetro a ser testado numa implementação prática do GRASP (Prais e Ribeiro, 2000). Feo e Resende (1995) discutiram o efeito da escolha do valor de α em termos de qualidade de solução e diversidade durante a fase de construção e qual o impacto no resultado do GRASP. Prais e Ribeiro (2000) mostraram um novo procedimento chamado GRASP Reativo para o qual o parâmetro α é auto-ajustável de acordo com a qualidade das soluções previamente encontradas.

Prais e Ribeiro (2000), ao invés de usar um valor fixo para o parâmetro α, que determina quais elementos serão colocados na lista de candidatos restrita a cada iteração da fase de construção, propuseram selecionar α aleatoriamente de um conjunto discreto $A = \{\alpha_1, \ldots, \alpha_v\}$ contendo v valores pré-determinados. Usar valores diferentes de α em diferentes iterações permite construir diferentes listas de candidatos restritas, eventualmente levando à construção de soluções diferentes que, provavelmente, não seriam construídas se um valor fixo de α fosse usado. Além do conjunto A, há, também, o conjunto discreto P cujos valores P_i são as probabilidades associadas a α_i com os valores, inicialmente, de $P_i = 1/v$, $i = 1, \ldots, v$ (Prais e Ribeiro, 2000).

O GRASP Reativo funciona do seguinte modo: seja v um número de valores permitidos de α e seja A = $\{\alpha_1, \alpha_2, \ldots, \alpha_v\}$ a lista que contem os valores. Inicialmente, cada α_k tem a mesma probabilidade para ser selecionada, isto é, $P_k = 1/v$. Em cada iteração, um valor α_k é escolhido com uma probabilidade de P_k. As probabilidades P_k são recalculadas a cada γ iterações de tal forma que os valores de α_k, que produziriam as melhores soluções, são aqueles que tem mais chance de serem selecionados. O método do GRASP Reativo (Algoritmo 4.2) é similar ao GRASP básico. A diferença do GRASP Reativo é o auto-ajuste do parâmetro α de acordo com a qualidade das soluções previamente encontradas.

O Algoritmo 4.2 inicia com o conjunto definido de valores de alfa (passo 2). No passo 4, há a inicialização dos valores das variáveis usadas no mecanismo reativo. Estas variáveis são atualizadas nos passos 18, 19 e 27. No passo 9, um índice k é escolhido, com probabilidade P_k. Com este índice, tem-se o α_k (valor de α escolhido), o $count_k$ (variável que mantém o número de vezes que o α_k foi escolhido) e $score_k$ (variável que indica o desempenho das soluções geradas a partir do α_k escolhido). No passo 10, a variável gi (GRASP *iteration*) tem a finalidade de controlar o número da iteração. Isso é necessário para o passo 20. Nele, o algoritmo verifica se a iteração é múltipla de γ para atualizar os valores de P_k. Os passos de 11 a 15 são os mesmos passos do Algoritmo 4.1. Os demais passos fazem parte do mecanismo reativo. A ideia deste mecanismo é o seguinte: o algoritmo verifica, para cada α escolhido na

iteração, o valor da solução gerada ao final da iteração do GRASP. A média de todos os valores das soluções geradas a partir deste determinado α é computada ao longo das iterações para calcular a sua probabilidade. Portanto, se um determinado α está gerando soluções de má qualidade, então este α tende a ter uma baixa probabilidade de ser escolhido. Por outro lado, se um determinado α está gerando soluções com boa qualidade, a sua probabilidade aumenta. Todo este processo começa no passo 18 e vai até o final.

Algoritmo 4.2: Pseudocódigo do GRASP Reativo

```
1: GRASPReativo(θ, γ):
2: A ← {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
3: v ← |A|;
4: for i = 1 to v do
5:    P_i ← 1/v;
6:    count_i ← 0;
7:    score_i ← 0;
8: end for
9: k ← índice do valor de α escolhido aleatoriamente com a probabilidade
   P_k;
10: gi ← 0; // Grasp iteration: contador do laço principal.
11: while critério de parada não for satisfeito do
12:    s' ← GreedyRandomized(α_k);
13:    s' ← BuscaLocal(s');
14:    if f(s') < f(s*) then
15:       s* ← s';
16:    end if
17:    gi ← gi + 1;
18:    count_k ← count_k + 1;
19:    score_k ← score_k + f(s');
20:    if gi é múltiplo de γ then
21:       for k=1 to v do
22:          avg_k ← score_k/count_k;
23:          Q_k ← (f(s*)/avg_k)^θ;
24:          σ ← σ + Q_k;
25:       end for
26:       for k=1 to v do
27:          P_k ← Q_k/σ;
28:       end for
29:    end if
30:    k ← índice do valor de α escolhido aleatoriamente com a
       probabilidade P_k;
31: end while
32: retorne s*;
```

4.2.1 Heurística construtiva

No início da Heurística Construtiva, Algoritmo 4.3, as tarefas são ordenadas pela regra EDD (Earliest Due Date), ou seja, as tarefas são ordenadas em ordem crescente de suas datas de entrega. As tarefas ordenadas formam a lista L de tarefas candidatas a serem inseridas nas máquinas. A cada iteração da heurística escolhe-se aleatoriamente uma tarefa dentre as primeiras $max\{1, \lfloor \alpha \times |L| \rfloor\}$ tarefas da lista. Esta tarefa é inserida na máquina que produz o menor atraso total considerando as tarefas já inseridas nas máquinas. O algoritmo finaliza quando todas as tarefas já forem inseridas nas máquinas, ou seja, quando $L = \emptyset$.

Algoritmo 4.3: Pseudocódigo da heurística construtiva.

1: **GreedyRandomized**(α):
2: $L \leftarrow$ lista de tarefas ordenadas de acordo com as suas datas de entegas.
3: $s^* \leftarrow$ solução vazia;
4: **while** L $\neq \emptyset$ **do**
5: $w \leftarrow$ tarefa escolhida, aleatoriamente, entre as $max\{1, \lfloor \alpha \times |L| \rfloor\}$;
6: $z \leftarrow$ máquina de s^* que gera a melhor função objetivo;
7: inserir a tarefa w na máquina z de s^*;
8: $L \leftarrow L$ - $\{w\}$;
9: **end while**
10: $s^* \leftarrow PrimeiraRD(s^*)$;
11: *retorne* s^*;

Um exemplo numérico pode ser dado. Suponha que, depois de ordenar as tarefas, houvesse a seguinte lista de sub-tarefas {1, 1, 1, 3, 3, 3, 2, 2, 2} e seja α igual a 0,3. A primeira tarefa que será escolhida é aquela (tomada aleatoriamente) para as primeiras $max\{1, \lfloor 0{,}3\text{x}9 \rfloor = 2\}$ tarefas. É preciso ter certeza de que, pelo menos, uma tarefa será candidata e esta é a razão pela qual é usado $max\{1, \lfloor \alpha \text{x} n \rfloor\}$. Após escolher a primeira tarefa, precisa-se escolher a máquina onde a tarefa será inserida. Esta máquina será aquela que gera uma solução com a menor função objetivo. Depois disso, remove-se esta tarefa da lista e escolhe-se outra tarefa repetindo o procedimento até que todas as tarefas sejam removidas da lista.

4.2.2 Busca Local

A busca local é uma família de técnicas não exaustivas de propósitos gerais para problemas de otimização. Um algoritmo de busca local começa de uma solução inicial s_0 e entra em um *loop* que navega pelo espaço de busca, caminhando de um estado s_i a outro de seus vizinhos s_{i+1}. A vizinhança é normalmente composta por soluções que são obtidas por algumas mudanças locais (chamadas movimentos) do estado atual. Diferentes técnicas de busca local podem ser combinadas e alternadas para dar origem a complexos algoritmos (Schaerf, 2002).

Depois que uma solução é criada a partir do algoritmo construtivo, a fase de busca local é usada para melhorar a solução. Nesta fase, a vizinhança da solução é gerada inserindo tarefas de uma máquina à outra. Depois da inserção, a regra de dominância é aplicada a fim de melhorar as soluções.

Neste trabalho foram propostas duas buscas locais. Os Algoritmos 4.4 e 4.5 mostram um pseudocódigo das buscas locais 1 e 2, respectivamente.

4.2.2.1 Busca Local 1 - BL1

Nesta busca local, uma máquina é escolhida aleatoriamente da e, a partir dela, uma tarefa desta máquina é escolhida com uma certa probabilidade *prob*. Esta tarefa é reinserída na melhor posição em outra máquina. Esta outra máquina é escolhida dentre todas as outras máquinas como sendo aquela que resulta num melhor valor de função objetivo, isto é, a tarefa é testada em todas as máquinas e, aquela que gera a solução com a melhor função objetivo é escolhida. Os parâmetros de entrada do Algoritmo 4.4 são: solução s^* e *prob*. Pela solução s^*, outras soluções serão geradas (soluções vizinhas) a partir de movimentos. O parâmetro *prob* é a probabilidade de ocorrer um movimento em BL1. Um movimento é uma ação, através da qual, outra solução é gerada. Ele significa que uma tarefa, de uma máquina, é re-inserida em outra posição (diferente da sua original).

4.2.2.2 Busca Local 2 - BL2

Nesta busca local, a máquina, com maior somatório dos tempos de processamento e preparação, é escolhida. A partir dela, uma tarefa é removida e reinserída na melhor posição em outra máquina. Esta outra máquina é escolhida dentre todas as outras máquinas como sendo aquela que resulta num melhor valor de função objetivo, isto é, a tarefa é testada em todas as máquinas e, aquela que gera a solução com a melhor função objetivo é escolhida.

Algoritmo 4.4: Pseudocódigo da busca local 1.

```
1: BuscaLocalBL1(s*, prob):
2: continuar ← true;
3: while continuar do
4:    continuar ← false;
5:    for cada máquina z não vazia do
6:       for cada tarefa w de z, escolhida aleatoriamente do
7:          if prob for satisfeita then
8:             s' ← melhor solução após inserir w na melhor posição em
               todas as máquinas;
9:             if f(s') < f(s*) then
10:               s* ← s';
11:               continuar ← true;
12:               vá para o passo 3;
13:            end if
14:         end if
15:      end for
16:   end for
17: end while
18: retorne s*;
```

Algoritmo 4.5: Pseudocódigo da busca local 2.

```
1: BuscaLocalBL2(s*):
2: continuar ← true;
3: while continuar do
4:    continuar ← false;
5:    z ← máquina com maior somatório dos tempos de preparação e
      processamento de z;
6:    for cada máquina z' não vazia, exceto z do
7:       for cada tarefa w de z, escolhida aleatoriamente do
8:          s' ← melhor solução após inserir w na melhor posição em todas
            as máquinas, exceto z;
9:          if f(s') < f(s*) then
10:            s* ← s';
11:            continuar ← true;
12:            vá para o passo 3;
13:         end if
14:      end for
15:   end for
16: end while
17: retorne s*;
```

4.2.3 Path Relinking

Path relinking foi proposto por Glover (1996) como uma estratégia de intensificação explorando trajetórias conectando soluções elite obtidas pela Busca Tabu ou a *Scatter Search* (Glover et al, 2000; Glover e Laguna, 1997; Glover e Martí, 2000). Começando de uma solução, são gerados e explorados vários caminhos no espaço da solução levando a outras soluções elites na busca por melhores soluções. Para gerar os caminhos, movimentos são selecionados para introduzir atributos na solução corrente que estão presentes na solução elite guia. *Path relinking* pode ser visto como uma estratégia que procura incorporar atributos de soluções de alta qualidade, favorecendo estes atributos nos movimentos selecionados.

O Algoritmo 4.7 ilustra o pseudocódigo do *Path Relinking* aplicado a um par de soluções s_s (solução inicial) e s_t (solução alvo). O procedimento começa criando uma solução s^* que é a solução que será retornada pelo procedimento. Cada vez que uma solução encontrada for melhor do que s^*, ela passará a ser s^*. O *loop* principal só será interrompido quando s_s for igual a s_t. O que é feito em cada iteração é, basicamente, reinserir uma tarefa, a partir de s_s em uma máquina de s_t (formando a solução s_l) de forma que diminua a diferença entre as soluções s_l e s_t. A melhor solução encontrada nesse processo passará a ser a s_s. Essa diferença é o número de movimentos necessários para uma solução (s_l) alcançar a solução (s_t). Um caminho de soluções é gerado ligando s_s com s_t. A melhor solução s^* neste caminho é retornada pelo algoritmo. Se necessário, a melhor solução s^* é atualizado. Ao terminar o *loop* principal, a solução s^* é retornada. A Figura 4.1 mostra um exemplo do procedimento *Path Relinking* para uma instância com duas máquinas e três tarefas. O valor das funções objetivo são mostrados como valores arbitrários. Em cada passo, a solução escolhida está colorida de vermelho e a tarefa reinserida está em negrito.

Nota-se que o *Path Relinking* também pode ser visto como uma estratégita de busca local restrita aplicada à solução inicial s_s, na qual só um conjunto limitado de movimentos podem ser feitos. Muitas alternativas foram consideradas e combinadas em implementações do *Path Relinking* (Rosseti et al, 2003; Ribeiro et al, 2002; Ribeiro e Rosseti 2002; Alex et al, 2002; Resende et al, 2003; Alex et al, 2005; Binato et al, 2001), entre elas:

– *forward relinking*: o *path relinking* é aplicado usando a pior solução entre s_s e s_t como a solução inicial e a outra como solução alvo;

– *backward relinking*: o *path relinking* é aplicado usando a melhor solução entre s_s e s_t como a solução inicial e a outra como solução alvo. Esta abordagem foi utilizada nas heurísticas propostas;

– *mixed relinking*: dois caminhos são explorados simultaneamente, o pri-

Algoritmo 4.6: Pseudocódigo do Path Relinking.

```
1: PathRelinking(s_s, s_t):
2: s* ← s_s;
3: f* ← f(s*);
4: while s_s ≠ s_t do
5:    for cada sub-tarefa l em s_s do
6:       s_l ← inserir em cada posição de cada máquina de s_t de forma que
         diminua a diferença entre s_l e s_t;
7:       if f(s_l) <  f* then
8:          s* ← s_l;
9:          f* ← f(s_l);
10:      end if
11:      s_s ← s_l;
12:   end for
13: end while
14: retorne s*;
```

meiro começando de s_s e o segundo de s_t, até que eles se encontrem em uma solução intermediária equidistante de s_s e s_t;

– *randomized relinking*: ao invés de selecionar o melhor movimento não selecionado até então, este método seleciona aleatoriamente um dentre uma lista candidata com os movimentos mais promissores no caminho investigado.

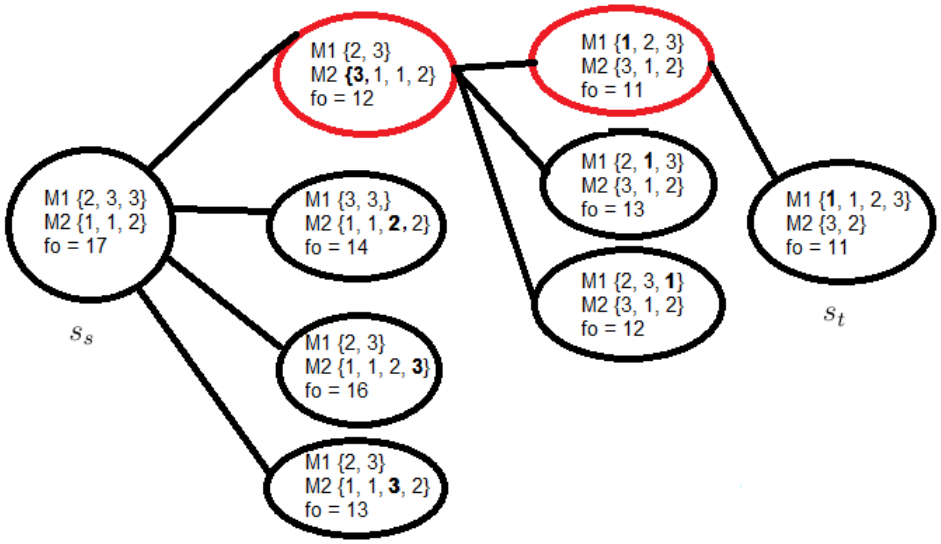


Figura 4.1: Exemplo gráfico do funcionamento do *Path-Relinking*

4.3 GRASP Reativo com Path-Relinking - GPR

As iterações do GRASP são independentes, i.e., soluções encontradas em iterações anteriores não influenciam o algoritmo na iteração corrente. O uso de soluções previamente encontradas para influenciar o procedimento na iteração atual pode ser pensado como um mecanismo de memória (Resende et al, 2010). Um modo de incorporar memória no GRASP é com *Path-Relinking* (Glover, 1996; Glover e Martí, 2000). No GRASP com *Path-Relinking*, um conjunto de diversas soluções de boa qualidade é mantido para ser usado durante cada iteração do GRASP. A este conjunto dá-se o nome de conjunto Elite. Depois que uma solução é gerada na fase construtiva e busca local, ela é combinada com outra solução escolhida aleatoriamente do conjunto elite é submetido ao procedimento de *Path-Relinking*. A melhor das soluções combinadas é uma candidata para inclusão no conjunto elite e é adicionada ao conjunto elite se os critérios de qualidade e diversidade são satisfeitos (Laguna e Martí, 1999; Resende e Ribeiro, 2005).

No Algoritmo 4.6, o pseudocódigo do GRASP Reativo com *Path Relinking* é mostrado. Os parâmetros usados são os mesmos do GRASP básico, exceto no *EliteSize* que é a variável usada para o tamanho do conjunto Elite.

Os passos do Algoritmo 4.6 são semelhantes com os passos do Algoritmo 4.2. É acrescido o conjunto Elite que guardará as melhores soluções durante

as iterações. Ele é iniciado com valores aleatórios, e, à medida que ocorrem as iterações, seus elementos iniciais são substituídos por soluções melhores. Na linha 15, uma solução *sa* é escolhida para ser usada como parâmetro no *Path-Relinking* da linha 16. As linhas 17 e 18 são as segunda e terceira regras de dominância, respectivamente, sendo aplicadas nas soluções. Estas funções procuram por uma solução dominante à solução passada no parâmetro, sem que piore a função objetivo desta. Na linha 19, a solução s', após passar por todos os processos iniciados na linha 13 e terminados na linha 18, é inserido no conjunto *Elite* passando por duas verificações: a solução deve ser diferente dos elementos que já estão no conjunto e ela tem que ser melhor do que, pelo menos, a pior solução no conjunto. A linha 23 em diante se refere ao início do GRASP reativo para atualização das variáveis utilizadas nele.

4.4 Algoritmo Genético

Segundo Bastos e Ochi (2008), "algoritmos genéticos são métodos evolutivos, que podem ser usados para resolver problemas de otimização e busca (Beasley et al, 1993). Eles são baseados no processo genético de organismos biológicos. Por várias gerações, populações naturais evoluíram de acordo com os princípios da seleção natural, isto é, sobrevivência do mais adaptado, dito primeiramente por Charles Darwin em "A origem das espécies pela Seleção Natural". Algoritmos Genéticos simulam os processos populacionais naturais que são essenciais à evolução. Assim, é possível "evoluir" soluções para problemas do mundo real se eles forem enquadrados de forma adequada".

Ainda de acordo com Bastos e Ochi (2008), "uma solução para um problema pode ser representada como um conjunto de parâmetros. Estes parâmetros (conhecidos como genes) são unidos para formar uma sequência de valores (cromossomos). Na terminologia genética, o conjunto de parâmetros representado por um cromossomo particular é referido como um indivíduo. O *fitness* de um indivíduo depende do seu cromossomo e é avaliado pela função de *fitness*. Durante a fase reprodutiva, os indivíduos são selecionados da população e recombinados, produzindo uma geração que compreende a próxima geração. Pais são selecionados aleatoriamente da população usando um esquema que favorece os indivíduos mais adaptados. Ao selecionar dois pais, seus cromossomos são recombinados, tipicamente usando mecanismos de *crossover* e mutação. Mutação é aplicada a alguns indivíduos para garantir diversidade populacional".

Na linha 2 do Algoritmo 4.8, tem a geração da população inicial (soluções iniciais). O passo 3 de avaliar a população P_0 significa simplesmente calcular o valor da função objetivo de cada solução. No ciclo principal, uma nova população (conjunto de soluções) é gerada a partir da população corrente.

Algoritmo 4.7: Pseudocódigo do GRASP Reativo com Path Relinking

1: $GRASPReativoComPR(\gamma, \theta, probBL, EliteSize)$:
2: $\alpha \leftarrow \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$;
3: $v \leftarrow |\alpha|$;
4: **for** $i = 1$ to v **do**
5: $\quad P_k \leftarrow 1/v$;
6: $\quad count_k \leftarrow 0$;
7: $\quad score_k \leftarrow 0$;
8: **end for**
9: $k \leftarrow$ *índice do valor de* α *escolhido aleatoriamente à probabilidade* P_k;
10: $gi \leftarrow 0$; # Grasp iteration: contador do laço principal.
11: $Elite \leftarrow$ soluções aleatórias;
12: **while** critério de parada não for satisfeito **do**
13: $\quad s' \leftarrow GreedyRandomized(\alpha_k)$;
14: $\quad s' \leftarrow BuscaLocal(s')$;
15: $\quad sa \leftarrow$*SoluçãoAleatória(Elite)*;
16: $\quad s' \leftarrow PathRelinking(s', sa)$;
17: $\quad s' \leftarrow SegundaRD(s')$;
18: $\quad s' \leftarrow TerceiraRD(s')$;
19: $\quad adicionar(s', Elite)$;
20: $\quad gi \leftarrow gi + 1$;
21: $\quad count_k \leftarrow count_k + 1$;
22: $\quad score_k \leftarrow score_k + f(s')$;
23: $\quad$ **if** gi é múltiplo de γ **then**
24: $\quad\quad$ **for** k=1 to v **do**
25: $\quad\quad\quad avg_k \leftarrow score_k/count_k$;
26: $\quad\quad\quad Q_k \leftarrow (f(s^*)/avg_k)^\theta$;
27: $\quad\quad\quad \sigma \leftarrow \sigma + Q_k$;
28: $\quad\quad$ **end for**
29: $\quad\quad$ **for** k=1 to v **do**
30: $\quad\quad\quad P_k \leftarrow Q_k/\sigma$;
31: $\quad\quad$ **end for**
32: $\quad$ **end if**
33: $\quad k \leftarrow$ índice do valor de α escolhido aleatoriamente com probabilidade P_k;
34: **end while**
35: $s^* \leftarrow melhorSolução(Elite)$;
36: *retorne* s^*;

Esta nova população será utilizada como população corrente da próxima iteração. Isso significa que uma população influencia a geração de uma outra população. Esta nova população que criada é feita a partir de algumas soluções da população corrente tendo sofrido *crossover* (quando uma nova solução é gerada a partir de duas outras) e/ou mutação (quando uma solução sofre uma pequena alteração) e algumas soluções da população corrente sem ter sofrido nenhuma alteração (passos 6, 7 e 8). Ao final do algoritmo, a melhor solução da população corrente P_g é retornada.

Algoritmo 4.8: Pseudocódigo do Algoritmo Genético.

```
1:  AlgoritmoGenético():
2:  P_0 ← gerar população inicial;
3:  Avaliar população P_0;
4:  g ← 0; // g = geração. Contador de loops;
5:  while critério de parada não for satisfeito do
6:      P_{g+1} ← alguns elementos de P_g;
7:      P_{g+1} ← alguns elementos de P_g tendo sofrido crossover;
8:      P_{g+1} ← alguns elementos de P_g tendo sofrido mutação;
9:      Avaliar população P_{g+1};
10:     g ← g + 1;
11: end while
12: retorne a melhor solução de P_g;
```

4.5 Algoritmos Genéticos propostos

Neste trabalho foram propostas três versões do Algoritmo Genético.

4.5.1 Algoritmo Genético Básico - AGB

Os parâmetros usados para o Algoritmo 4.9 foram:

• *Npop*: tamanho das populações P_0, P e S inicializados nas linhas 2, 5 e 6, respectivamente.

• *prob_crossover*: probabilidade de ocorrer um cruzamento entre duas soluções (linha 9).

• *prob_mutação*: probabilidade de ocorrência de mutação em uma solução (linhas 10 e 11).

• *limiteQt*: valor limite no qual metade das soluções com maiores valores de função objetivo da população P_0 são substituídas por outras, isto é, reinicializadas. Esta reinicialização ocorre quando, depois de *limiteQt* iterações

e ocorrer que nenhuma solução de P conseguir entrar em P_0 ao executar o passo 14.

Algoritmo 4.9: Pseudocódigo da versão básica do Algoritmo Genético, AGB.

```
1:  AGB(Npop, prob_crossover, prob_mutação, limiteQt):
2:  P_0 ← pop(Npop);
3:  qt ← 0; // Parâmetro para controlar a reinicialização das soluções.
4:  while critério de parada não for satisfeito do
5:     P ← ∅;
6:     S ← seleção(P_0);
7:     while P não estiver completo do
8:        X, Y ← seleciona 2 soluções de S;
9:        X', Y' ← cruzamento(X, Y, prob_crossover);
10:       X' ← mutação(X', prob_mutação);
11:       Y' ← mutação(Y', prob_mutação);
12:       P ← P U{X', Y'};
13:    end while
14:    P_0 ← melhoresDiferentes(P_0 U P);
15:    if nenhuma solução de P foi inserida em P_0 then
16:       qt ← qt + 1;
17:    else
18:       qt ← 0;
19:    end if
20:    if qt = limiteQt then
21:       P_0 ← reiniciarMetadeDasSoluções(P_0);
22:    end if
23: end while
24: s* ← melhorSolução(P_0);
25: retorne s*;
```

Na linha 2 do Algoritmo 4.9, o procedimento *pop*() gera uma nova população com o tamanho indicado pelo parâmetro *Npop*. Esta função preenche o população P_0 com soluções criadas pelo algoritmo construtivo (Algoritmo 4.3).

Na linha 6, há uma seleção de elementos para compor o conjunto S. Ela é feita pela regra de seleção por torneio ternário. Nesta regra, três indivíduos são escolhidos aleatoriamente e, então, o melhor será escolhido. Este procedimento é repetido até que o conjunto S, que tem o mesmo tamanho Npop seja todo preenchido.

Antes da operação de *crossover*, duas soluções são escolhidas do conjunto S. É importante citar que esta escolha é aleatória. Outro fato que também deve ser mostrado é que as operações de mutação e *crossover* retornarem as próprias soluções pais em caso de a probabilidade (de as operações ocorrerem) não for satisfeita.

O operador de *crossover*, ou função cruzamento, foi utilizado por Mendes et al (2002) e é bem conhecido como *Order Crossover* (OX). No algoritmo, este operador tem uma probabilidade *prob_crossover* de ser executado. Depois de escolher duas soluções para o cruzamento (X e Y), um fragmento de um cromossomo (sequência de tarefas) de uma das soluções é escolhido aleatoriamente e copiado para o filho. Numa segunda parte do método, as posições vazias são preenchidas sequencialmente de acordo com o cromossomo da outra solução. Um exemplo, para uma instância com 5 tarefas e 2 máquinas, é apresentado a seguir:

Solução pai 1 (X):
M1: 1, 1, 3, 4, 4
M2: 5, 5, 2, 2, 3

Solução pai 2 (Y):
M1: 2, 1, 3, 5, 4
M2: 1, 2, 3, 4, 5

Na primeira parte do método, um fragmento de cromossomo de X é escolhido aleatoriamente. Digamos que as tarefas nas posições 2 e 3 da máquina 1 e a tarefa na posição 1 da máquina 2 sejam escolhidas. Então o filho tem a seguinte solução parcial:

Filho 1:
M1: *, 1, 3, 4, 4
M2: 5, 5, 2, *, *

Na segunda parte, as posições vazias do filho (representadas pelos asteriscos) são preenchidas com as tarefas de Y na ordem em que elas aparecem sem que haja extrapolação no número de unidades-tarefa de cada tarefa. Portanto tem-se a solução completa do filho:
Filho 1:
M1: *2*, **1, 3, 4, 4**
M2: **5, 5, 2**, *1, 3*

As tarefas que vieram de X estão em negrito. As que vieram de Y estão em itálico. A função foi adaptada para gerar dois filhos. Para tal feito, troca-se X e Y de lugar e o processo é executado novamente gerando o segundo filho.

Na implementação da mutação, um método simples foi implementado que é baseado no movimento de inserção. Nesse método, uma tarefa é escolhida aleatoriamente de uma máquina e é inserida em uma máquina, escolhida também aleatoriamente. Se as máquinas escolhidas forem as mesmas, então o conjunto de sub-tarefas é realocada para a melhor posição (posição que resulta na menor função objetivo) dentro da máquina.

Na linha 14 do Algoritmo AGB, tem o método *melhoresDiferentes* que retorna as melhores $Npop$ soluções distintas da união de P e P_0.

Se, depois de certo número de iterações, o conjunto P_0 não mudar após a sua união com P pelo método *melhoresDiferentes*, então aquele deve alterar metade de suas soluções para haver uma maior variabilidade em seu conjunto. A questão é como as novas soluções (diferentes das que foram mantidas na lista) serão geradas. Para tal, é utilizada a heurística construtiva para substituir metade das soluções de P_0 que contem os piores valores de atraso total.

4.5.2 Algoritmo Genético com Perturbação e Busca Local - AGBL

No Algoritmo 4.10 é mostrado o pseudocódigo da primeira versão do algoritmo genético não-básico, o AGBL, que utiliza o método de busca local para melhorar as soluções previamente perturbadas. Ressalta-se que os parâmetros relacionadas ao tamanho das listas P_0, P e S, à probabilidade de *crossover* e mutação, e ao limite de iterações para reinicialização já estão com valores fixos após a calibração do Algoritmo AGB. Para manter as melhores soluções, um conjunto de soluções de boa qualidade é utilizado e, do mesmo modo como foi utilizado no Algoritmo 4.6, recebe o nome de conjunto Elite.

Porém, para este algoritmo, outros parâmetros também foram calibrados:

• $Pert$: é o nível de perturbação e está relacionado ao número de trocas de tarefas aleatórias neste método.

• Ne: este parâmetro refere-se à probabilidade de ocorrer a busca local nos elementos do conjunto $Elite$.

• $EliteSize$: é o tamanho do conjunto $Elite$.

O início do Algoritmo 4.10 é similar ao do Algoritmo 4.9. Na linha 24, o método *seleçãoAleatóriaGulosa* é aplicado escolhendo aleatoriamente $EliteSize$ soluções das $CandEliteSize$ melhores do conjunto P_0. Na linha 25 o método da Perturbação envolve trocar aleatoriamente $Pert$ tarefas de cada solução do conjunto $Elite$ a fim de diversificar um pouco mais as soluções.

Algoritmo 4.10: Pseudocódigo do AG com Busca Local, AGBL.

```
1: AGBL(Npop, prob_crossover, prob_mutação,
        limiteQt, Pert, Ne, EliteSize, CandEliteSize):
2: P_0 ← pop();
3: Elite ← melhoresDiferentes(P_0);
4: qt ← 0; // Contador para controlar a reinicialização das soluções.
5: while critério de parada não for satisfeito do
6:    P ← ∅;
7:    S ← seleção(P_0);
8:    while P não estiver completo do
9:       X, Y ← seleciona 2 soluções de S;
10:      X', Y' ← cruzamento(X, Y, prob_crossover);
11:      X' ← mutação(X', prob_mutação);
12:      Y' ← mutação(Y', prob_mutação);
13:      P ← P U{X', Y'};
14:   end while
15:   P_0 ← melhoresDiferentes(P_0 U P);
16:   if nenhuma solução de P foi inserida em P_0 then
17:      qt ← qt + 1;
18:   else
19:      qt ← 0;
20:   end if
21:   if qt = limiteQt then
22:      P_0 ← reiniciarMetadeDasSoluções(P_0);
23:   end if
24:   Elite ← seleçãoAleatóriaGulosa(P_0, EliteSize, CandEliteSize);
25:   Elite ← Perturbação(Elite, Pert);
26:   Elite ← buscaLocal(Elite, Ne);
27:   P_0 ← melhoresDiferentes(P_0 U Elite);
28: end while
29: s* ← melhorSolução(P_0);
30: retorne s*;
```

Este método (perturbação) é executado em cada geração. Por outro lado, para cada solução do conjunto *Elite*, a Busca local tem uma chance de $Ne\%$ de ser executada.

4.5.3 Algoritmo Genético com *Path-Relinking* - AGPR

O Algoritmo 4.11 mostra o pseudocódigo da segunda versão do algoritmo genético não-básico, o AGPR, que utiliza o método de *Path-Relinking*. Para este algoritmo, outro parâmetro também foi calibrado: $EliteSize$ que é o tamanho do conjunto *Elite* para o AGPR.

O método *PathRelinkingEntreCadaPar* aplica o método do *Path-Relinking* da subseção 4.2.3 entre cada par de solução do conjunto Elite.

Algoritmo 4.11: Pseudocódigo da versão com Path Relinking do Algoritmo Genético, AGPR.

```
1: AGPR(Npop, prob_crossover, prob_mutação,
        EliteSize, CandEliteSize):
2: P_0 ← pop();
3: Elite ← melhoresDiferentes(P_0);
4: qt ← 0; # Variável para controlar a reinicialização das soluções.
5: while critério de parada não for satisfeito do
6:    P ← ∅;
7:    S ← seleção(P_0);
8:    while P não estiver completo do
9:       X, Y ← seleciona 2 soluções de S;
10:      X', Y' ← cruzamento(X, Y, prob_crossover);
11:      X' ← mutação(X', prob_mutação);
12:      Y' ← mutação(Y', prob_mutação);
13:      P ← P U{X', Y'};
14:   end while
15:   P_0 ← melhoresDiferentes(P_0 U P);
16:   if nenhuma solução de P foi inserida em P_0 then
17:      qt ← qt + 1;
18:   else
19:      qt ← 0;
20:   end if
21:   if qt = limiteQt then
22:      P_0 ← reiniciarMetadeDasSoluções(P_0);
23:   end if
24:   Elite ← seleçãoAleatóriaGulosa(P_0, EliteSize, CandEliteSize);
25:   Elite ← PathRelinkingEntreCadaPar(Elite);
26:   P_0 ← melhoresDiferentes(P_0 U Elite);
27: end while
28: s* ← melhorSolução(P_0);
29: retorne s*;
```

5 Calibração de parâmetros das heurísticas

Neste capítulo são analisados os valores dos parâmetros utilizados nas heurísticas GRASP Reativo e Algoritmo Genético que foram apresentados no capítulo 4. Apresenta-se, também, a forma de geração dos problemas a serem utilizados nos testes.

No Algoritmo GRASP Reativo, são testados valores para os parâmetros γ e θ. Analisam-se, também, dois tipos de Busca Local utilizadas por este algoritmo.

No Algoritmo Genético, são analisados os parâmetros tamanho da população, probabilidade de *crossover*, probabilidade de mutação e tempo limite para reinicialização da população ($limiteQT$). Também foram testadas versões híbridas deste algoritmo que utilizam Busca Local e *Path-Relinking*. Os parâmetros utilizados na Busca Local e no *Path-Relinking* também são calibrados.

Para analisar o desempenho das diferentes versoes das heurísticas propostas, ou seja, utilizando diferentes parâmetros e estratégia, utiliza-se o Desvio Percentual Relativo (RPD, em inglês: *Relative Percent Deviation*). O RPD é calculado da seguinte maneira:

$$RPD(\%) = 100 \times \frac{(f_{algoritmo} - f_{melhor})}{f_{melhor}}$$

O valor de $f_{algoritmo}$ é o valor da função objetivo do algoritmo do qual se quer calcular o RPD, enquanto f_{melhor} é o melhor valor da função objetivo obtido entre todos os algoritmos considerados.

5.1 Geração dos problemas

Para fazer os testes computacionais das heurísticas, são utilizados 3 conjuntos de problemas: pequeno, médio e grande porte. A dimensão de um problema é dada pela quantidade de tarefas, máquinas e unidades-tarefa. O tamanho de um problema é denotado por [n, m, u], sendo n, m e u, respectivamente, o número de tarefas, o número de máquinas e o número de unidades-tarefa. Os problemas de pequeno porte possuem as seguintes dimensões: [2, 2, 5], [3, 2, 5], [3, 3, 5], [4, 2, 5] e [4, 3, 5]. Os problemas de médio porte possuem as seguintes dimensões: [10, 4, 4], [15, 4, 6] e [15, 8, 6]. As dimensões dos problemas de grande porte são: [20, 8, 8], [20, 12, 8], [25, 12, 10], [30, 15, 10), [40, 20, 12] e [50, 20, 12].

Para cada problema de tamanho [n, m, u] são gerados três tipos de tempos de preparação gerados aleatoriamente: o *short*, *middle* e o *long*, do mesmo modo como em (Sariçiçek e Çelik, 2011). O tempo de processamento de

cada unidade-tarefa é inteiro e gerado aleatoriamente de uma distribuição uniforme em um escopo de (5, 60). O tempo de preparação para uma tarefa é gerado aleatoriamente de uma distribuição uniforme com escopo (5, 60) para o tipo *short*, (6, 120) para o tipo *middle* e (120, 180) para o tipo *long*. Assim como em Shim e Kim (2008), as datas de entrega de uma tarefa é gerada de uma distribuição uniforme de escopo $[\alpha \sum(s_j + u_j p_j)/m, \beta \sum(s_j + u_j p_j)/m]$, em que s_j, p_j, u_j e m denotam, respectivamente, o tempo de preparação da tarefa j, o tempo de processamento da tarefa j, o número de unidades-tarefa da tarefa j e o número de máquinas, e α e β são parâmetros usados para controlar o escopo das datas de entrega. Assim como em Sariçiçek e Çelik (2011), neste trabalho, os valores de α e β foram 0 e 1,2.

Foram gerados um total de 210 problemas, sendo que 75, 45 e 90 problemas de pequeno, médio e grande porte, respectivamente. Para cada combinação do tamanho [n, m, u] de problema e tipo de tempo de preparação, foram gerados 5 instâncias.

Para fazer a calibração das heurísticas foram utilizadas 42 instâncias, sendo uma instância de cada combinação do tamanho [n, m, u] e tipo de tempo de preparação. No capítulo 6, as melhores versões das heurísticas são comparadas utilizando todas as 210 instâncias.

5.2 Calibração dos parâmetros da heurística GRASP Reativo

5.2.1 Calibração da heurística GRASP Reativo com Busca Local 1

Esta busca local foi calibrada no GRASP Reativo. Os valores dos parâmetros do mecanismo Reativo (γ e θ) foram calibrados juntos aos valores do parâmetro da busca local (*prob*). Os valores testados para cada parâmetro são mostrados a seguir:

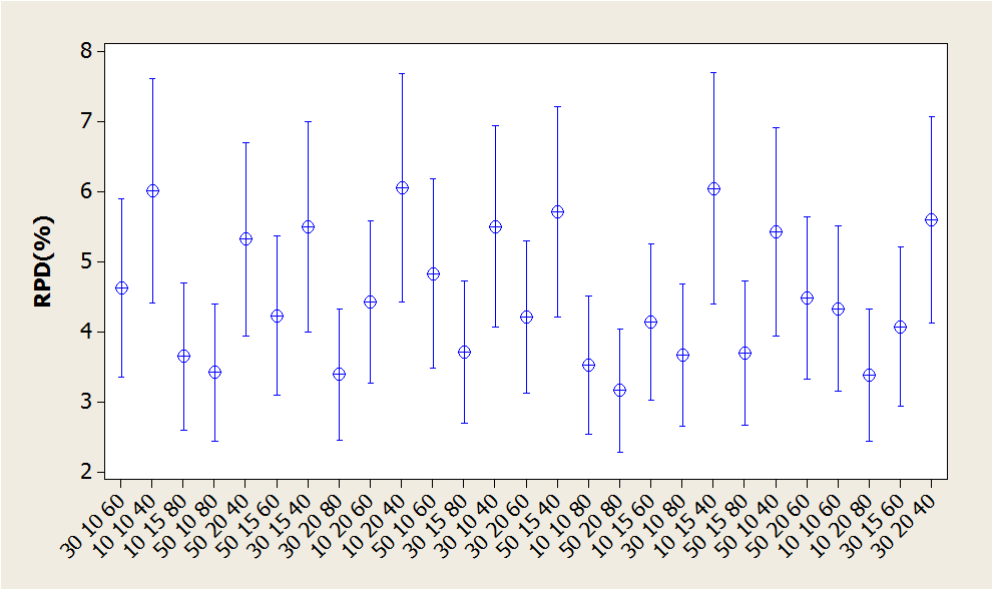
- γ: 10, 30 e 50.
- θ: 10, 15 e 20.
- *prob*(%): 40, 60 e 80.

Na Figura 5.1 são mostrados os intervalos de Tukey HSD (*Honestly Significant Difference*) e as médias do RPD (95% de confiança) para as diferentes combinações dos parâmetros γ, θ e *prob*. Note que intervalos com sobreposição denotam diferenças insignificantes estatisticamente entre as combinações destes intervalos sobrepostos (Pan e Ruiz, 2012).

De acordo com a Figura 5.1, não houve uma combinação que pode ser dita como melhor do que todas as outras, estatisticamente. Porém, a combinação

que gerou o melhor RPD médio, com o menor intervalo, foi a combinação $\gamma = 50, \theta = 20, prob = 80$.

Figura 5.1: Intervalos de Tukey HSD e as médias do RPD (95% de confiança) para o GRASP com a Busca Local 1.



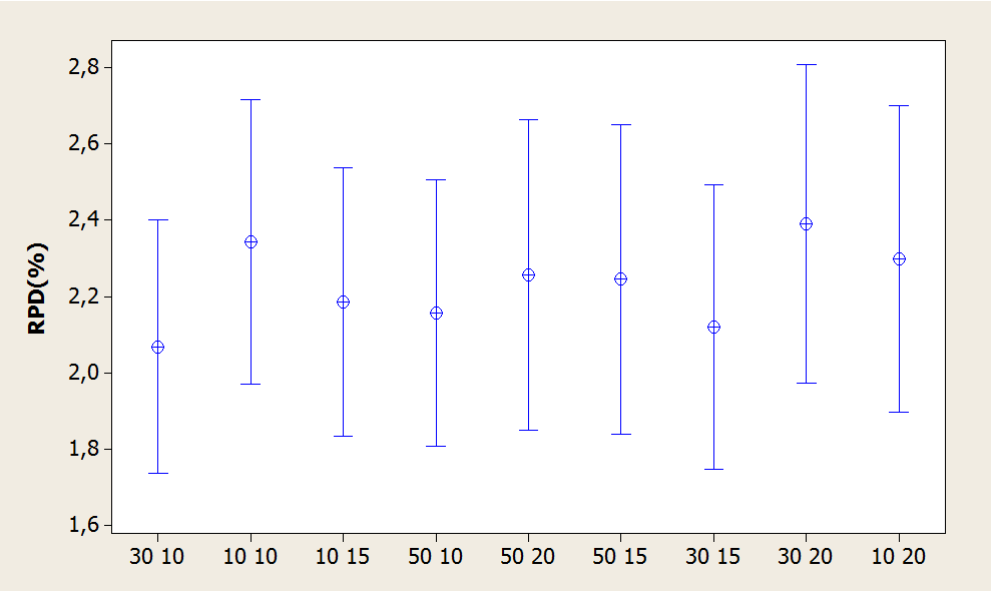
5.2.2 Calibração da heurística GRASP Reativo com Busca Local 2

A BL2 não tem parâmetro para ser calibrado. Porém, a heurística construtiva foi calibrada no GRASP Reativo e os valores testados para cada parâmetro são mostrados a seguir:

- γ: 10, 30 e 50.
- θ: 10, 15 e 20.

Na Figura 5.2 são mostrados os intervalos de Tukey HSD (*Honestly Significant Difference*) e as médias do RPD (95% de confiança) para as diferentes combinações dos parâmetros γ, θ. Não houve uma combinação que pode ser dita como melhor do que todas as outras, estatisticamente falando. A combinação que gerou o melhor RPD médio, com um intervalo pequeno, foi a combinação $\gamma = 30, \theta = 10$ e esta foi a combinação escolhida.

Figura 5.2: Intervalos de Tukey HSD e as médias do RPD (95% de confiança) para o GRASP com a Busca Local 2.



5.2.3 GRASP Reativo com *Path-Relinking*

Os valores testados para o tamanho do conjunto Elite foram 6, 8, 10, 12 e 15. Nesta calibração, os testes já foram feitos considerando a Busca Local 1 pelo fato de ela ter apresentado valores melhores do que a Busca Local 2. Como pode-se ver na Figura 5.3, não houve uma combinação estatisticamente melhor do que as outras, e os resultados ficaram bem próximos, porém, em média, o valor de 15 foi ligeiramente melhor (e com pequeno intervalo) e foi escolhido.

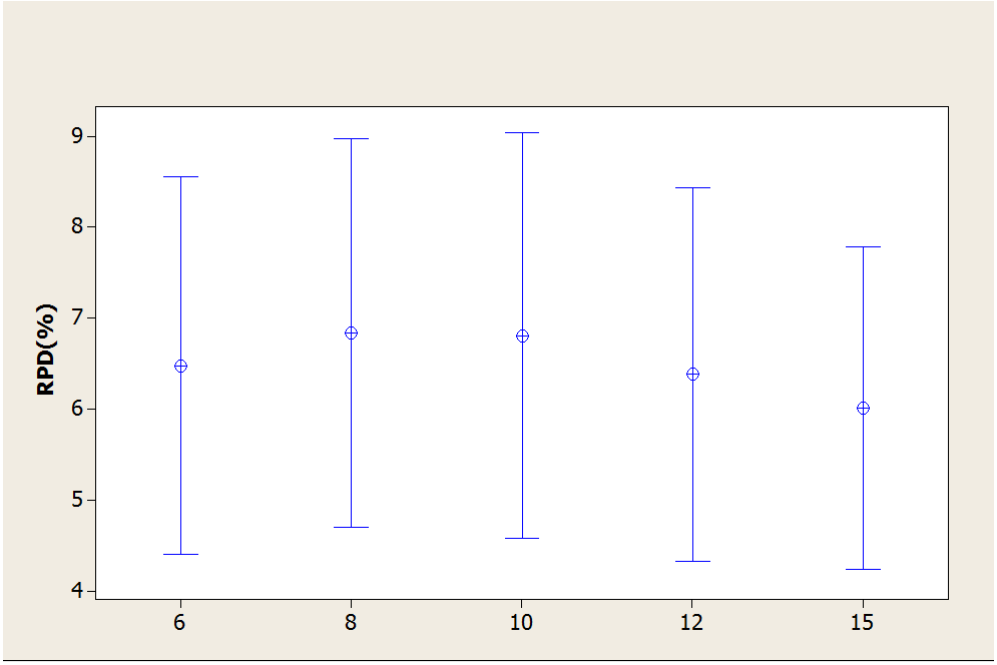
5.3 Calibração dos parâmetros do Algoritmo Genético

5.3.1 Algoritmo Genético Básico - AGB

Os parâmetros do Algoritmo Genético foram calibrados e os valores para cada parâmetro são mostrados a seguir:

- *Npop*: 50 e 80.
- *prob_crossover*: 50%, 70% e 90%.
- *prob_mutação*: 10%, 30% e 50%.
- *limiteQt*: 5 e 8.

Figura 5.3: Intervalos de Tukey HSD e as médias do RPD (95% de confiança) para o tamanho do conjunto Elite.



Todas as combinações de $Npop, prob\_crossover$, *prob_mutação, limiteQt* foram testadas para o Algoritmo Genético e os intervalos de Tukey HSD e as médias do RPD são mostrados na Figura 5.4. Não houve uma combinação que pode ser dita como melhor do que todas as outras, estatisticamente falando. A combinação que gerou o melhor RPD médio e com um intervalo pequeno foi a combinação $Npop = 80, prob\_crossover = 70\%$, *prob_mutação = 10%, limiteQt = 8* e foi escolhida.

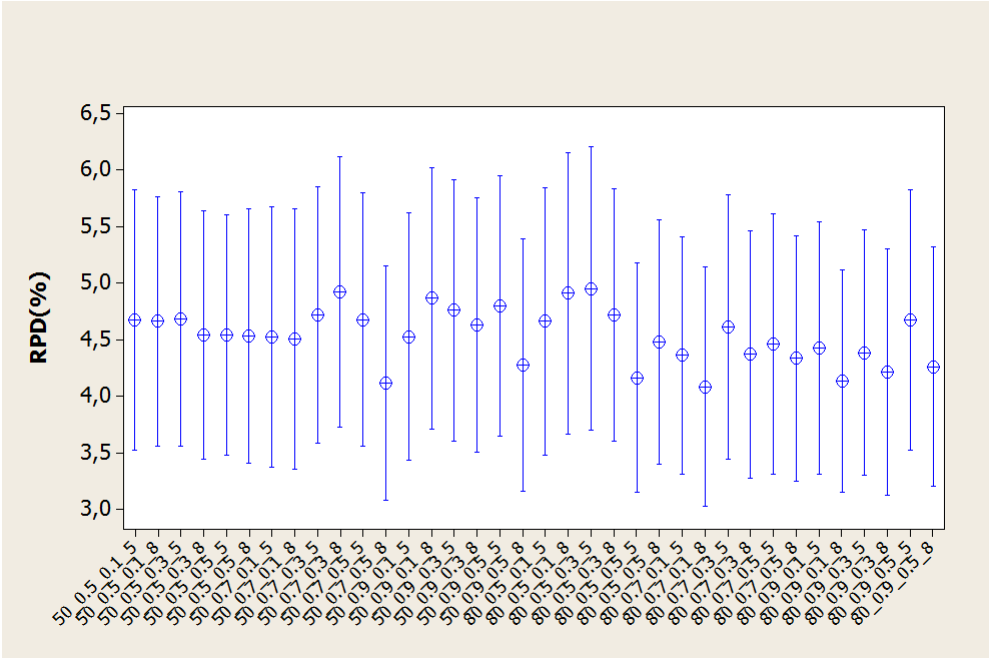
5.3.2 Algoritmo Genético com Perturbação e Busca Local - AGBL

Os parâmetros do AGBL foram calibrados e os valores para cada parâmetro são mostrados a seguir:

- Ne: 2, 4, 6 e 10.
- $EliteSize$: 20, 50 e 100.
- $Pert$: 4, 7 e 10.

Todas as combinações de $Pert, Ne, EliteSize$ foram testadas para o AGBL

Figura 5.4: Intervalos de Tukey HSD e as médias do RPD (95% de confiança) para o Algoritmo Genético Básico.



e os intervalos de Tukey HSD e as médias do RPD são mostrados na Figura 5.5 criada. Não houve uma combinação que pode ser dita como melhor do que todas as outras, estatisticamente falando. A combinação que gerou o melhor RPD médio, e com pequeno intervalo, foi a combinação $Pert = 10, Ne = 6 e EliteSize = 20$ e foi escolhida.

5.3.3 Algoritmo Genético com *Path-Relinking* - AGPR

Os parâmetros do AGPR foram calibrados e os valores para o parâmetro do tamanho conjunto Elite foram 4, 8, 12, 16. O valor escolhido foi 4 devido ao seu menor valor médio de RPD.e um intervalo pequeno. Os intervalos de Tukey HSD e as médias do RPD, relativo aos valores do tamanho do conjunto Elite, estão na Figura 5.6.

Figura 5.5: Intervalos de Tukey HSD e as médias do RPD (95% de confiança) para o AGBL.

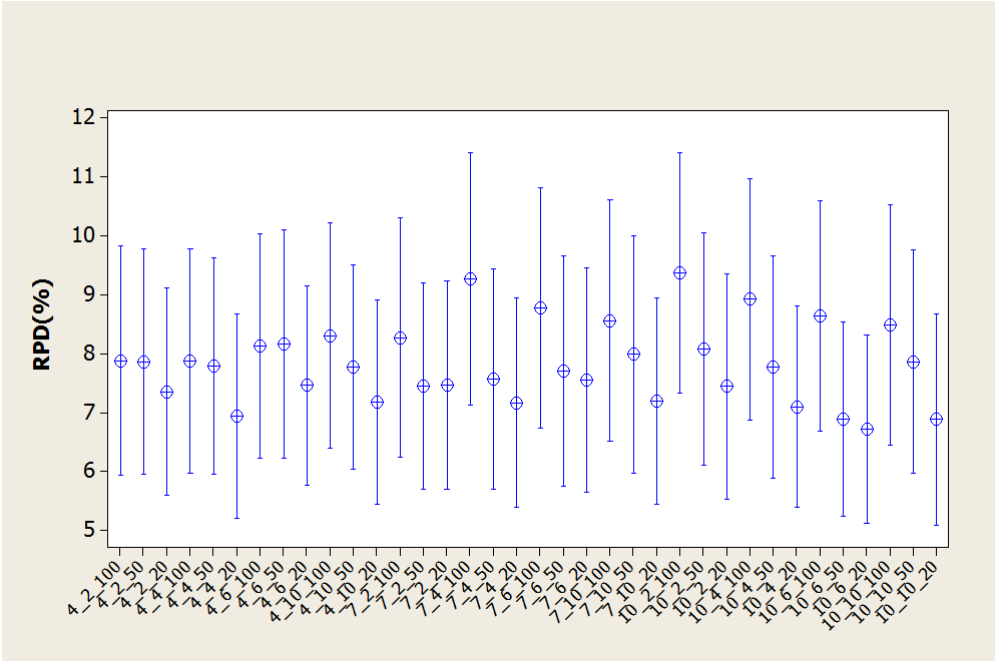
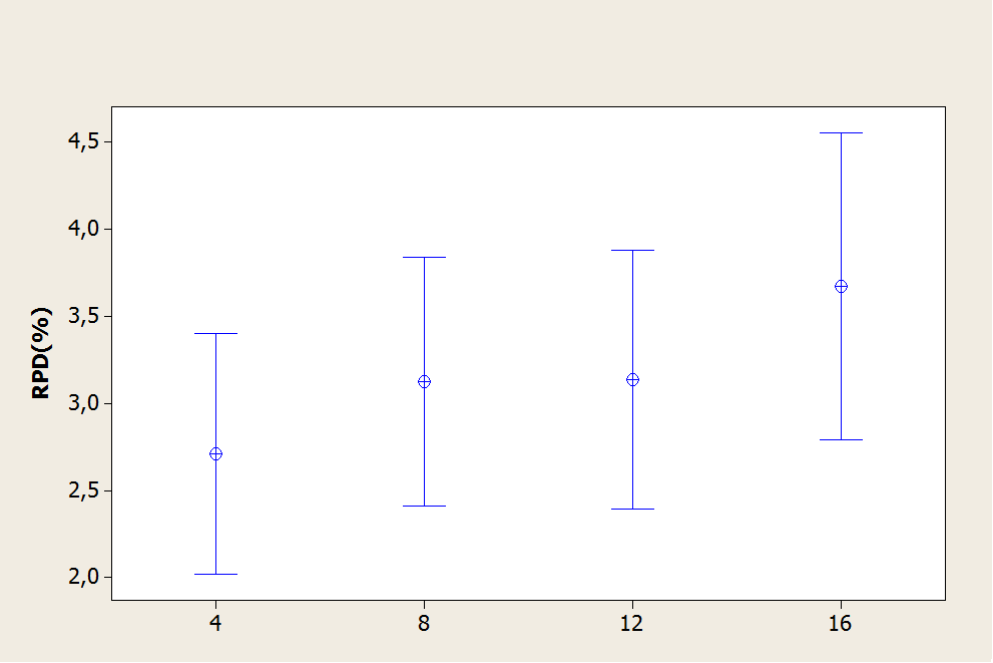


Figura 5.6: Intervalos de Tukey HSD e as médias do RPD (95% de confiança) para o tamanho do conjunto Elite do AGPR.



6 Comparação das Heurísticas GR, GPR, AGB, AGPR, SA e TS

Neste capítulo, serão apresentadas as comparações das melhores versões das heurísticas GRASP e Algoritmo Genético, ou seja, são analisadas o desempenho das heurísticas GR, GPR, AGB e AGPR. Também estas heurísticas são comparadas com as heurísticas da literatura SA e TS (Sariçiçek e Çelik, 2011).

Além dos resultados mostrados neste capítulo, no Apêndice A são apresentados os valores absolutos médios (média de 5 rodadas) da função objetivo obtidos pelos algoritmos comparados para todas as 210 instâncias testadas. No Apêndice B são apresentados os melhores valores da função objetivo encontrados para todas instâncias.

Todas as heurísticas foram executadas em um computador de processador Intel Core i7 3.07GHz com 6Gb de RAM usando Windows 7. O código foi compilado usando a linguagem de programação C++ em um compilador g++ 3.4.2 (mingw-special).

Para cada instância, são criados, também, cinco diferentes tipos de problemas e, para cada problema, são executados os algoritmos cinco vezes, isto é, há 14 tipos de problemas no total (cinco para o pequeno, três para o médio e seis para o grande), 3 classes de tempos de preparação e 5 problemas para cada problema, totalizando 210 problemas (cada um sendo executado cinco vezes, como dito no capítulo 5). Então são comparadas as heurísticas usando o RPD. No capítulo 5, a fórmula do RPD foi explicada, mas neste capítulo, é apresentado um exemplo para ficar mais claro como funciona o RPD. Considere a Tabela 6.1 onde tem valores de funções objetivos de 3 algoritmos hipotéticos (alg1, alg2 e alg3). Para cada problema, há um algoritmo que chegou na melhor solução. O valor da função objetivo deste algoritmo está em negrito na Tabela indicando que este valor é o f_{melhor}.

Problema	$f_{algoritmo1}$	$f_{algoritmo2}$	$f_{algoritmo3}$	f_{melhor}
1	**20**	35	23	20
2	**20**	25	24	20
3	30	35	**27**	27
4	**22**	25	36	22
5	**23**	25	26	23

Tabela 6.1: Valores hipotéticos de função objetivo de 3 algoritmos para 5 problemas

$Problema$	$RPD_{algoritmo1}(\%)$	$RPD_{algoritmo2}(\%)$	$f_{algoritmo3}(\%)$
1	0	75	15
2	0	25	20
3	11,11	29,63	0
4	0	13,64	63,64
5	0	8,7	13,04

Tabela 6.2: Valores do RPD com relação à Tabela 6.1.

Na Tabela 6.2, há o valor do RPD para cada problema da Tabela 6.1, para uma melhor visualização, calculado utilizando a fórmula mostrada no capítulo 5. Pode acontecer de cada problema ser executado várias vezes para se ter várias funções objetivo de cada problema (neste trabalho, cada problema foi executado 5 vezes). Neste caso, para calcular o RPD, foram utilizados dois métodos: o primeiro método de comparação considera a média dos RPD de todas as rodadas do problema como sendo o $f_{algoritmo}$; o segundo método de comparação considera o melhor dos RPD de todas as rodadas do problema como sendo o $f_{algoritmo}$. O modo de calcular o valor de f_{melhor} é o mesmo para ambos os métodos que é escolher o melhor valor de RPD de todos os algoritmos considerando todas as rodadas do problema.

Neste trabalho, o critério de parada baseou-se num tempo limite de execução, e esse tempo limite é definido de acordo com o tamanho da instância do problema.

Definiu-se que as medidas que determinam o tamanho de uma instância do problema são o número de tarefas n, o número de máquinas m e o número de unidades-tarefa u de cada tarefa. A fórmula usada neste trabalho para definir o tempo de execução que cada algoritmo utilizará para resolver um problema foi: $n \times m \times u \times 0,015$ segundos.

Na Tabela 6.3, os valores para n, m, u e t são, respectivamente, o número de tarefas, máquinas, unidades-tarefa e o tempo que cada problema será resolvido.

Para realizar os testes, dois modos de comparação dos algoritmos usando o RPD são feitos. No primeiro modo de comparação, o RPD, para cada problema, foi feito comparando o melhor valor da função objetivo, tendo como parâmetro todos os algoritmos, com a média das 5 rodadas de cada algoritmo executado para cada problema. Isto é, para cada problema, o melhor valor de função objetivo é selecionado considerando todas as 5 rodadas dos 6 algoritmos. Após isso, é feito uma média das 5 rodadas de cada algoritmo e o RPD é calculado comparando o melhor valor que foi selecionado com a média de cada algoritmo. No segundo modo de comparação, ao calcular o

n	m	u	t(s)
2	2	5	0,3
3	2	5	0,4
3	3	5	0,7
4	2	5	0,6
4	3	5	0,9
10	4	4	2,4
15	4	6	5,4
15	8	6	10,8
20	8	8	19,2
20	12	8	28,8
25	12	10	45,0
30	15	10	67,5
40	20	12	144,0
50	20	12	180,0

Tabela 6.3: Tempo t requerido (em segundos) para os problemas-teste do grupo pequeno, médio e grande.

RPD, ao invés de utilizar a média das 5 rodadas, utilizou-se comparar com o melhor valor de função objetivo das 5 rodadas.

			Média dos RPD%					
n	m	u	GR	GPR	AGB	$AGPR$	SA	TS
2	2	5	0,00	0,24	0,25	0,25	3,33	3,28
3	2	5	0,17	0,19	2,04	0,19	5,88	4,13
3	3	5	0,48	0,72	10,08	2,32	8,43	8,78
4	2	5	0,06	0,00	4,46	0,00	5,98	6,83
4	3	5	1,48	0,22	11,13	0,75	14,90	12,11
10	4	4	12,32	3,49	10,10	2,33	12,13	12,71
15	4	6	20,83	7,75	13,44	6,17	10,04	10,99
15	8	6	24,98	10,09	13,34	6,45	21,56	21,00
20	8	8	33,49	13,27	20,75	11,75	15,47	17,65
20	12	8	30,30	15,47	15,39	9,48	19,51	20,53
25	12	10	33,36	13,96	18,58	9,23	17,54	69,96
30	15	10	38,70	20,96	21,65	11,75	21,84	31,10
40	20	12	34,03	17,50	22,92	6,82	57,59	58,90
50	20	12	36,16	17,18	33,52	6,70	73,23	71,07
		Média	19,03	8,65	14,12	5,30	20,53	24,93

Tabela 6.4: Valores das médias do RPD dos algoritmos.

Numa primeira parte, serão explicados os resultados somente do primeiro método de comparação para o cálculo do RPD, isto é, para cada problema, é considerada, no cálculo do RPD, a média das 5 rodadas de cada algoritmo. Os valores são mostrados na tabela 6.4. Um ponto interessante a ser notado são os valores de RPD para os problemas de pequeno porte obtidos pelo Algoritmo GR e GPR que são bem similares. À medida que o tamanho do problema aumenta, esta diferença aumenta e o GPR se sobressai em relação ao GR. Este fato não acontece quando se compara os Algoritmos AGB e AGPR. Neste caso, o Algoritmo AGPR se sobressai desde os problemas de pequeno porte até os problemas de grande porte. Outro fato a se notar são os altos valores dos Algoritmos SA e TS, em relação aos demais Algoritmos.

Mais adiante, serão explicados os resultados do segundo método de comparação que utilizou, ao invés da média das 5 rodadas, o melhor valor das 5 rodadas.

A fim de verificar qual algoritmo teve melhor desempenho (discutido no capítulo 6) e verificar se existe diferença estatística significativa entre as soluções obtidas pelos algoritmos procedeu-se uma ANOVA simples. Na ANOVA são feitas três suposições principais a respeito do conjunto de dados (Gonçalves e dos Santos, 2008):

• Cada observação é independente das demais;

• A variável dependente deve ser normalmente distribuída nos tratamentos;

• As variâncias dos diferentes tratamentos devem ser iguais.

Nos testes feitos, os dados não obedeceram a suposição de normalidade. Todos os testes da ANOVA foram baseados sobre os resíduos da variável resposta, neste caso, o RPD. Os valores dos resíduos são calculados através da fórmula: $(x_i - \bar{x})/\sigma$, onde x_i é o valor do i-ésimo RPD, $\bar{x}$ é a média dos valores do RPD e σ é o desvio padrão.

Já que uma das suposições da ANOVA não foi satisfeita, existe uma ferramenta estat'istica não-paramétrica para poder verificar qual algoritmo teve melhor desempenho. Para avaliar os resultados obtidos pelos algoritmos, o teste da Mediana de *Mood* (Montgomery et al, 2001) foi aplicado para verificar se as diferenças observadas são estatisticamente significantes. O teste da Mediana de *Mood* é uma alternativa robusta não-paramétrica que não requer dados distribuidos normalmente. O teste da Mediana de *Mood* foi realizado empregando as medidas do RPD como variável resposta e foi realizado com um nível de confiança de 95%. A Tabela 6.5 mostra os resultados obtidos por este teste. Em cada linha da Tabela 6.5, tem-se o número de amostras de cada Algoritmo sendo que, neste caso, este valor é 210, totalizando 1260 amostras. Além disso, as colunas três e quatro de cada linha mostram dois grupos, quais sejam, a quantidade de valores menores ou iguais e maiores, respectivamente, à mediana amostral que, neste caso, tem valor 10,81. A suposição, ou hipótese nula, é que, se não há diferença de mediana entre estes grupos, a porcentagem de valores menores ou iguais e maiores do que a mediana amostral será igual para cada grupo. Como o valor-P $= 0 \leq$ risco alpha de 5% (0,05), a hipótese nula (hipótese que os algoritmos são estatisticamente iguais) é rejeitada e pelo menos uma diferença significante pode ser assumida (hipótese alternativa), isto é, há diferenças estatísticas significativas entre os algoritmos.

Total x = 1260
Mediana amostral = 10,81

Algoritmo	Tamanho da amostra	x $\leq$	x >	Mediana
AGB	210	93	117	12
AGPR	210	173	37	2,31
GR	210	83	127	17,95
GPR	210	134	76	6,34
SA	210	77	133	14,31
TS	210	70	140	15,79

Valor-P = 0

Tabela 6.5: Resultado do teste não-paramétrico da Mediana de *Mood*.

O teste da Mediana de *Mood* só mostra que há diferença significativa entre os algoritmos, mas não informa qual algoritmo obteve melhor desempenho. Para isso, foi usado o teste de *Multiple Range* e Tukey HSD (*Honestly Significant Difference*) para comparar todas as médias, com um nível de significância de 95%. A Tabela 6.6 mostra estas comparações. Vale informar que o *, em algumas células da coluna "Sig.", significa diferença entre os algoritmos analisados. Outra observação é: na coluna "Diferença", o valor positivo significa que o segundo algoritmo do par de algoritmos em comparação é melhor estatisticamente do que o primeiro (obviamente só se houver um * na coluna de "Sig."). O valor negativo indica que o primeiro algoritmo em questão é melhor estatisticamente do que o segundo.

Para verificar qual algoritmo foi melhor do que o outro, foi utilizado o gráfico de medianas com intervalo de confiança de 95% como mostrado na Figura 6.1. Pela Figura 6.1 e pela Tabela 6.6, nota-se que não há diferença estatística significativa entre os algoritmos AGPR e GPR, GR e SA, SA e TS. Entre todos os outros pares, há diferença estatística significativa. Nota-se um melhor desempenho dos algoritmos AGPR e GPR em relação aos demais.

Par de algoritmo	Sig.	Diferença	+/- Limites
AGB - AGPR	*	8,81919	4,8629
AGB - GR	*	-4,90948	4,8629
AGB - GPR	*	5,47162	4,8629
AGB - SA	*	-6,41224	4,8629
AGB - TS	*	-10,8156	4,8629
AGPR - GR	*	-13,7287	4,8629
AGPR - GPR		-3,34757	4,8629
AGPR - SA	*	-15,2314	4,8629
AGPR - TS	*	-19,6348	4,8629
GR - GPR	*	10,3811	4,8629
GR - SA		-1,50276	4,8629
GR - TS	*	-5,90614	4,8629
GPR - SA	*	-11,8839	4,8629
GPR - TS	*	-16,2872	4,8629
SA - TS		-4,40338	4,8629

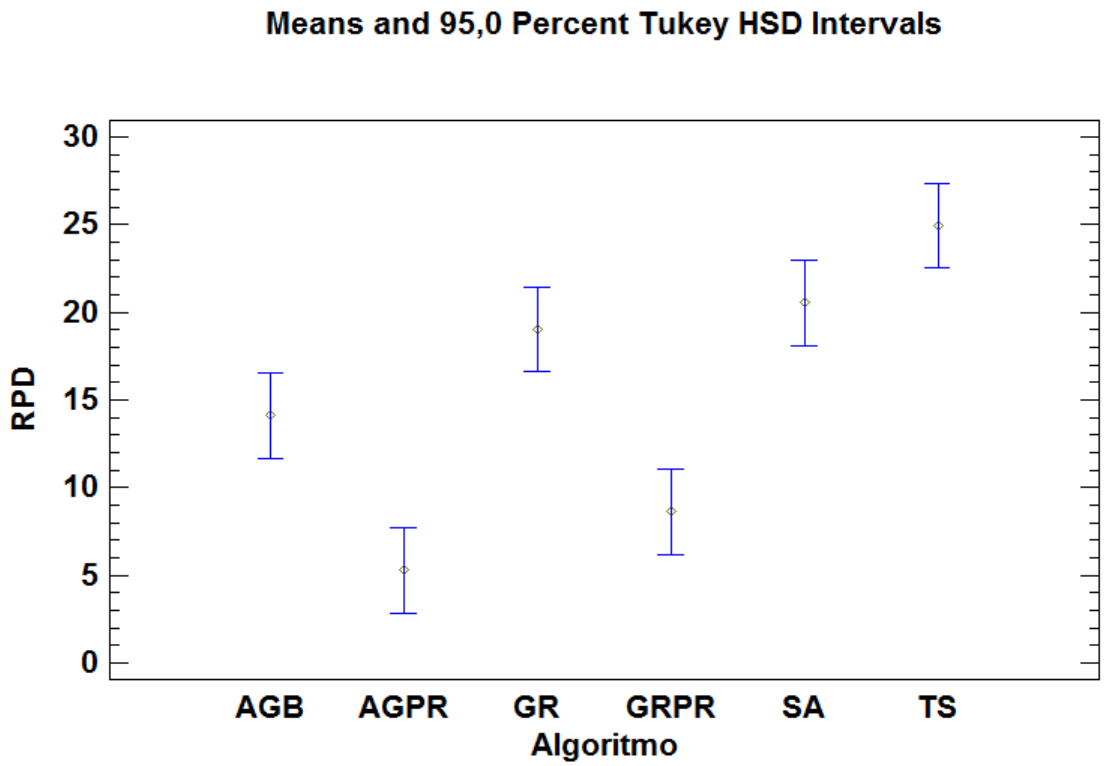
Tabela 6.6: Resultado do teste de *Multiple Range* e Tukey HSD. O * denota diferença significante entre o par de algoritmo em questão.

Nos testes feitos para este segundo método, os dados não obedeceram a suposição de normalidade. Para avaliar os resultados obtidos pelos algoritmos, o teste da Mediana de *Mood* (Montgomery et al, 2001) foi aplicado para verificar se as diferenças observadas são estatisticamente significantes. O teste da Mediana de *Mood* foi realizado empregando as medidas do RPD como variável resposta e foi realizado com um nível de confiança de 95%. A Tabela 6.7 mostra os resultados obtidos por este teste de forma similar á tabela 6.5. Como o valor-P $= 0 \leq$ risco alpha de 5% (0,05), a hipótese nula (hipótese que os algoritmos são estatisticamente iguais) é rejeitada e pelo menos uma diferença significante pode ser assumida (isto é, hipótese alternativa), isto é, há diferenças estatísticas significativas entre os algoritmos.

Para mostrar onde estão as diferenças, foi usado o teste de *Multiple Range* e Tukey HSD (*Honestly Significant Difference*) para comparar todas as médias, com um nível de significância de 95%. A Tabela 6.8 mostra estas comparações.

Para verificar qual algoritmo foi melhor do que o outro, foi utilizado o gráfico de medianas com intervalo de confiança de 95% como mostrado na Figura 6.2. Pela Figura 6.2 e pela Tabela 6.8, nota-se que não há diferença estatística significativa entre os algoritmos AGPR e GPR, AGB e TS, AGB e SA, SA e TS. Entre todos os outros pares, há diferença estatística signifi-

Figura 6.1: Intervalo de médias e Tukey HSD 95% para o primeiro método de comparação.



Total x = 1260
Mediana amostral = 10,81

Algoritmo	Tamanho da amostra	x ≤	x >	Mediana
AGB	210	77	133	5,1
AGPR	210	146	64	0
GR	210	72	138	14,79
GPR	210	137	73	0
SA	210	102	108	2,96
TS	210	96	114	3,41
Valor-P = 0				

Tabela 6.7: Resultado do teste não-paramétrico da Mediana de *Mood*.

cativa. Assim como no método de comparação anterior, nota-se um melhor

desempenho dos algoritmos AGPR e GPR em relação aos demais.

Par de algoritmo	Sig.	Diferença	+/- Limites
AGB - AGPR	*	5,32257	4,04948
AGB - GR	*	-8,18424	4,04948
AGB - GPR	*	4,16362	4,04948
AGB - SA		-1,49124	4,04948
AGB - TS		-3,16629	4,04948
AGPR - GR	*	-13,5068	4,04948
AGPR - GPR		-1,15895	4,04948
AGPR - SA	*	-6,81381	4,04948
AGPR - TS	*	-8,48886	4,04948
GR - GPR	*	12,3479	4,04948
GR - SA	*	6,693	4,04948
GR - TS	*	5,01795	4,04948
GPR - SA	*	-5,65486	4,04948
GPR - TS	*	-7,3299	4,04948
SA - TS		-1,67505	4,04948

Tabela 6.8: Resultado do teste de *Multiple Range* e Tukey HSD. O * denota diferença significante entre o par de algoritmo em questão.

Figura 6.2: Intervalo de médias e Tukey HSD 95% para o segundo método de comparação.

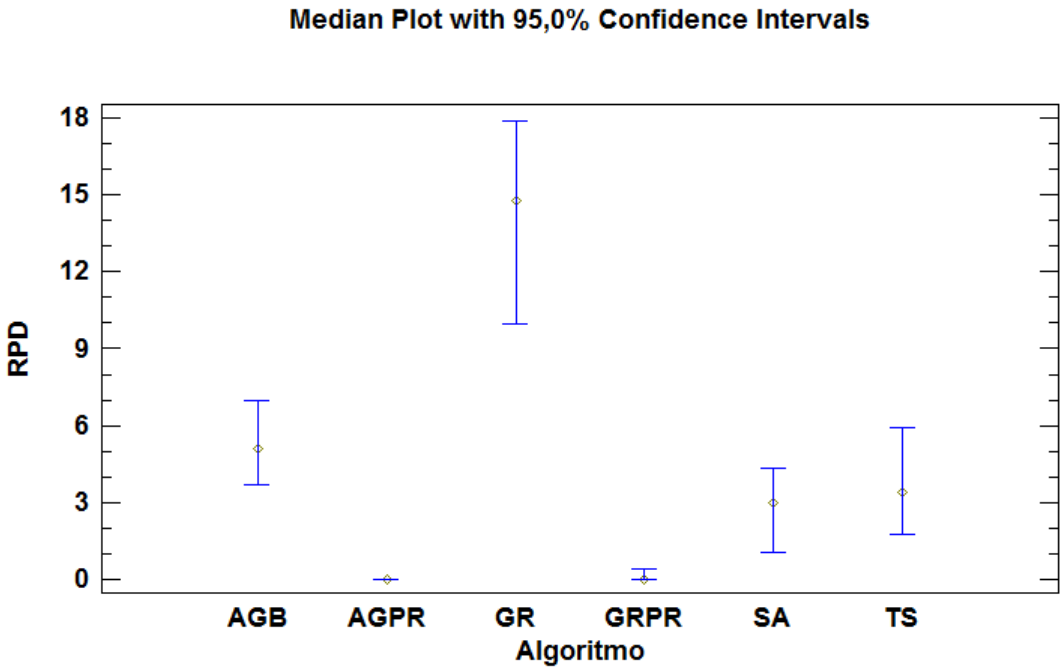
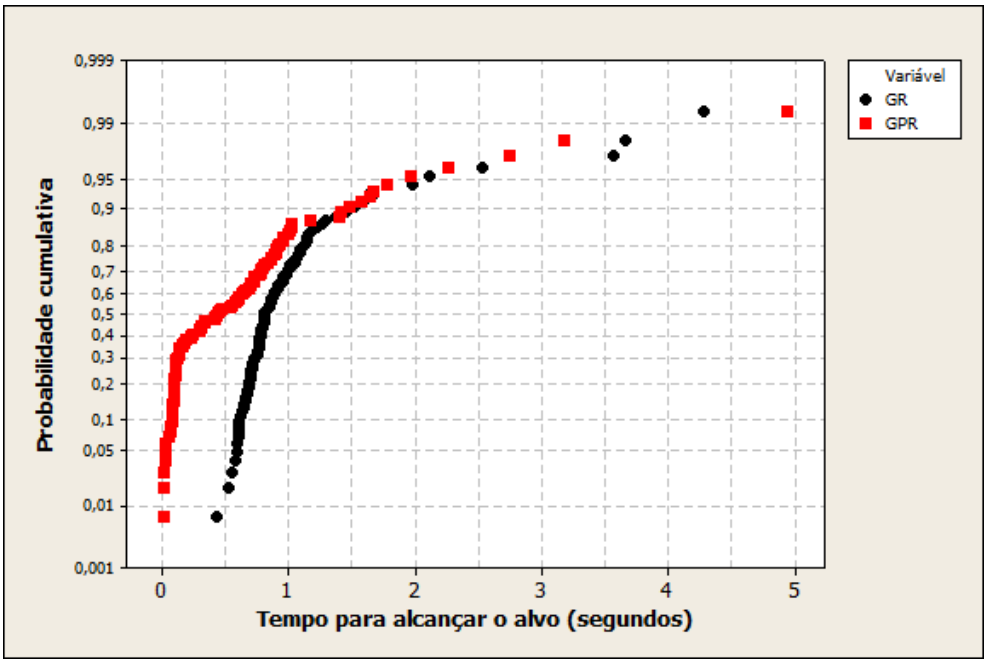


Figura 6.3: Teste de probabilidade empírica para comparar os algoritmos GR e GPR.



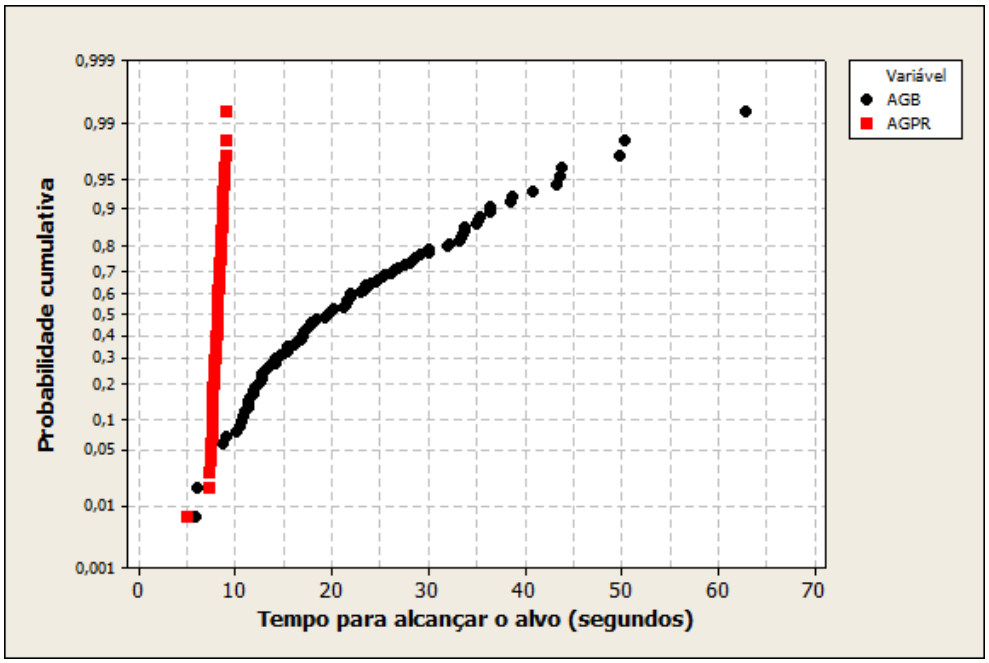
6.1 Probabilidade empírica

Para examinar a robustez dos algoritmos AGPR, em comparação com o AGB, do GPR, em comparação com o GR, e do AGPR, em comparação com o GPR, três experimentos computacionais foram realizados para gerar a FDP (Função de Distribuição de Probabilidade) para obter uma solução em um tempo computacional dado (Aiex et al, 2002). Para ilustrar os resultados destes experimentos, cada algoritmo foi executado com um critério de parada que corresponde a achar uma solução alvo. A utilização deste teste é importante para se comparar diferentes algoritmos ou estratégias para resolver um problema, além do mais esse teste tem sido bastante utilizado como uma ferramenta de desenvolvimento de algoritmos e comparações (Feo et al., 1994; Ribeiro e Resende, 2011).

O teste de probabilidade empírica foi realizado a partir de uma instância composta por 40 tarefas, 20 máquinas e 12 unidades-tarefa. Todos os algoritmos foram aplicados 100 vezes e sempre que o alvo era alcançado, eles eram interrompidos, registrando o tempo de execução em segundos. Ao final, os tempos marcados foram, então, ordenados de forma crescente tal que, para

cada execução i = 1, 2, ..., 100, existe um tempo t_i e uma probabilidade $p_i = (i - 0,5)/100$ associada. O gráfico t_i x p_i gerado é apresentado na Figura 6.3 e corresponde à comparação dos algoritmos GR e GPR. A Figura 6.4 representa o gráfico da comparação dos algoritmos AGB e AGPR. A Figura 6.5 representa o gráfico da comparação dos algoritmos GPR e AGPR.

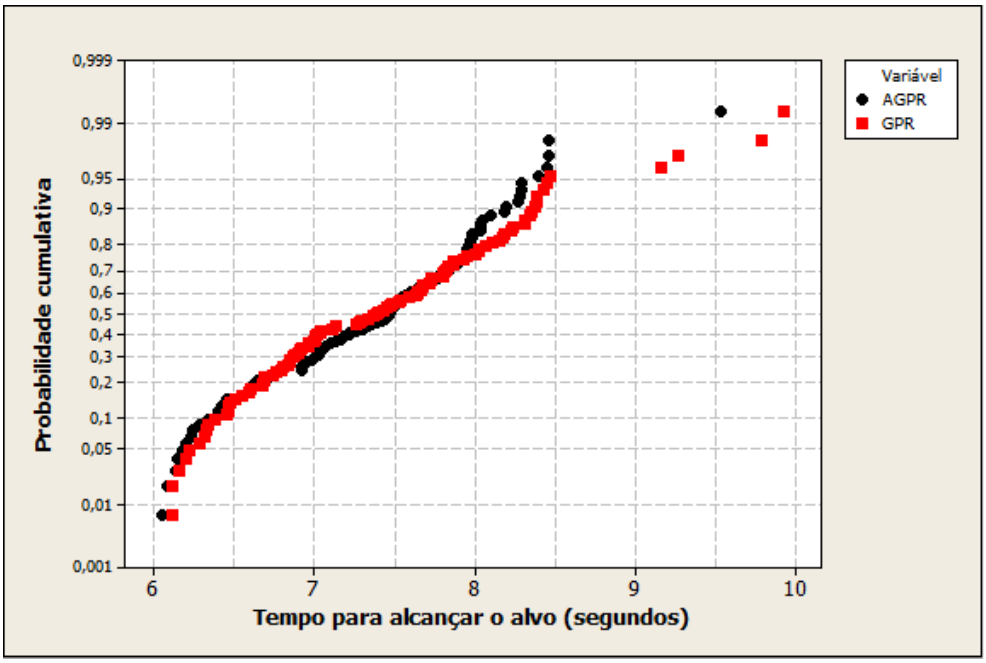
Figura 6.4: Teste de probabilidade empírica para comparar os algoritmos AGB e AGPR.



Ao analisar as curvas da Figura 6.3, nota-se que o GPR superou o GR, encontrando a solução alvo mais rapidamente. Na Figura 6.4, o mesmo acontece para o AGPR que tem maior probabilidade de encontrar a solução alvo mais rapidamente. Tanto na Figura 6.3, quanto na Figura 6.4, a curva mais à esquerda indica que o algoritmo (que tem a curva mais à esquerda) converge mais rapidamente em direção ao valor objetivo. Quanto à Figura 6.5, nota-se uma igualdade nos algoritmos AGPR e GPR. Porém, somente a visualização não garante que um algoritmo seja melhor do que o outro. Há a necessidade de um método objetivo para comparar os algoritmos, explicado em seguida.

Dados dois algoritmos de busca local estocásticos A1 e A2 para o mesmo problema, tem-se que X1 (respectivamente X2) são variáveis aleatórias contínuas representando o tempo necessário pelo algoritmo A1 (e respectivamente A2) para encontrar uma solução para uma instância do problema sob consideração

Figura 6.5: Teste de probabilidade empírica para comparar os algoritmos GPR e AGPR.



(Ribeiro et al., 2011).

Ribeiro et al. (2009) desenvolveu uma ferramenta numérica para calcular a probabilidade Pr($X1 \leq X2$) de o tempo de execução do algoritmo A1 ser menor ou igual ao algoritmo A2. Usando esta ferramenta e considerando X1 como o GPR e X2 como o GR, foi obtido que Pr ($X1 \leq X2$) = 0,74805 com um erro de 0,0039. Considerando X1 como sendo o AGPR e X2 como sendo AGB, tem-se que Pr ($X1 \leq X2$) = 0,94975 e um erro de 0,0007. E, por último, considerando X1 como GPR e X2 como sendo AGPR, tem-se que Pr ($X1 \leq X2$) = 0,50035 e um erro de 0,0031.

7 Conclusões

Este trabalho abordou o Problema de Escalonamento em Máquinas Paralelas (Parallel Machine Scheduling, PMS) com a propriedade de divisão de tarefas.

Foram implementadas quatro metaheurísticas pelo autor do trabalho e estas foram comparados com duas metaheurísticas propostas por Sariçiçek e Çelik (2011).

As metaheurísticas propostas foram: GRASP Reativo, GRASP Reativo com *Path-Relinking*, Algoritmo Genético Básico, Algoritmo Genético com *Path-Relinking*.

Os algoritmos propostos por Sariçiçek e Çelik (2011) foram: *Simulated Annealing* e Busca Tabu.

O GRASP Reativo foi escolhido ao invés do GRASP básico (sua forma pura) por haver uma maior efetividade na criação das soluções no algoritmo construtivo.

Foram testadas duas buscas locais, sendo que a melhor foi escolhida para ser utilizada nos algoritmos. Além das buscas locais, foram testadas duas versões do Algoritmo Genético: uma delas utilizou os métodos de perturbação seguido de busca local e a outra versão utilizou *Path-Relinking*. A versão com *Path-Relinking* foi a escolhida por apresentar valores melhores.

Os 210 problemas foram divididos em três grupos: pequeno, médio e grande. Cada problema foi rodado por 5 vezes e sua média foi calculada para poder realizar um dos métodos do cálculo do RPD.

Houve dois métodos de comparação para o cálculo do RPD. Um deles considerou a média das 5 rodadas de cada problema, como dito no capítulo 6. O outro método de comparação considerou o melhor valor dentre as 5 rodadas.

No primeiro modo de comparação, pode-se notar que tanto o AGPR, quanto o GPR foram melhores do que os outros e, comparando entre si, foram estatisticamente iguais. Os algoritmos SA e TS foram estatisticamente iguais, mas piores do que os outros algoritmos, com exceção do GR (no caso, somente o SA foi igual ao GR estaticamente). Entre as versões básicas e as versões melhoradas do GRASP e do Algoritmo Genético, a versão melhorada apresentou melhores resultados. Isso mostra que as funções, acrescentadas às versões básicas, foram fundamentais na obtenção de melhores soluções.

No segundo modo de comparação, mais uma vez, os algoritmos AGPR e GPR foram melhores do que os outros e, comparando entre si, foram iguais, de acordo com os testes estatísticos mostrados no capítulo anterior. Uma novidade, em relação ao primeiro método, é o GR ser o algoritmo com pior desempenho entre todos. Os algoritmos SA e TS foram iguais estatisticamente. As versões melhoradas dos algoritmos foram melhores em relação às

versões básicas deles.

Portanto, é possível afirmar que a utilização, tanto do AGPR, quanto do GPR, em indústrias, onde ocorre o PMS, proporciona economia na quantidade de recursos utilizados durante o processo de produção e redução no tempo de conclusão das tarefas o que, consequentemente, garante uma redução nos custos de fabricação e torna a indústria mais eficiente e com vantagem competitiva.

Os métodos que podem ser aplicados ao PMS não se limitam aos que foram apresentados neste trabalho. Há muitas alterações que podem ser feitas neste trabalho para que ele fique mais interessante.

Uma possível extensão deste trabalho seria colocar o tempo de preparação dependente de máquina o que aumentaria a complexidade do problema.

Outra possível extensão seria utilizar máquinas paralelas não idênticas, deixando o problema mais genérico e abrangendo uma maior gama de indústrias.

Poder-se-ia, também, utilizar outras heurísticas construtivas para o GRASP, além do EDD que fora utilizado neste trabalho.

Outra sugestão seria testar outros métodos de perturbação e de cruzamento de soluções no Algoritmo Genético.

Além disso, poderia ser testadas outras metaheurísticas tais como ILS, *Scatter Search*, *Bee Colony*, etc.

Por fim, considerando o tempo de parada, este poderia ser ligeiramente aumentado, ou até mesmo, o critério de parada ser modificado.

8 Referências bibliográficas

Aiex, R., Resende, M., e Ribeiro, C.: Probability distribution of solution time in grasp: An experimental investigation. Journal of Heuristics, vol. 8, no. 3, pp. 343 - 373. (2002).

Alex, R., Binato, S., Resende, M.: Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing 29, 393-430 (2003).

Alex, R., Resende, M., Pardalos, P., Toraldo, G.: GRASP with path relinking for the three-index assignment problem. INFORMS J. on Computing 17(2), 224-247 (2005).

Alex, R.: Uma investigação experimental da distribuição de probabilidade de tempo de solução em heurísticas GRASP e sua aplicação na anáse de implementações paralelas. Ph.D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil (2002).

Azizoglu M., Parallel Machine Scheduling To Minimize Total Cost Functions, Ph.D. Thesis, Middle East Technical University (1994).

Azizoglu, M., Kirca, O.: Tardiness minimization on parallel machines. Int. J. Production Economics 55 (1998) 163-168.

Bastos, L.O., Ochi, L.: A Genetic Algorithm with Evolutionary Path-relinking for the SONET Ring Assignment Problem. EngOpt- International Conference on Engineering Optimization. (2008).

Baykasoglu, A.: Capability-based distributed layout approach for virtual manufacturing cells, Int. J. Prod. Res. 41 (11) (2003) 2597-2618.

Beasley, D., Bull, D. R., Martin, R. R. An overview of genetic algorithms: Part 2, research topics. University Computing, (1993), Vol. 15(4), 170-181.

Beraldi P., Ghiani G., Grieco A., Guerriero E.: Rolling-horizon and fix-and-relax heuristics for the parallel machine lot-sizing and scheduling problem with sequence-dependent set-up costs, Comput. Oper. Res. 35 3644-3656 (2008).

Bilge U., Kirac F., Kurtulan M., Pekgun P.A..: Tabu search algorithm for parallel machine total tardines problem, Comput. Oper. Res. 31 (2004)

397-414.

Binato, S., Faria Jr., H., Resende, M.: Greedy randomized adaptive path relinking. In: Sousa, J. (ed.) Proceedings of the IV Metaheuristics International Conference. pp. 393-397 (2001).

Cao, J.; Bedworth, D.D. Flow shop scheduling in serial multi product process with transfer and set-up times. International Journal of Production Research. London, v. 30, n. 8, p. 1819-1830, (1992).

Cheng, TCE e Sin, CCS. (1990). A state-of-the-art review of parallel-machine scheduling research. European Journal of Operational Research, v. 47, n. 3, p.271-292.

De Paula. M. R.: Heurísticas para a minimização dos atrasos em Sequenciamento de Máquinas Paralelas com tempos de Preparação Dependentes da Sequência. 2008. Tese (Doutorado em Ciência da Computação) - Universidade Federal de Minas Gerais. Orientador: Geraldo Robson Mateus.

F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, London, (1997).

Fanjul-Peyro, L., Ruiz, R.: Size-reduction heuristics for the unrelated parallel machines scheduling problem. Computers & Operations Research 38, 301-309 (2011)

Feo T.A., Resende M.G.C.: Greedy randomized adaptive search procedures. J. of Global Optimization, 6 109-133 (1995).

Feo, T.A.; Resende, M.G.C. e Smith, S.H.: A greedy randomized adaptive search procedure for maximum independent set. Operations Research, v. 42, p.860 - 878. (1994).

Festa P., Resende M.G.C.: An annotated bibliography of GRASP, Part II: Applications. International Transactions in Operational Research 16 131-172 (2009).

França Filho, M. F.: Grasp e Busca Tabu aplicados a problemas de programação de tarefas em máquinas paralelas. 2007. Tese (Doutorado em Engenharia Elétrica) - Universidade Estadual de Campinas. Orientador: Vinícius Amaral Armentano.

Frinhani, R.M.D., Silva, R.M.A., Mateus, G.R., Festa, P., Resende, M.G.C.: GRASP with path-relinking for data clustering: a case study for biological data. In proceeding of: Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, (2011).

Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control and Cybernetics 39, 653-684 (2000).

Glover, F.: Multi-start and strategic oscillation methods - Principles to exploit adaptive memory. In: Laguna, M., Gonzáles-Velarde, J. (eds.) Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, pp. 1-24. Kluwer (2000).

Glover, F.: Tabu Search-Part I, ORSA J. Comput. 1 (3) (1989) 190-206.

Glover, F.: Tabu search and adaptive memory programing - Advances, applications and challenges. In: Barr, R., Helgason, R., Kennington, J. (eds.) Interfaces in Computer Science and Operations Research, pp. 1-75. Kluwer (1996).

Gonçalves, F. G., dos Santos, J., R.: Composição florística e estrutura de uma unidade de manejo florestal sustentável na Floresta Nacional do Tapajós, Pará. Acta Amaz. vol.38 no.2 Manaus. (2008)

Kashan, A.H., Karimi, B., Jenabi, M.: A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. Computers & Operations Research 35. 1084-1098 (2008).

Kim Y.D., Shim S.O., Kim S.B., Choi Y.C., Yoon H.M.: Parallel machine scheduling considering a job splitting property, Int. J. Prod. Res. 42 4531-4546 (2004).

Kim, D. W.; Kim, K. H.; Jang, W. e Frank Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. Robotics and Computer-Integrated Manufacturing, v. 18, p. 223-231.

Laguna M., Martí R.: GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing, 11 44-52 (1999).

Mazzini, R.: Estudo de Metaheurísticas Populacionais para a Programação

de Máquinas Paralelas com Tempos de Preparação Dependentes da Seqüência. 1998. 0 f. Tese (Doutorado em Engenharia Elétrica) - Universidade Estadual de Campinas, Fundação de Amparo à Pesquisa do Estado de São Paulo. Orientador: Vinicius Amaral Armentano.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. Management Science, v. 6, n. 1, p. 1-12.

Mendes, A.S., Müller, F.M., França, P.M. e Moscato, P.: Comparing metaheuristic approaches for parallel machine scheduling problems with sequence dependente setup times. Production Planning & Control: The Management of Operations. Volume 13, Issue 2, (2002).

Min, L, Cheng, W.: A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. Arti?cial Intelligence in Engineering 13 (1999) 399-403.

Montgomery, D.G.: Design and Analysis of Experiments. New York: Wiley. (2001)

Müller, F. M., Dias, O. B., Araújo, O. C. B.: Algorithm to solve the unrelated parallel machine scheduling problem. Prod. vol.12 no.2 São Paulo (2002).

Pan, Q. K., Ruiz R.: Local Search methods for the flowshop scheduling problem with flowtime minimization. European Journal of Operational Research 222 (31-42). (2012)

Piersma, N., Dijk, V. W. A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search. Mathl. Comput. Modelling 24 (9), pp. 11-19 (1996).

Pinedo, M.: Scheduling: Theory, Algorithms, and Systems, second ed., Prentice Hall, (2002).

Prais M., Ribeiro C.C.: Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. INFORMS Journal on Computing, 12(3), 164-176 (2000).

Resende M.G.C., Ribeiro C.C.: GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, Metaheuristics: Progress as Real Problem Solvers, pages 29-63. Springer

(2005).

Resende M.G.C., Ricardo M.A.S., Geraldo R.M.: GRASP with path-relinking for the generalized quadratic assignment problem. AT&T& Labs Research Technical Report (2010).

Ribeiro, C., Rosseti, I.: A parallel GRASP for the 2-path network design problem. Lecture Notes in Computer Science 2004, 922-926 (2002).

Ribeiro, C., Uchoa, E., Werneck, R.: A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS Journal on Computing 14, 228-246 (2002).

Ribeiro, C.C. e Resende, M.G.C.: Path-relinking intensification methods for stochastic local search algorithms. Journal of Heuristics, p. 1-22. (2011)

Ribeiro, C.C.; Rosseti, I.: Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. In Proceedings of the VIII Metaheuristics International Conference, Hamburg. (2009).

Rosseti, I.: Heurísticas para o problema de síntese de redes a 2-caminhos. Ph.D. thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil (July 2003).

Ruiz, R., Andrés-Romano, C.:Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. Int J Adv Manuf Technol. (2011)

Ruiz, R., Vallada, E.: Genetic algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times. Grupo de Sistemas de Optimización Aplicada. (2009)

Sariçiçek, I., Çelik, C.: Two meta-heuristics for parallel machine scheduling with job splitting to minimize total tardiness. Applied Mathematical Modelling 35 4117-4126 (2011).

Schaerf, A.: Local Search Technique for Constrained Portfolio Selection Problems. Computational Economics 20: 177-190 (2002).

Serafini P.: Scheduling jobs on several machines with job splitting property, Oper. Res. 44 617-628 (1996).

Shim S.O., Kim Y.D.: A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property, Comput. Oper. Res. 35 863-875 (2008).

Tahar, D.N., F. Yalaoui, C. Chu, L. Amodeo, A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times, Int. J. Prod. Econ. 99 (2006) 63-73.

Xing W., Zhang J.: Parallel machine scheduling with splitting jobs, Discr. Appl. Math. 103 259-269 (2000).

9 Apêndice

Apêndice A - Tabela das médias dos valores absolutos dos resultados de todas as heurísticas

n	m	u	setup	inst.	GR	GPR	AGB	AGPR	SA	TS
2	2	5	short	1	191	191	191	191	191	191
2	2	5	short	2	136	136	141	141	136	136
2	2	5	short	3	118	118	118	118	118	118
2	2	5	short	4	87	87	87	87	87	87
2	2	5	short	5	126	126	126	126	126	126
2	2	5	middle	1	190	190	190	190	190	190
2	2	5	middle	2	229	229	229	229	229	229
2	2	5	middle	3	183	183	183	183	183	183
2	2	5	middle	4	242	242	242	242	242	242
2	2	5	middle	5	355	355	355	355	355	355
2	2	5	long	1	362	362	362	362	362	362
2	2	5	long	2	305	305	305	305	305	305
2	2	5	long	3	382	382	382	382	382	382
2	2	5	long	4	268	268	268	268	268	268
2	2	5	long	5	289	289	289	289	289	289
3	2	5	short	1	345	345	345	345	345	345
3	2	5	short	2	368	368	368	368	399	386
3	2	5	short	3	288	296	296	296	288	296
3	2	5	short	4	207	207	214	207	214	207
3	2	5	short	5	247	247	247	247	247	247
3	2	5	middle	1	494	494	494	494	494	494
3	2	5	middle	2	501	501	501	501	501	501
3	2	5	middle	3	487	487	491	487	491	491
3	2	5	middle	4	806	806	806	806	806	806
3	2	5	middle	5	241	241	241	241	241	241
3	2	5	long	1	997	997	997	997	997	997
3	2	5	long	2	515	515	515	515	557	515
3	2	5	long	3	839	839	839	839	839	839
3	2	5	long	4	417	417	417	417	417	417
3	2	5	long	5	781	781	781	781	781	781
3	3	5	short	1	332	332	332	332	339	356
3	3	5	short	2	315	315	345	315	315	315
3	3	5	short	3	215	215	221	215	215	215
3	3	5	short	4	371	371	371	371	371	371

3	3	5	short	5	379	360	388	360	360	388
3	3	5	middle	1	548	548	548	548	568	548
3	3	5	middle	2	407	407	469	417	407	407
3	3	5	middle	3	267	267	267	267	267	280
3	3	5	middle	4	405	405	508	405	405	408
3	3	5	middle	5	267	267	267	267	267	267
3	3	5	long	1	635	635	635	635	635	635
3	3	5	long	2	528	528	528	528	528	528
3	3	5	long	3	393	393	393	393	393	393
3	3	5	long	4	567	567	567	567	567	567
3	3	5	long	5	410	410	410	410	410	410
4	2	5	short	1	259	259	264	259	259	259
4	2	5	short	2	1031	1031	1068	1031	1031	1046
4	2	5	short	3	333	333	333	333	343	333
4	2	5	short	4	408	408	408	408	408	441
4	2	5	short	5	249	249	252	249	257	249
4	2	5	middle	1	1026	1026	1038	1026	1038	1026
4	2	5	middle	2	614	614	614	614	614	614
4	2	5	middle	3	274	274	274	274	290	290
4	2	5	middle	4	433	433	433	433	433	433
4	2	5	middle	5	1114	1114	1114	1114	1145	1114
4	2	5	long	1	1160	1160	1160	1160	1160	1160
4	2	5	long	2	1220	1220	1288	1220	1220	1220
4	2	5	long	3	650	650	650	650	709	709
4	2	5	long	4	1702	1702	1718	1702	1702	1702
4	2	5	long	5	755	755	821	755	755	755
4	3	5	short	1	460	460	482	445	480	510
4	3	5	short	2	222	222	235	222	238	238
4	3	5	short	3	230	230	261	230	230	230
4	3	5	short	4	289	289	296	289	311	316
4	3	5	short	5	368	357	455	357	380	380
4	3	5	middle	1	445	445	445	445	445	445
4	3	5	middle	2	744	744	813	744	744	768
4	3	5	middle	3	677	677	758	677	677	677
4	3	5	middle	4	658	658	730	658	658	658
4	3	5	middle	5	321	321	414	321	321	321
4	3	5	long	1	585	585	585	585	585	585
4	3	5	long	2	668	668	692	668	668	668

4	3	5	long	3	1023	1023	1023	1023	1023	1023
4	3	5	long	4	1110	1110	1153	1150	1153	1110
4	3	5	long	5	1006	1006	1006	1006	1006	1041
10	4	4	short	1	1235	1105	1135	1076	1151	1124
10	4	4	short	2	1650	1580	1560	1552	1548	1590
10	4	4	short	3	1220	1134	1155	1110	1162	1207
10	4	4	short	4	1401	1298	1313	1277	1312	1363
10	4	4	short	5	955	881	904	844	862	913
10	4	4	middle	1	2168	2039	2140	2084	2140	2039
10	4	4	middle	2	2054	1902	2002	1875	1866	1896
10	4	4	middle	3	1685	1530	1501	1479	1581	1476
10	4	4	middle	4	1933	1804	1839	1800	1788	1841
10	4	4	middle	5	2410	2238	2241	2212	2200	2222
10	4	4	long	1	2223	2115	2180	2125	2281	2147
10	4	4	long	2	1533	1462	1584	1456	1522	1570
10	4	4	long	3	1994	1822	1846	1846	1911	1911
10	4	4	long	4	1735	1623	1633	1621	1636	1633
10	4	4	long	5	3015	2751	2792	2771	2798	2832
15	4	6	short	1	3308	3322	3113	3001	3015	2924
15	4	6	short	2	5707	5404	5236	5368	5342	5303
15	4	6	short	3	5043	4621	4654	4612	4626	4705
15	4	6	short	4	2043	2003	1983	1886	2032	2037
15	4	6	short	5	2048	2001	1784	1803	1887	1890
15	4	6	middle	1	6309	5384	5267	5404	5266	5290
15	4	6	middle	2	5072	4425	4324	4346	4516	4413
15	4	6	middle	3	6714	5594	5825	5571	5432	5445
15	4	6	middle	4	6526	5830	5861	5720	5336	5454
15	4	6	middle	5	5375	4479	4328	4570	4362	4368
15	4	6	long	1	8398	6810	7593	7189	7131	7014
15	4	6	long	2	5098	4741	4689	4577	4587	4573
15	4	6	long	3	6332	5411	5757	5598	5437	5426
15	4	6	long	4	8148	6557	7114	6860	6622	6921
15	4	6	long	5	7676	6237	6924	6351	6256	6187
15	8	6	short	1	2543	2433	2237	2217	2575	2596
15	8	6	short	2	2613	2260	2352	2225	2457	2320
15	8	6	short	3	3071	3034	2826	2564	2810	2887
15	8	6	short	4	2481	2273	2246	2131	2560	2404
15	8	6	short	5	2389	2370	2115	2019	2328	2306

15	8	6	middle	1	3512	2955	3064	3126	3078	3081
15	8	6	middle	2	4206	3480	3812	3623	3565	3708
15	8	6	middle	3	4522	3529	3626	3656	3683	3969
15	8	6	middle	4	5365	4526	4643	4705	4658	5236
15	8	6	middle	5	3586	2800	3146	2944	3008	3017
15	8	6	long	1	6035	4727	5012	5211	5417	4935
15	8	6	long	2	4178	3682	4053	3766	3635	3868
15	8	6	long	3	5911	4678	4886	5059	4965	4984
15	8	6	long	4	5531	4717	4928	4680	4754	5030
15	8	6	long	5	5437	4325	4537	4470	5275	4393
20	8	8	short	1	5274	5286	4525	4560	4459	4956
20	8	8	short	2	5614	5326	4881	4692	4659	5284
20	8	8	short	3	7257	7430	6652	6563	6820	6389
20	8	8	short	4	5533	4991	5117	4579	4863	4639
20	8	8	short	5	5893	5845	5664	5226	5472	5998
20	8	8	middle	1	8917	7740	7729	7462	8236	7938
20	8	8	middle	2	9194	8366	7865	7622	7751	7606
20	8	8	middle	3	8203	6359	7236	6779	6599	6571
20	8	8	middle	4	9526	7720	8677	8426	7044	8345
20	8	8	middle	5	7919	6294	6346	6375	5865	5904
20	8	8	long	1	10075	7539	8182	8493	7960	7442
20	8	8	long	2	14279	9958	12960	11396	10272	10260
20	8	8	long	3	10709	7484	8940	8790	7240	8149
20	8	8	long	4	11210	8233	9631	9514	8336	8876
20	8	8	long	5	11793	7748	9940	9258	8848	8746
20	12	8	short	1	5269	5282	4844	4672	4830	5334
20	12	8	short	2	4629	4742	4348	4235	4567	4336
20	12	8	short	3	3862	3610	3277	3223	3632	3324
20	12	8	short	4	3584	3620	3189	3091	3222	3309
20	12	8	short	5	4016	4033	3418	3573	3694	3939
20	12	8	middle	1	7520	6360	6525	6428	6509	6793
20	12	8	middle	2	7182	6458	6028	5679	5920	6231
20	12	8	middle	3	8397	6359	6955	6788	5816	6112
20	12	8	middle	4	7039	5825	5583	5619	4823	5545
20	12	8	middle	5	7439	5911	5912	5932	5969	6422
20	12	8	long	1	10090	7744	8456	8405	8853	8607
20	12	8	long	2	9888	7839	8640	8676	8263	8824
20	12	8	long	3	8733	6695	7770	7368	7687	7534

20	12	8	long	4	8862	6323	7403	7300	6262	7304
20	12	8	long	5	8844	6165	7228	7173	6495	7019
25	12	10	short	1	7432	7146	6551	6431	6721	7598
25	12	10	short	2	9720	9910	9209	8882	8572	9949
25	12	10	short	3	8484	7969	7127	7039	7336	7804
25	12	10	short	4	7403	7166	6822	6507	7075	7198
25	12	10	short	5	8545	7643	7148	6756	7038	6854
25	12	10	middle	1	10156	8669	8845	8306	8688	8621
25	12	10	middle	2	12589	10785	11246	10779	11452	9636
25	12	10	middle	3	12555	10669	10751	10586	10237	12753
25	12	10	middle	4	10567	7787	8799	8104	9543	7452
25	12	10	middle	5	12044	10230	10841	9627	10142	17862
25	12	10	long	1	17656	12453	13746	13371	13828	16581
25	12	10	long	2	19219	14591	16145	15893	15069	14406
25	12	10	long	3	16355	10943	12455	12404	11017	11725
25	12	10	long	4	13255	9054	11566	11150	8663	9937
25	12	10	long	5	15473	11009	13369	13265	12591	16103
30	15	10	short	1	10302	10208	8897	8574	9224	8100
30	15	10	short	2	10148	9692	9205	8716	9031	8802
30	15	10	short	3	10375	9771	9144	8819	8832	9097
30	15	10	short	4	8628	8824	7471	6713	7126	7526
30	15	10	short	5	8557	8275	7849	6991	7356	7282
30	15	10	middle	1	17183	14031	14800	13842	15028	12760
30	15	10	middle	2	12767	12786	10693	10179	11819	11769
30	15	10	middle	3	14769	13449	11865	11674	10971	10024
30	15	10	middle	4	16346	14295	14311	13213	13217	11146
30	15	10	middle	5	16218	11872	12794	12369	11830	13743
30	15	10	long	1	19886	13565	17118	14841	13832	17179
30	15	10	long	2	19919	14065	17222	16327	14083	14365
30	15	10	long	3	19085	13321	14766	15672	14036	14801
30	15	10	long	4	18652	11807	15849	14844	14801	13251
30	15	10	long	5	17884	12363	15096	14083	14283	16939
40	20	12	short	1	16933	16943	15010	14457	16368	16436
40	20	12	short	2	16065	15797	14572	13669	16187	14297
40	20	12	short	3	15788	16686	15388	14133	16792	14937
40	20	12	short	4	15119	15188	13510	12300	13982	16249
40	20	12	short	5	16022	16666	14730	13733	16706	18196
40	20	12	middle	1	25566	18945	22560	20048	25216	26442

40	20	12	middle	2	24073	22915	23358	20396	27844	31103
40	20	12	middle	3	27840	22877	24471	21987	31091	29417
40	20	12	middle	4	23067	20589	20326	17677	24980	26856
40	20	12	middle	5	24381	18725	20451	17744	14367	26291
40	20	12	long	1	31984	23367	28057	24450	36759	37726
40	20	12	long	2	33591	24423	29425	26264	39063	37528
40	20	12	long	3	29823	19825	26130	22612	37358	35792
40	20	12	long	4	30021	21970	25875	22959	38147	33373
40	20	12	long	5	31728	20303	28328	23343	39308	37719
50	20	12	short	1	22150	22171	20595	18215	28056	24950
50	20	12	short	2	26851	27319	27034	24008	32734	34003
50	20	12	short	3	20919	20766	19120	17379	26239	25868
50	20	12	short	4	22701	23040	23283	20271	26944	27293
50	20	12	short	5	22597	23926	21032	19292	29421	26858
50	20	12	middle	1	35101	29837	31104	26301	37145	37380
50	20	12	middle	2	35366	29398	35267	27960	37267	37254
50	20	12	middle	3	35114	29560	32971	26503	38495	39717
50	20	12	middle	4	37879	33047	37129	29425	42056	40178
50	20	12	middle	5	33811	27408	31776	24563	37600	36492
50	20	12	long	1	53014	32833	48378	38328	61951	58333
50	20	12	long	2	44152	29058	42845	32960	51035	56469
50	20	12	long	3	51010	30673	47063	36602	57861	57186
50	20	12	long	4	47710	30533	47425	36063	57941	60566
50	20	12	long	5	45316	33152	42297	33752	54937	54553

Apêndice B - Tabela dos valores absolutos dos resultados de todas as heurísticas

n	m	u	setup	inst.	GR	GPR	AGB	AGPR	SA	TS
2	2	5	short	1	191	191	191	191	191	191
2	2	5	short	2	136	136	141	141	136	136
2	2	5	short	3	118	118	118	118	118	118
2	2	5	short	4	87	87	87	87	87	87
2	2	5	short	5	126	126	126	126	126	126
2	2	5	middle	1	190	190	190	190	190	190
2	2	5	middle	2	229	229	229	229	229	229
2	2	5	middle	3	183	183	183	183	183	183
2	2	5	middle	4	242	242	242	242	242	242
2	2	5	middle	5	355	355	355	355	355	355
2	2	5	long	1	362	362	362	362	362	362
2	2	5	long	2	305	305	305	305	305	305
2	2	5	long	3	382	382	382	382	382	382
2	2	5	long	4	268	268	268	268	268	268
2	2	5	long	5	289	289	289	289	289	289
3	2	5	short	1	345	345	345	345	345	345
3	2	5	short	2	368	368	368	368	399	386
3	2	5	short	3	288	288	296	296	288	296
3	2	5	short	4	207	207	214	207	214	207
3	2	5	short	5	247	247	247	247	247	247
3	2	5	middle	1	494	494	494	494	494	494
3	2	5	middle	2	501	501	501	501	501	501
3	2	5	middle	3	487	487	491	487	491	491
3	2	5	middle	4	806	806	806	806	806	806
3	2	5	middle	5	241	241	241	241	241	241
3	2	5	long	1	997	997	997	997	997	997
3	2	5	long	2	515	515	515	515	557	515
3	2	5	long	3	839	839	839	839	839	839
3	2	5	long	4	417	417	417	417	417	417
3	2	5	long	5	781	781	781	781	781	781
3	3	5	short	1	332	332	332	332	339	356
3	3	5	short	2	315	315	345	315	315	315
3	3	5	short	3	215	215	221	215	215	215
3	3	5	short	4	371	371	371	371	371	371
3	3	5	short	5	379	360	388	360	360	388

3	3	5	middle	1	548	548	548	548	568	548
3	3	5	middle	2	407	407	469	417	407	407
3	3	5	middle	3	267	267	267	267	267	280
3	3	5	middle	4	405	405	508	405	405	408
3	3	5	middle	5	267	267	267	267	267	267
3	3	5	long	1	635	635	635	635	635	635
3	3	5	long	2	528	528	528	528	528	528
3	3	5	long	3	393	393	393	393	393	393
3	3	5	long	4	567	567	567	567	567	567
3	3	5	long	5	410	410	410	410	410	410
4	2	5	short	1	259	259	264	259	259	259
4	2	5	short	2	1031	1031	1068	1031	1031	1046
4	2	5	short	3	333	333	333	333	343	333
4	2	5	short	4	408	408	408	408	408	441
4	2	5	short	5	249	249	252	249	257	249
4	2	5	middle	1	1026	1026	1038	1026	1038	1026
4	2	5	middle	2	614	614	614	614	614	614
4	2	5	middle	3	274	274	274	274	290	290
4	2	5	middle	4	433	433	433	433	433	433
4	2	5	middle	5	1114	1114	1114	1114	1145	1114
4	2	5	long	1	1160	1160	1160	1160	1160	1160
4	2	5	long	2	1220	1220	1288	1220	1220	1220
4	2	5	long	3	650	650	650	650	709	709
4	2	5	long	4	1702	1702	1718	1702	1702	1702
4	2	5	long	5	755	755	821	755	755	755
4	3	5	short	1	460	460	482	445	480	510
4	3	5	short	2	222	222	235	222	238	238
4	3	5	short	3	230	230	261	230	230	230
4	3	5	short	4	289	289	296	289	311	316
4	3	5	short	5	368	357	455	357	380	380
4	3	5	middle	1	445	445	445	445	445	445
4	3	5	middle	2	744	744	813	744	744	768
4	3	5	middle	3	677	677	758	677	677	677
4	3	5	middle	4	658	658	730	658	658	658
4	3	5	middle	5	321	321	414	321	321	321
4	3	5	long	1	585	585	585	585	585	585
4	3	5	long	2	668	668	692	668	668	668
4	3	5	long	3	1023	1023	1023	1023	1023	1023

4	3	5	long	4	1110	1110	1153	1150	1153	1110
4	3	5	long	5	1006	1006	1006	1006	1006	1041
10	4	4	short	1	1235	1118	1135	1076	1151	1124
10	4	4	short	2	1650	1628	1560	1552	1548	1590
10	4	4	short	3	1220	1173	1155	1110	1162	1207
10	4	4	short	4	1401	1319	1313	1277	1312	1363
10	4	4	short	5	955	943	904	844	862	913
10	4	4	middle	1	2168	2042	2140	2084	2140	2039
10	4	4	middle	2	2054	1902	2002	1875	1866	1896
10	4	4	middle	3	1685	1568	1501	1479	1581	1476
10	4	4	middle	4	1933	1804	1839	1800	1788	1841
10	4	4	middle	5	2410	2245	2241	2212	2200	2222
10	4	4	long	1	2223	2126	2180	2125	2281	2147
10	4	4	long	2	1533	1462	1584	1456	1522	1570
10	4	4	long	3	1994	1822	1846	1846	1911	1911
10	4	4	long	4	1735	1696	1633	1621	1636	1633
10	4	4	long	5	3015	2783	2792	2771	2798	2832
15	4	6	short	1	3308	3308	3113	3001	3015	2924
15	4	6	short	2	5707	5707	5236	5368	5342	5303
15	4	6	short	3	5043	4621	4654	4612	4626	4705
15	4	6	short	4	2043	2043	1983	1886	2032	2037
15	4	6	short	5	2048	2048	1784	1803	1887	1890
15	4	6	middle	1	6309	5546	5267	5404	5266	5290
15	4	6	middle	2	5072	4657	4324	4346	4516	4413
15	4	6	middle	3	6714	6008	5825	5571	5432	5445
15	4	6	middle	4	6526	6202	5861	5720	5336	5454
15	4	6	middle	5	5375	4604	4328	4570	4362	4368
15	4	6	long	1	8398	7110	7593	7189	7131	7014
15	4	6	long	2	5098	4854	4689	4577	4587	4573
15	4	6	long	3	6332	5466	5757	5598	5437	5426
15	4	6	long	4	8148	6779	7114	6860	6622	6921
15	4	6	long	5	7676	6237	6924	6351	6256	6187
15	8	6	short	1	2543	2522	2237	2217	2575	2596
15	8	6	short	2	2613	2520	2352	2225	2457	2320
15	8	6	short	3	3071	3034	2826	2564	2810	2887
15	8	6	short	4	2481	2481	2246	2131	2560	2404
15	8	6	short	5	2389	2371	2115	2019	2328	2306
15	8	6	middle	1	3512	3278	3064	3126	3078	3081

15	8	6	middle	2	4206	3958	3812	3623	3565	3708
15	8	6	middle	3	4522	3998	3626	3656	3683	3969
15	8	6	middle	4	5365	4633	4643	4705	4658	5236
15	8	6	middle	5	3586	2959	3146	2944	3008	3017
15	8	6	long	1	6035	4727	5012	5211	5417	4935
15	8	6	long	2	4178	4178	4053	3766	3635	3868
15	8	6	long	3	5911	4811	4886	5059	4965	4984
15	8	6	long	4	5531	4754	4928	4680	4754	5030
15	8	6	long	5	5437	4325	4537	4470	5275	4393
20	8	8	short	1	5274	5274	4525	4560	4459	4956
20	8	8	short	2	5614	5387	4881	4692	4659	5284
20	8	8	short	3	7257	7257	6652	6563	6820	6389
20	8	8	short	4	5533	5476	5117	4579	4863	4639
20	8	8	short	5	5893	5893	5664	5226	5472	5998
20	8	8	middle	1	8917	8582	7729	7462	8236	7938
20	8	8	middle	2	9194	9155	7865	7622	7751	7606
20	8	8	middle	3	8203	7116	7236	6779	6599	6571
20	8	8	middle	4	9526	7720	8677	8426	7044	8345
20	8	8	middle	5	7919	6596	6346	6375	5865	5904
20	8	8	long	1	10075	8601	8182	8493	7960	7442
20	8	8	long	2	14279	9958	12960	11396	10272	10260
20	8	8	long	3	10709	7575	8940	8790	7240	8149
20	8	8	long	4	11210	8752	9631	9514	8336	8876
20	8	8	long	5	11793	8187	9940	9258	8848	8746
20	12	8	short	1	5269	5269	4844	4672	4830	5334
20	12	8	short	2	4629	4629	4348	4235	4567	4336
20	12	8	short	3	3862	3862	3277	3223	3632	3324
20	12	8	short	4	3584	3584	3189	3091	3222	3309
20	12	8	short	5	4016	4016	3418	3573	3694	3939
20	12	8	middle	1	7520	6403	6525	6428	6509	6793
20	12	8	middle	2	7182	6912	6028	5679	5920	6231
20	12	8	middle	3	8397	6359	6955	6788	5816	6112
20	12	8	middle	4	7039	5825	5583	5619	4823	5545
20	12	8	middle	5	7439	6500	5912	5932	5969	6422
20	12	8	long	1	10090	7744	8456	8405	8853	8607
20	12	8	long	2	9888	9129	8640	8676	8263	8824
20	12	8	long	3	8733	6866	7770	7368	7687	7534
20	12	8	long	4	8862	6509	7403	7300	6262	7304

20	12	8	long	5	8844	7190	7228	7173	6495	7019
25	12	10	short	1	7432	7432	6551	6431	6721	7598
25	12	10	short	2	9720	9720	9209	8882	8572	9949
25	12	10	short	3	8484	7969	7127	7039	7336	7804
25	12	10	short	4	7403	7403	6822	6507	7075	7198
25	12	10	short	5	8545	8178	7148	6756	7038	6854
25	12	10	middle	1	10156	9716	8845	8306	8688	8621
25	12	10	middle	2	12589	11046	11246	10779	11452	9636
25	12	10	middle	3	12555	11787	10751	10586	10237	12753
25	12	10	middle	4	10567	7787	8799	8104	9543	7452
25	12	10	middle	5	12044	11991	10841	9627	10142	17862
25	12	10	long	1	17656	12453	13746	13371	13828	16581
25	12	10	long	2	19219	15323	16145	15893	15069	14406
25	12	10	long	3	16355	11126	12455	12404	11017	11725
25	12	10	long	4	13255	9546	11566	11150	8663	9937
25	12	10	long	5	15473	13358	13369	13265	12591	16103
30	15	10	short	1	10302	10302	8897	8574	9224	8100
30	15	10	short	2	10148	9989	9205	8716	9031	8802
30	15	10	short	3	10375	9771	9144	8819	8832	9097
30	15	10	short	4	8628	8628	7471	6713	7126	7526
30	15	10	short	5	8557	8557	7849	6991	7356	7282
30	15	10	middle	1	17183	16418	14800	13842	15028	12760
30	15	10	middle	2	12767	12767	10693	10179	11819	11769
30	15	10	middle	3	14769	13829	11865	11674	10971	10024
30	15	10	middle	4	16346	14695	14311	13213	13217	11146
30	15	10	middle	5	16218	12141	12794	12369	11830	13743
30	15	10	long	1	19886	14337	17118	14841	13832	17179
30	15	10	long	2	19919	15236	17222	16327	14083	14365
30	15	10	long	3	19085	13321	14766	15672	14036	14801
30	15	10	long	4	18652	13225	15849	14844	14801	13251
30	15	10	long	5	17884	14616	15096	14083	14283	16939
40	20	12	short	1	16933	16933	15010	14457	16368	16436
40	20	12	short	2	16065	16065	14572	13669	16187	14297
40	20	12	short	3	15788	15788	15388	14133	16792	14937
40	20	12	short	4	15119	15119	13510	12300	13982	16249
40	20	12	short	5	16022	16022	14730	13733	16706	18196
40	20	12	middle	1	25566	18945	22560	20048	25216	26442
40	20	12	middle	2	24073	24073	23358	20396	27844	31103

40	20	12	middle	3	27840	25620	24471	21987	31091	29417
40	20	12	middle	4	23067	20589	20326	17677	24980	26856
40	20	12	middle	5	24381	18999	20451	17744	14367	26291
40	20	12	long	1	31984	24065	28057	24450	36759	37726
40	20	12	long	2	33591	24423	29425	26264	39063	37528
40	20	12	long	3	29823	21658	26130	22612	37358	35792
40	20	12	long	4	30021	27549	25875	22959	38147	33373
40	20	12	long	5	31728	25455	28328	23343	39308	37719
50	20	12	short	1	22150	22150	20595	18215	28056	24950
50	20	12	short	2	26851	26851	27034	24008	32734	34003
50	20	12	short	3	20919	20919	19120	17379	26239	25868
50	20	12	short	4	22701	22701	23283	20271	26944	27293
50	20	12	short	5	22597	22597	21032	19292	29421	26858
50	20	12	middle	1	35101	30547	31104	26301	37145	37380
50	20	12	middle	2	35366	35366	35267	27960	37267	37254
50	20	12	middle	3	35114	33564	32971	26503	38495	39717
50	20	12	middle	4	37879	36501	37129	29425	42056	40178
50	20	12	middle	5	33811	31721	31776	24563	37600	36492
50	20	12	long	1	53014	33988	48378	38328	61951	58333
50	20	12	long	2	44152	29058	42845	32960	51035	56469
50	20	12	long	3	51010	34215	47063	36602	57861	57186
50	20	12	long	4	47710	30533	47425	36063	57941	60566
50	20	12	long	5	45316	36810	42297	33752	54937	54553