JOÃO PAULO DE CASTRO MARTINS NOGUEIRA

HEURÍSTICAS HÍBRIDAS PARA O PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS NÃO RELACIONADAS COM PENALIDADES POR ANTECIPAÇÃO E ATRASO

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
2011

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

T
N778h
2011

Nogueira, João Paulo de Castro Martins. 1985-
Heurísticas híbridas para o problema de programação de tarefas em máquinas paralelas não relacionadas com penalidades por antecipação e atraso / João Paulo de Castro Martins Nogueira. - Viçosa, MG, 2011.
xii, 67f. : il. ; 29cm.

Inclui apêndice.
Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referencias bibliográficas: f. 57-60

1. Pesquisa operacional. 2. Otimização combinatória. 3. Solução de problemas. 4. Heurística. I. Universidade Federal de Viçosa. II. Título.

CDD 22. ed. 001.424

JOÃO PAULO DE CASTRO MARTINS NOGUEIRA

HEURÍSTICAS HÍBRIDAS PARA O PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS NÃO RELACIONADAS COM PENALIDADES POR ANTECIPAÇÃO E ATRASO

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: 03 de agosto de 2011.

Luciana Brugiolo Gonçalves

Marcone Jamilson Freitas Souza

José Elias Claudio Arroyo
(Orientador)

AGRADECIMENTOS

Agradeço meus pais, Manoel e Martha, e aos meus irmãos Rafael e Manuella.

Agradeço ao meu orientador José Elias Cláudio Arroyo.

Agradeço também ao secretário da Pós-graduação, Altino Alves de Souza Filho.

A todos do Departamento de Informática, da Universidade Federal de Viçosa.

A todos meus colegas do tempo em que vivi em Viçosa.

A minha família e seu inesgotável suporte.

A meus amigos de Belém que não me esqueceram.

A Universidade Federal de Viçosa.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

NOGUEIRA, João Paulo de Castro Martins, M.Sc., Universidade Federal de Viçosa, agosto de 2011. **Heurísticas híbridas para o problema de programação de tarefas em máquinas paralelas não relacionadas com penalidades por antecipação e atraso.** Orientador: José Elias Cláudio Arroyo.

O presente trabalho trata o problema de sequenciamento de tarefas em máquinas paralelas não relacionadas. No problema abordado, é considerado tanto o tempo de preparação das máquinas, o qual depende da sequência de produção, quanto o tempo de processamento das tarefas, que dependem das máquinas. Cada tarefa possui uma data de entrega que deve ser comprida, caso contrario uma penalidade é aplicada. O objetivo do problema é minimizar a soma de penalidades por atraso e adiantamento das tarefas. Em termos práticos, as penalidades por adiantamento são consequências de custos gerados pela necessidade de estocagem, enquanto as penalidades por atraso das tarefas são originadas de multas contratuais. Primeiramente é utilizado um modelo matemático de programação linear inteira mista (PLIM) para representar o problema. Este modelo é resolvido pelo *software* de otimização CPLEX 12.0. Em seguida é utilizado um algoritmo baseado no método *Greedy Randomized Adaptive Search Procedure* (GRASP) com o objetivo de determinar soluções aproximadas de boa qualidade. Após isso, o método GRASP é hibridizado com o procedimento de intensificação *Path Relinking* (PR) e o método *Iterated Local Search* (ILS), resultando nas heurísticas híbridas GRASP+ILS, GRASP+PR e GRASP+ILS+PR. As heurísticas foram testadas em conjuntos de instâncias de pequeno, médio e grande porte. Os resultados obtidos pelas heurísticas utilizadas são comparados entre si. A análise dos resultados obtidos mostra que a hibridização da heurística GRASP faz com que o desempenho do procedimento melhore.

ABSTRACT

NOGUEIRA, João Paulo de Castro Martins, M.Sc., Universidade Federal de Viçosa, August of 2011. **Hybrid heuristics for the problem of scheduling tasks on unrelated parallel machines with penalties for earliness and tardiness.** Adviser: José Elias Cláudio Arroyo.

This work deals with the problem of sequencing jobs on parallel unrelated machines. In the addressed problem, it was considered both the setup time of the machines, which depends on the job sequence and the processing time of the jobs, which depends on the machines. Each job has a due date which should finish processing, otherwise a penalty is applied. The objective of the problem is to minimize the sum of job penalties for tardiness and earliness. Practically, penalties for earliness are consequence of cost generated by storage while penalties for tardiness are originated from contractual fines. First, it was used a mathematical model, mixed integer linear programming (MILP), to represent the problem. Such model was solved by the optimization software CPLEX 12.0. Following, an algorithm based on the Greedy Randomized Adaptive Search Procedure (GRASP) was utilized in order to determine approximate solutions of good quality. After this, the procedure was hybridized with the intensification procedure Path Relinking (PR) and the Iterated Local Search (ILS) heuristic, resulting in the hybrid heuristics GRASP + ILS, GRASP+PR and GRASP+ILS+PR. These heuristics were tested on sets of instances of small, medium and large size. Results obtained though the heuristics were compared among themselves. Obtained results show that the hybridization of GRASP heuristic can increase the procedure's performance.

Capítulo 1

INTRODUÇÃO

1.1 Motivação

A competitividade crescente no mercado tem levado as indústrias a utilizarem técnicas cada vez mais sofisticadas em seus processos de produção com a finalidade de reduzirem custos e atenderem melhor às necessidades de seus consumidores. As técnicas de planejamento de produção surgem como opção para fornecer vantagens em relação ao problema de decisão para empresas. Tais vantagens são derivadas do melhoramento do gerenciamento de recursos em relação às necessidades da empresa. Sendo assim, o aprimoramento das técnicas de planejamento de produção torna-se importante para uma empresa se manter competitiva no mercado moderno.

Problema de programação de produção é categorizado como problema de decisão e, de acordo com Arenales et al. (2006), geralmente podem ser decompostos hierarquicamente em três níveis: estratégico, tático e operacional. O nível estratégico é o nível mais alto e trabalha com decisões de longo prazo, envolvendo investimentos elevados, como a escolha do que produzir. O nível tático é responsável por decisões de médio prazo provenientes de ordens originadas do nível estratégico. Este nível é responsável pelo planejamento agregado da produção e planejamento de quantidades de produção. O nível operacional controla atividades diárias de produção, derivadas de ordens provenientes do nível tático. Decisões nesse nível orbitam entorno da designação de tarefas em máquinas e a programação das tarefas em cada máquina, ou seja, a sequência de programação das tarefas. As decisões provenientes de um nível inferior possuem grandes consequências no nível imediatamente superior, assim, decisões ao nível operacional afetam fortemente o nível tático que por si possui um imenso efeito sobre o nível estratégico. Portanto, planejar a produção em todos os níveis é importante para atender todos os objetivos desejados. A Figura 1.1 descreve os níveis estruturais

de planejamento e programação da produção.

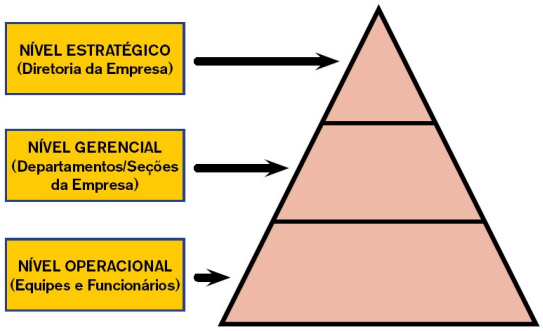


Figura 1.1: Divisão hierárquica dos níveis estruturais

No nível operacional das empresas são encontrados em abundância os problemas denominados de problemas de sequenciamento. Em tais problemas as principais decisões envolvidas são o sequenciamento de n tarefas (corte, soldagem, etc.) para m máquinas e o agendamento (*scheduling*) das tarefas em cada máquina, isto é, os instantes de início e término do processamento das tarefas nas máquinas. No entanto, o sequenciamento e *scheduling* das tarefas normalmente não é otimizado, em relação à minimização de custos e recursos disponíveis a empresa, resultando em prejuízos nos processos de produção.

Tais problemas possuem diversas características e as decisões sobre o modo em que são abordados afetam todo o processo de produção da empresa.

A instância mais simples do problema de sequenciamento é o agendamento de tarefas em apenas uma máquina. No entanto é possível haver ambientes do problema de sequenciamento no qual existem mais de uma máquina. Em tais ambientes, o processamento das tarefas é realizado em paralelo com outras máquinas e qualquer máquina é capaz de processar qualquer tarefa.

Ambientes que utilizam máquinas paralelas, de acordo com Arenales et al. (2006), podem ser categorizados de três formas distintas: idênticas, uniformes e não relacionadas. No ambiente com máquinas idênticas, todas as máquinas processam as tarefas com o mesmo tempo de processamento. No ambiente com máquinas uniformes o tempo de processamento das tarefas depende da máquina na qual estão sendo processadas. No ambiente de máquinas paralelas não relacionadas (MPNR), o tempo de processamento das tarefas depende da tarefa e da máquina na qual ela foi processada.

Além do ambiente de produção, o problema de sequenciamento possui diversas características que são atribuídas às tarefas. Tais características incluem: instantes de disponibilidade, tempos de preparação de máquinas, quebra no processamento de

tarefas, tempo ocioso entre os processamentos das tarefas (*idle time*), penalidades por atraso, penalidades por adiantamento, restrições de precedência, entre outros.

O objetivo (critério de otimização) do problema de sequenciamento também está sujeito a uma ampla gama de variações. Entre os objetivos mais comuns se encontram: a minimização do tempo máximo de conclusão (*makespan*), a minimização do tempo de fluxo total, a minimização do atraso máximo, a minimização da soma de atrasos e a minimização do número de tarefas atrasadas.

Os problemas de sequenciamento são classificados como problemas de dificuldade computacional NP-Difícil. O que caracteriza um problema como NP-Difícil é o fato de que, caso seja capaz de ser resolvido em tempo polinomial, será possível resolver todos os problemas de dificuldade NP em tempo polinomial. Por tal fato, problemas de dificuldade NP-Difícil podem ser considerados tão difícil quanto qualquer problema da classe NP. Atualmente os problemas NP-Difíceis não possuem um algoritmo que obtenha a solução do problema em tempo polinomial. Uma metodologia de solução interessante para problemas dessa dificuldade é a utilização de algoritmos aproximados ou heurísticos, que buscam soluções aproximadas da ótima em um tempo de processamento aceitável.

O problema de sequenciamento é encontrado em diversas áreas de estudo. Suas aplicações estão presentes em áreas díspares como energia, transporte, telecomunicações, circuitos eletrônicos, biologia molecular e finanças. Por exemplo, na ciência da computação ele pode ser encontrado em problemas de multiprocessadores e problemas de base de dados. Portanto, a importância do problema e a dificuldade em encontrar soluções de boa qualidade em tempo computacional aceitável, estabelecem a motivação para esta pesquisa, a qual se baseia na utilização de métodos heurísticos.

Este trabalho aborda o problema de sequenciamento de tarefas num ambiente de máquinas paralelas não relacionadas, com tempos de preparação dependentes da sequência objetivando a minimização da soma de penalidades de atrasos e adiantamentos.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo desta dissertação é testar a eficiência de heurísticas híbridas no problema de sequenciamento de tarefas em máquinas paralelas, com tempos de preparação de máquinas dependentes da sequência da produção e com penalidades por adiantamento e atraso.

1.2.2 Objetivos Específicos

Os objetivos específicos desta dissertação são os seguintes:

* Implementar um modelo matemático do problema.
* Criar um conjunto de instâncias do problema estudado.
* Testar o modelo matemático em diversos cenários utilizando um *software* de otimização.
* Desenvolver metodologias baseadas em heurísticas para resolver o problema.
* Implementar as heurísticas numa linguagem de programação para obter soluções do problema em tempo aceitável.
* Comparar as soluções de instâncias de pequeno e médio porte resolvidas pelo modelo matemático com soluções resultantes das heurísticas implementadas.
* Avaliar o desempenho das heurísticas implementadas na resolução de instâncias de grande porte.

1.3 Estrutura do Trabalho

Esta dissertação está organizada da seguinte maneira:

No Capítulo 1, é apresentada a motivação para este trabalho. Também, neste capítulo são apresentados o objetivo geral e os objetivos específicos deste trabalho.

O Capítulo 2 contém uma descrição detalhada do problema de sequenciamento em MPNR. Em seguida é apresentado um modelo matemático do problema. O capítulo é concluído com uma a revisão bibliográfica do problema.

No Capítulo 3, é apresentado um exemplo de uma solução para o problema de sequenciamento em máquinas paralelas, com tempo de preparação dependente da sequência das tarefas, com o objetivo de minimizar a soma de penalidades por atraso e adiantamento. Logo em seguida, é apresentado o método de avaliação de uma solução do problema. O capítulo é concluído com uma descrição detalhada das heurísticas desenvolvidas para resolver o problema.

No Capítulo 4, é descrita a forma de geração das instâncias do problema para os experimentos e testes das heurísticas apresentadas. Em seguida é mostrado o processo de calibração dos parâmetros de entrada das heurísticas propostas. Logo em seguida, o capítulo apresenta os testes computacionais, que são divididos em três experimentos. O

primeiro experimento trata-se de uma comparação entre os resultados das heurísticas implementadas e o *software* CPLEX. No segundo experimento, os resultados das heurísticas são comparados utilizando testes estatísticos. O terceiro experimento compara a convergência das heurísticas implementadas.

No Capítulo 5, é apresentada a conclusão desta dissertação e recomendações para trabalhos futuros sobre o problema estudado.

Capítulo 2

DEFINIÇÃO DO PROBLEMA

2.1 Caracterização do Problema de sequenciamento em MPNR

O problema de sequenciamento em MPNR, abordado neste trabalho, possui as seguintes características: existem n tarefas, nas quais cada uma deve ser processada em apenas uma das m máquinas; todas as tarefas estão disponíveis para processamento no instante inicial (zero); para cada tarefa i são conhecidos os tempos de processamento $p_{i,k}$ na máquina k; cada tarefa i possui uma data de entrega d_i; cada tarefa i também possui uma penalidade por atraso α_i e uma penalidade por adiantamento β_i. O instante de início do processamento da tarefa i é denominado de t_i e sua data de conclusão de C_i.

Se $C_i < d_i$ então a tarefa i está adiantada (com relação a sua data de entrega). Este adiantamento é calculado por $E_i = max(0, d_i - C_i)$. A penalidade a ser paga é $\beta_i \times E_i$. Porém, se $C_i > d_i$ a tarefa i é considerada atrasada por $T_i = max(0, C_i - d_i)$ e é paga uma penalidade de $\alpha_i \times T_i$. São permitidos tempos ociosos (*idle time*) das máquinas, ou seja, em alguns períodos de tempo as máquinas podem ficar sem executar nenhuma tarefa de tal maneira que esta seja finalizada o mais próximo da sua data de entrega, evitando adiantamentos.

O problema considera tempos de preparação dependentes da sequência de tarefas. Finalizado o processamento de uma tarefa i, na máquina k e antes de iniciar o processamento da tarefa j existe um tempo de preparação $s_{i,j,k}$. O tempo de preparação não é considerado quando a máquina executa a primeira tarefa da sequência.

2.2 Modelagem Matemática do Problema

Neste trabalho é proposto um modelo matemático de programação linear inteira misto (PLIM) para o problema abordado. Este modelo é uma adaptação do modelo apresentado em Arenales et al. (2006). A adaptação feita é a utilização de penalidades de atraso e adiantamento, no calculo do valor da função objetivo. Para tanto, sejam os seguintes parâmetros e variáveis:

n = quantidade de tarefas.

m = quantidade de máquinas.

$p_{i,k}$ = tempo de processamento da tarefa i na máquina k.

$s_{i,j,k}$ = tempo de preparação da máquina k para processar a tarefa j depois de finalizar a tarefa i.

d_i = data de entrega da tarefa i.

α_i = penalidade por cada unidade de tempo que a tarefa i está atrasada.

β_i = penalidade por cada unidade de tempo que a tarefa i está adiantada.

M = um número suficientemente grande.

As variáveis de decisão do modelo são:

$C_{i,k}$ = instante de conclusão do processamento da tarefa i na máquina k.

$x_{i,j,k} = \begin{cases} 1, & \text{se a tarefa } i \text{ precede a tarefa } j \text{ na máquina } k \\ 0, & \text{caso contrário} \end{cases}$

$T_i = max(C_i - d_i, 0)$, o atraso da tarefa i.

$E_i = max(d_i - C_i, 0)$, o adiantamento da tarefa i.

O modelo de PLIM é o seguinte:

$$\min f = \sum_{i=1}^{n} (\alpha_i T_i + \beta_i E_i) \tag{2.1}$$

$$\sum_{k=1}^{m} \sum_{i=0}^{n} x_{i,j,k} = 1, \qquad j = 1, \cdots, n \tag{2.2}$$

$$\sum_{j=1}^{n} x_{0,j,k} \leq 1, \qquad k = 1, \cdots, m \tag{2.3}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{n} x_{i,h,k} - \sum_{\substack{j=0 \\ j \neq h}}^{n} x_{h,j,k} = 0, \tag{2.4}$$

$$h = 1, \cdots, n; \qquad k = 1, \cdots, m$$

$$C_{0,k} = 0, \qquad k = 1, \cdots, m \tag{2.5}$$

$$C_{j,k} \geq C_{i,k} - M + (p_{j,k} + s_{i,j,k} + M)x_{i,j,k} \tag{2.6}$$
$$i = 0, \cdots, n;\ j = 1, \cdots, n;\ k = 1, \cdots, m$$

$$T_i \geq C_{i,k} - d_i; \qquad i = 1, \cdots, n;\ k = 1, \cdots, m \tag{2.7}$$

$$E_i \geq d_i - C_{i,k}; \qquad i = 1, \cdots, n;\ k = 1, \cdots, m \tag{2.8}$$

$$T_i \geq 0, E_i \geq 0, x_{i,j,k} \in \{0, 1\}, \qquad i = 0, \cdots, n, \qquad j, k = 1, \cdots, n \tag{2.9}$$

A função objetivo f em (2.1) é definida como a soma das penalidades de atraso e adiantamento. As restrições (2.2) indicam que cada tarefa j pode apenas ter uma única tarefa precedente em uma única máquina. As restrições (2.3) garantem que cada máquina k, se usada, tem uma única sequência exclusiva de processamento. As restrições (2.4) garantem que cada tarefa j tenha uma única tarefa imediata sucessora, com exceção da tarefa 0 na máquina k. O modelo utiliza a tarefa 0 para estabelecer o início e fim de uma sequência na máquina k e as restrições (2.5) garantem que esta tarefa tenha o tempo de conclusão igual a zero. Esta tarefa possui tempo de processamento em todas as maquinas, data de entrega e penalidades de atraso e adiantamento igual a 0. Se $x_{i,j,k} = 1$ então a restrição (2.6) implica que, na máquina k,

$$C_{j,k} \geq C_{i,k} + s_{i,j,k} + p_{j,k}$$

ou seja, a tarefa i não sobrepõe o tempo de processamento da tarefa j na máquina k. Se $x_{i,j,k} = 0$, (2.6) implica que

$$C_{j,k} - C_{i,k} \geq -M$$

isto é, a restrição (2.6) fica desativada. As restrições (2.7) garantem que cada tarefa i concluída na máquina k tenha seu atraso igual à diferença entre seu instante de conclusão e sua data de entrega. As restrições (2.8) indicam que cada tarefa i finalizada na máquina k tem seu adiantamento igual à diferença entre sua data de entrega e seu tempo de conclusão. As restrições (2.9) indicam os tipos das variáveis de decisão do modelo.

2.3 Revisão Bibliográfica

O problema de sequenciamento num ambiente de uma única máquina, sem a utilização de tempo de preparação, objetivando a minimização do atraso total é, como feito por Du & Leung (1990), um problema de dificuldade NP-Difícil. O problema de sequenciamento com tempos de preparação dependentes da sequência em uma máquina, com o objetivo de minimizar o número de tarefas atrasadas também foi provado ser NP-Difícil por Liaee & Emmons (1997). Problemas com uma máquina que consideram tempos de preparação de máquina dependente da sequência, com objetivo de minimizar a soma de atrasos ponderados, foi demonstrado ser um problema de dificuldade NP-Difícil por Pinedo (1995) e Du & Leung (1990). Garey & Johnson (1979) e Lenstra et al. (1977) demonstraram que, problemas em duas máquinas idênticas, com objetivo de minimizar o *makespam* também são NP-Difícil. A pesquisa de Potts & M.Y (2000) demonstra que problemas de sequenciamento em uma máquina com tempo de preparação de máquina são considerados NP-Difíceis. Como esses problemas são casos particulares do problema tratado nesta dissertação, o problema de sequenciamento de tarefas em máquinas paralelas não relacionadas com penalidades por adiantamento e atraso é também NP-Difícil.

Na literatura foram publicados diversos artigos relacionados ao problema de sequenciamento em ambientes de máquinas paralelas. Alguns dos principais trabalhos sobre o assunto encontram-se organizados aqui.

- Cheng & Sin (1990) produziram uma revisão bibliográfica sobre o problema de sequenciamento de tarefas em MPNR abordando diversos objetivos.

- Kim et al. (2003a) investigaram o problema de sequenciamento de tarefas com objetivo de minimizar o atraso total ponderado em um ambiente de máquinas paralelas não relacionadas.

- Kim et al. (2003b) pesquisaram a minimização do atraso total ponderado em um ambiente de máquinas paralelas idênticas.

- Logendran & Saputra (2007) abordaram o problema de sequenciamento de tarefas objetivando a minimização da soma ponderada de atrasos em máquinas paralelas não relacionadas utilizado um procedimento de Busca Tabu, sendo que tarefas e máquinas podem não estarem disponíveis no instante inicial.

No entanto, estudos sobre o problema de sequenciamento em máquinas paralelas que implementam penalidades por adiantamento e atraso de tarefas são mais recentes

que aqueles que possuem outros objetivos. Para estes, normalmente as tarefas são concluídas com antecedência para evitar custos por atraso. No entanto, muitas empresas utilizam o modelo *Just-in-Time*, no qual penaliza a conclusão antecipada e atrasada de tarefas.

O modelo de produção *Just-in-Time* (JIT) surgiu na década de 70 como o principal componente do Sistema Toyota de Produção proposto por Ohno (1988) e os resultados de suas implementações mostraram a importância do planejamento das atividades de produção. O modelo JIT preconiza a minimização de atrasos e adiantamento da conclusão das tarefas, ou seja, que as tarefas sejam concluídas o mais próximo possível de suas datas de entrega. Isto se deve ao fato que atrasos e adiantamentos incrementam custos ao processo de produção. Considerando isto, a ocorrência de atrasos e adiantamentos, e atribuída penalidade.

De acordo com Liaw (1999), a conclusão de uma tarefa com atraso resulta em multas contratuais, perda de credibilidade da empresa e redução de vendas. De igual forma, uma tarefa concluída com adiantamento resulta em custos adicionais com a necessidade de disponibilização antecipada de capital, demanda de espaço para armazenamento ou necessidade de outros recursos para manter e gerenciar o estoque (França Filho, 2007). O modelo JIT fornece a vantagem de reduzir atrasos na entrega de produtos e minimizar os custos de estoque e inventário causados por adiantamentos (Hirano & Makota, 2006).

O problema de sequênciamento de tarefas que utiliza o modelo JIT tem como objetivo minimizar a soma ponderada de adiantamento e atrasos. Baker (1990) descreve que tal ambiente de produção reflete melhor o modelo *Just-in-Time*.

A seguir são apresentados os trabalhos envolvendo penalizações por adiantamento e por atraso utilizado máquinas paralelas.

- James & Buchanan (1997) implementaram um procedimento de Busca Tabu objetivando resolver o problema de minimização da soma ponderada de custos de adiantamentos e atrasos de tarefas.

- Zhu & Heady (1998) desenvolveram uma heurística gulosa para o problema de máquinas paralelas não relacionadas com o objetivo de minimizar a soma ponderada de custos de adiantamentos e de atraso com tempos de preparação dependente da sequência de tarefas.

- Serifoglu & Ulusoy (1999) utilizam Algoritmos Genéticos em máquinas paralelas uniformes para minimizar a soma ponderada de custos de adiantamento e de

atraso considerando tempos de preparação de máquina dependentes da sequência de tarefas.

- Balakrishnan et al. (1999) desenvolveram um modelo matemático para o problema de máquinas paralelas uniformes com o objetivo de minimizar a soma ponderada de custos de adiantamento e atraso com instantes de liberação das tarefas e tempo de preparação.

- Radhakrishnan & Ventura (2000) utilizam uma heurística com *Simulated Annealing* para o problema máquinas paralelas idênticas com o objetivo de minimizar a soma de adiantamentos e atrasos das tarefas. Tempos de preparação de máquina são dependentes da sequência de tarefas e tempos ociosos entre tarefas não são permitidos.

- França Filho (2007) implementa as heurísticas GRASP e Busca Tabu para o problema de máquinas paralelas objetivando a minimizar a soma ponderada dos custos de atraso e adiantamentos das tarefas considerando tempo de preparação de máquina dependente da sequência e instantes de liberação para cada tarefa.

Capítulo 3

METODOLOGIA

Neste capítulo são descritas as metodologias utilizadas para resolver o problema de sequenciamento abordado neste trabalho. Como mencionado, o problema de sequenciamento é da classe de problemas NP-difíceis, ou seja, não é conhecido um método de solução que o resolva em tempo computacionalmente aceitável. No entanto, é interessante a utilização de procedimentos heurísticos, pois são capazes de obter uma solução de boa qualidade (próxima à solução ótima) em tempo computacional aceitável. Porém, tais procedimentos não podem garantir que a solução obtida é a solução ótima, nem quanto distante esta solução está de ser a solução ótima.

Neste trabalho é aplicada a heurística GRASP para resolver o problema de sequenciamento em máquinas paralelas não relacionadas, considerando tempos de preparação de máquina e penalidades por adiantamento e atraso. São propostos métodos híbridos combinando a heurística GRASP com a técnica de *Path Relinking*. Esta técnica é utilizada como uma estratégia de intensificação para a solução ótima local obtida após a etapa de busca local do GRASP, como feito no trabalho de Resende & Ribeiro (2005). Também é proposto um método GRASP híbrido que utiliza a heurística ILS para substituir a etapa de busca local, assim como no trabalho de Ribeiro & Urrutia (2007). Este trabalho também apresenta um método que utiliza GRASP com ILS no qual combina a técnica de PR, gerando uma heurística híbrida que utiliza os três procedimentos.

A seguir apresenta-se a representação e o método de avaliação de uma solução, os procedimentos GRASP, ILS e *Path Relinking*. Este capítulo é concluído com a descrição das heurísticas hibridas implementadas.

3.1 Representação de uma solução

Uma solução s do problema é representada por um conjunto de m subsequências s_i. Cada subsequência s_i corresponde às tarefas processadas na máquina i. Um exemplo de uma solução com 12 tarefas e 3 máquinas é s =[[2,5,1,4],[6,11,9,10],[8,7,12,3]], onde s_1=[2,5,1,4], s_2=[6,11,9,10] e s_3=[8,7,12,3] representam as sequências de tarefas nas máquinas 1, 2 e 3.

A Figura 3.1 representa um sequenciamento típico de uma solução para uma instância que possui 12 tarefas e 3 máquinas. Cada máquina da solução s é implementada como subsequência s_i, em que os elementos correspondem às tarefas em cada máquina. Os tempo de início e de conclusão das tarefas são armazenados, respectivamente, nos arranjos t e C.

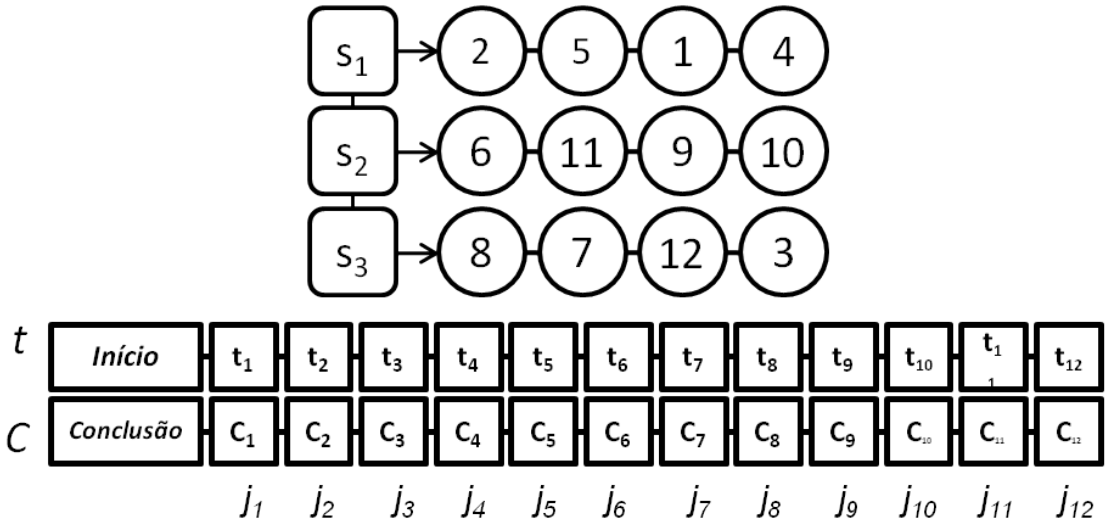


Figura 3.1: Exemplo de uma solução com 12 tarefas e 3 máquinas.

3.2 Avaliação de uma solução

Dada uma solução (conjunto de sequências de tarefas) $s = (s_1, s_2, ..., s_m)$, para calcular o valor de sua função objetivo, é necessário determinar os tempos ótimos de início $(t_i, i = 1, ..., n)$ e conclusão $(C_i, i = 1, ..., n)$ do processamento de cada tarefa i. Os tempos ótimos de inicio e conclusão das tarefas são necessários para calcular a menor soma de penalidades de atraso e adiantamento de uma determinada solução. Para obter estes tempos, de maneira que estejam o mais próximo da data de entrega (evitando atrasos e adiantamentos), é aplicado um algoritmo ótimo que possui tempo de execução polinomial. Na literatura, este algoritmo foi proposto, para o caso de uma máquina, por

Lee & Choi (1995). Wan & Yen (2002) adaptaram este algoritmo para o caso de janelas de entrega distintas em lugar de datas de entrega distintas. Neste trabalho propõe-se uma adaptação desse algoritmo que consiste na utilização de datas de entregas distintas para o ambiente de máquinas paralelas não relacionadas.

3.2.1 Cálculo dos Tempos Ótimos de Conclusão

Seja s_k a sequência de tarefas da máquina k e $B_j = (j_0, j_1, ...j_{n_k} \subseteq s_k)$ um conjunto de tarefas denominado de bloco, ou seja, um conjunto de tarefas que são processadas consecutivamente, sem tempo ocioso de máquina entre elas. Suponha que há b blocos em s_k e a primeira e a última tarefa do bloco B_j sejam, respectivamente, $first(B_j)$ e $last(B_j)$.

O Algoritmo 1 apresenta um pseudocódigo para determinar os tempos de início e conclusão das tarefas de uma sequência s_k com n_k tarefas processadas na máquina k. Neste algoritmo, inicialmente, determina-se os tempos de início e conclusão da primeira tarefa da sequência, j_1. Esta tarefa deve ser concluída na sua data de entrega se seu tempo de processamento é menor que a sua data de entrega ($p_{j_1,k} < d_{j_1}$), caso contrário a tarefa concluirá no seu tempo de processamento ($C_{j_1} = p_{j_1,k}$), ou seja, iniciará no instante zero. Em seguida, o algoritmo determina os tempos de início e conclusão das demais tarefas da sequência s_k, ou seja, $j_1, j_2, ..., j_{n_k}$.

Para cada tarefa j_i, é verificado se ela pode ser concluída exatamente na sua data de entrega. Esta verificação é feita levando em conta a tarefa anterior j_{i-1}. Na Figura 3.2 mostra-se os dois casos que existem no cálculo dos tempos de início e conclusão da tarefa j_i. No primeiro caso, ilustrado na Figura 3.2a, observa-se que $C_{j_{i-1}} + s_{j_{i-1},j,k} + p_{j_i,k} < d_{j_i}$. Neste caso, o tempo de conclusão da tarefa j_i será exatamente na sua data de entrega (d_{j_i}) e haverá um tempo de ocioso da máquina. Além disso, a tarefa j_i formará um novo bloco. No segundo caso, ilustrado na Figura 3.2b, no qual $C_{j_{i-1}} + s_{j_{i-1},j,k} + p_{j_i,k} \geq d_{j_i}$, a tarefa finalizará depois da sua data de entrega. Neste caso a tarefa estará no mesmo bloco que a tarefa anterior e não haverá tempo de ocioso de máquina. Note que, nos dois casos o tempo de início da tarefa j_i será $t_{j_i} = C_{j_i} - p_{j_i,k}$.

Após serem obtidos os tempos de início e conclusão de uma tarefa j_i, é necessário verificar o posicionamento do bloco no qual a tarefa pertence. Inicialmente é necessário calcular os possíveis tempos de início do bloco. Esses tempos são obtidos posicionando cada tarefa do bloco na sua data de entrega. Isto ocorre pelo fato do ponto mínimo do bloco ser concomitante com um dos possíveis tempos de início do bloco. Entre os possíveis tempos de inicio do bloco, aquele que possuir a menor soma de penalidades de atrasos e adiantamento é o ponto mínimo do bloco.

(a) tarefa com tempo de conclusão igual a data de entrega.

(b) tarefa tempo de conclusão maior que a data de entrega.

Figura 3.2: Casos que exitem no cálculo de datas de inicio e conclusão.

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

1. $t_{first(B_j)} = 0$;
2. o ponto mínimo é alcançado;
3. $t_{first(B_j)}$ torna-se igual a $C_{j_{i-1}} + s_{j_{i-1},j,k}$;

Se ocorrer o caso 3, o bloco B_j é concatenado ao bloco B_{j-1}, resultando em um novo bloco B_{j-1}. Logo, o ponto mínimo do novo bloco B_{j-1} deve ser obtido recursivamente. Este procedimento deve ser realizado até que ocorra o caso 1 ou 2 para cada bloco.

Na próxima seção é apresentado um exemplo detalhado do funcionamento do Algoritmo 1 que determina os tempos ótimos de início e conclusão das tarefas de uma sequência de tarefas correspondente a uma máquina.

O procedimento *moverbloco* movimentará o bloco B_j até o seu ponto mínimo e, se necessário, combinará este bloco com o bloco anterior.

3.2.2 Exemplo do cálculo dos tempos ótimos de início de processamento

A Tabela 3.1 apresenta os dados de entrada para uma instância do problema de sequenciamento com $n = 12$ tarefas e $m = 3$ máquinas.

Os parâmetros de entrada para o problema de MPNR consistem de uma matriz $n \times m + 3$ com os tempos de processamento das tarefas. A Tabela 3.1 apresenta os tempos de processamento de $n = 12$ tarefas em $m = 3$ máquinas. A Tabela 3.1 também contém as datas de entrega d_i de cada tarefa i. Por exemplo, a tarefa 10 na máquina 2 possui o tempo de processamento igual 6 e sua data de entrega é igual a 29. Também é apresentado na Tabela 3.1, as penalidades de atraso α_i e adiantamento β_i, para

Algoritmo 1 <Completion Time(n_k, s_k)>

$i \leftarrow 1$;
$b \leftarrow 1$;
$C_{j_1} \leftarrow max(d_{j_i}, p_{j_i,k})$;
$t_{j_1} \leftarrow C_{j_1} - p_{j_1,k}$;
for $(i = 2, ..., n_k)$ **do**
  if $C_{j_{i-1}} + s_{j_{i-1},j,k} + p_{j_i,k} < d_{j_i}$ **then**
    $b \leftarrow b + 1$;
    $first(B_b) \leftarrow last(B_b) \leftarrow j_i$;
    $C_{j_i} \leftarrow d_{j_i}$;
    $t_{j_i} \leftarrow C_{j_i} - p_{j_i,k}$;
  else
    $last(B_b) \leftarrow j_i$;
    $C_{j_i} = C_{j_{i-1}} + p_{j_i,k} + s_{j_{i-1},j_i,k}$;
    $t_{j_i} \leftarrow C_{j_i} - p_{j_i,k}$;
  end if
  $moverbloco(B_b)$;
end for

Tabela 3.1: Dados de entrada para uma instância do problema.

i	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$	d_i	α	β
1	6	12	14	25	7	9
2	5	17	16	5	9	7
3	15	11	5	16	9	9
4	3	21	14	30	9	9
5	7	19	11	20	2	1
6	18	6	17	10	3	5
7	15	14	5	12	2	9
8	13	15	7	10	1	7
9	15	7	11	25	3	4
10	11	6	17	29	6	5
11	17	4	19	13	8	7
12	15	17	3	14	4	8

cada tarefa i. Para facilitar este exemplo, não é considerado tempos de preparação de máquina entre tarefas.

Para uma solução exemplo s, vamos atribuir às sequências de tarefas $s = [[2, 5, 1, 4], [5, 11, 9, 10], [8, 7, 12, 3]]$, ou seja, para a máquina 1 a subsequência de tarefas é $[2, 5, 1, 4]$, para máquina 2 é $[5, 11, 9, 10]$ e para máquina 3 é $[8, 7, 12, 3]$. Em seguida é descrito passo a passo o processo de cálculo de tempo de conclusão ótimo, utilizando a primeira sequência s_1 =[2,5,1,4].

O processo de inicialização determina os tempos de início e o tempo de conclusão

da primeira tarefa da subsequência. Calculando o tempo de conclusão da primeira tarefa $j_1 = 2$, a conclusão do processamento será $C_2 = max(d_2, p_{2,1})$, ou seja, $C_2 = 5$. O tempo de início será $t_{j_1} = C_2 - p_{2,1}$, ou seja, $t_2 = 0$. A inserção da primeira tarefa na máquina 1 pode ser vista na Figura 3.3.

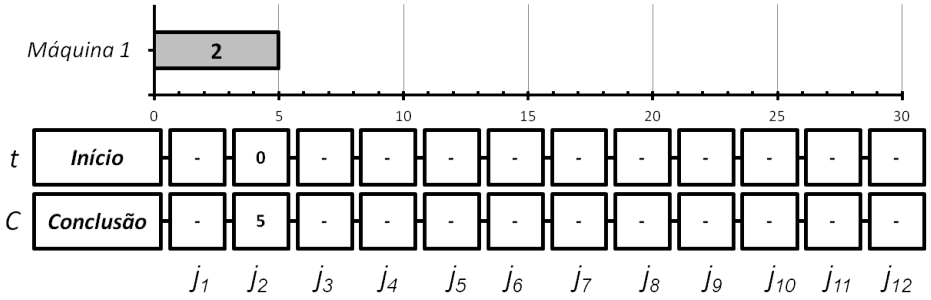


Figura 3.3: Sequência s_1 após a inserção da tarefa 2.

Em seguida determina-se o tempo de início e conclusão da próxima tarefa da sequência, ou seja, da tarefa $j_2 = 5$. Como $C_2 + p_{5,1} < d_5$ ou seja, $5 + 7 + 0 < 20$, então a tarefa 5 irá iniciar um novo bloco (bloco 2), sendo que $C_5 = d_5$ e $t_5 = C_5 - p_{5,1}$, ou seja $C_5 = 20$ e $t_5 = 13$. O posicionamento da tarefa 5 é apresentado na Figura 3.4.

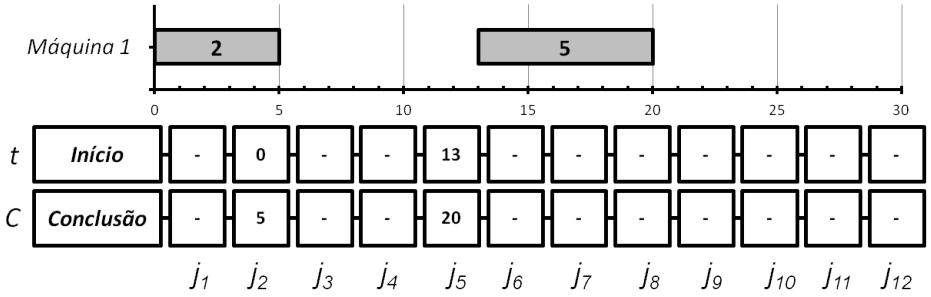


Figura 3.4: Sequência s_1 após a inserção da tarefa 5.

Depois de obter o tempo de início e conclusão de uma tarefa j_i, com $i \geq 2$ é necessário posicionar o bloco no seu ponto mínimo, porém, pelo fato deste bloco conter apenas uma tarefa ele já está no seu ponto mínimo.

Em seguida é necessário calcular o tempo de início e conclusão da terceira tarefa da subsequência, $j_3 = 1$. Como $C_5 + p_{1,1} > d_1$, ou seja, $20 + 6 + 0 > 25$ então a tarefa 1 será inserida como a última tarefa do bloco atual (bloco 2), sendo que $C_1 = C_5 + p_{1,1}$ e $t_1 = C_1 - p_{1,1}$, ou seja, $t_1 = 20$ e $C_1 = 26$. O posicionamento da tarefa 1 é apresentado na Figura 3.5.

Agora é necessário verificar a posição do bloco 2 para garantir que ele esteja no seu ponto mínimo. Inicialmente é necessário verificar os possíveis tempos de início do

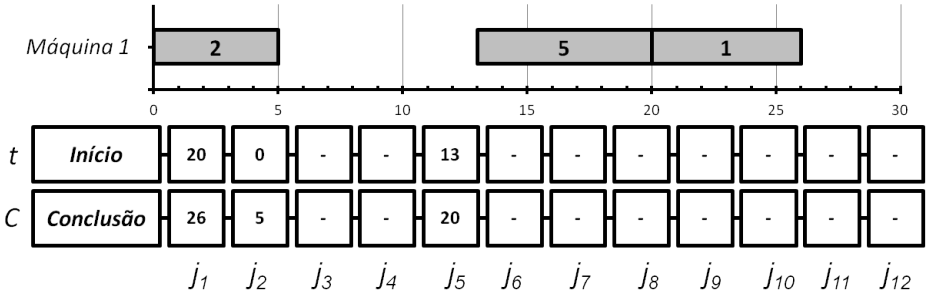


Figura 3.5: Sequência s_1 após a inserção da tarefa 1.

bloco. Esses tempos são obtidos posicionando cada tarefa do bloco em sua respectiva data de entrega. Isso ocorre devido ao ponto mínimo do bloco ser concomitante com um dos possíveis tempos de início do bloco.

No exemplo, os possíveis tempos de início do processamento do bloco são 12 e 13. Depois de obter os possíveis tempos de início do bloco é necessário calcular a soma de penalidades de atraso e de adiantamento para cada. Cada tempo possuirá um valor correspondente à soma das penalidades de atraso e de adiantamento do bloco, se ele tivesse seu início nesse tempo. O tempo com a menor soma de penalidades positiva é considerada como ponto mínimo do bloco. Se por alguma razão o ponto mínimo não for alcançável, seja ele negativo ou menor que a conclusão da tarefa antecedente a primeira tarefa do bloco, então o bloco será alocado para o instante viável mais próximo a este ponto. No exemplo, se o bloco 2 iniciar no tempo 12, a tarefa 5 ficará adiantada em 1 e a soma de penalidades será igual a 1. Porém, se o bloco iniciar no instante 13, a tarefa 1 ficará atrasada em 1 e a soma de penalidades será igual a 7. Como o tempo de início 12 possui a menor soma de penalidades, ele é considerado como ponto mínimo do bloco, ou seja, a primeira tarefa deste bloco iniciará no tempo 12. Na Figura 3.6 mostra-se o deslocamento do bloco 2 para seu ponto mínimo.

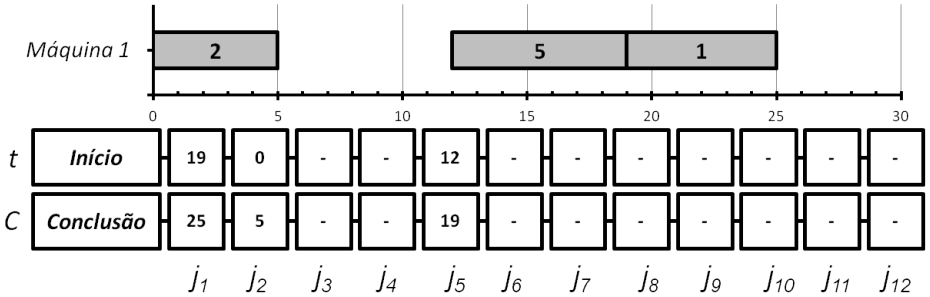


Figura 3.6: Sequência s_1 após movimentar o bloco.

Para concluir o cálculo dos tempos de conclusão ótimo da subsequência s_k é necessário calcular o início e conclusão da quarta tarefa $j_4 = 4$. Como $C_1 + p_{4,1} < d_4$

ou seja $20 + 3 + 0 < 30$ então a tarefa 1 irá iniciar um novo bloco (bloco 3), sendo que $t_4 = d_4 - p_{4,1}$ e $C_4 = d_4$. Como a tarefa 4 é a única tarefa do bloco 3, ela já está no seu ponto mínimo. A Figura 3.7 representa o estado final da primeira subsequência, na máquina 1, após o cálculo dos tempos de início e conclusão das tarefas.

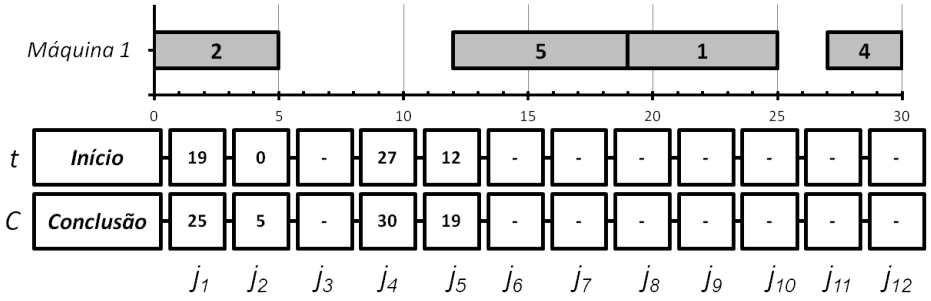


Figura 3.7: Sequência s_1 após da inserção de todas as tarefas.

O processo é repetido para as demais subsequências de s, ou seja, para as outras máquinas, no qual o resultado é apresentado na Figura 3.8. Note que nas máquinas 1 e 2 existem tempos de ociosos e na máquina 3 não existem esses tempos.

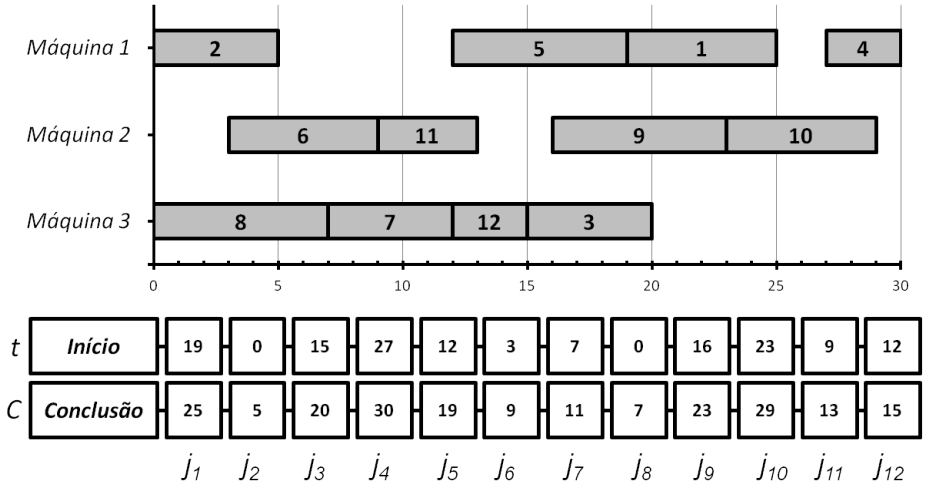


Figura 3.8: Solução s após da inserção de todas as tarefas

3.3 Heurística GRASP

A heurística GRASP introduzida por Feo & Resende (1989) pode ser aplicada para obter soluções de boa qualidade de modo eficiente em diferentes problemas de otimização combinatória. Esta heurística consiste em um processo iterativo composto por duas etapas principais, construção e busca local. A etapa de construção produz uma

solução viável utilizando um algoritmo guloso com escolhas parcialmente aleatórias. Esta solução é submetida à etapa de busca local para ser melhorada até encontrar uma solução ótima local.

A heurística GRASP é um processo iterativo na qual a presença de aleatoriedade em cada iteração é necessária para garantir a variedade nas soluções construídas. O parâmetro de entrada α é utilizado para garantir aleatoriedade na etapa de construção. O parâmetro α é definido de modo que $0 \leq \alpha \leq 1$. Quanto maior α mais aleatória será a etapa de construção, porém quanto menor for α, mais gulosa será a etapa de construção de soluções.

O processamento da heurística GRASP é concluído com base em um critério de parada. Assim, a melhor solução encontrada durante todas as iterações da heurística GRASP é retornada como resultado final. No método GRASP, os parâmetros a serem definidos são o grau de aleatoriedade (α) e o critério de parada.

Na maioria das implementações da literatura, o critério de parada é geralmente um número máximo de iterações. Outros critérios de parada podem ser um limite de tempo estabelecido (ou um número limite de iterações) em que o procedimento poderá ficar sem atualizar a melhor solução encontrada ou um tempo máximo de execução Rangel & Abreu (2000).

Um pseudocódigo da heurística GRASP é apresentado no Algoritmo 2. O algoritmo possui dois parâmetros de entrada, α (porcentagem de aleatoriedade na etapa de construção) e critério de parada. A cada iteração é verificado a necessidade de atualizar a melhor solução s^*.

Algoritmo 2 GRASP(α, Critério de Parada)

```
f(s*) ← ∞;
while (Critério de Parada não ser atendido) do
  s ← Etapa de Construção(α);
  s' ←Busca Local(s);
  if f(s') < f(s*) then
    s* ← s';
  end if
end while
retorne s*;
```

A seguir descreve-se o procedimento de construção da heurística GRASP implementada neste trabalho.

3.3.1 Construção de soluções

A etapa de construção da heurística GRASP é um processo iterativo, que retorna uma solução viável. Iniciando com uma solução vazia, s, a cada iteração da etapa de construção é inserida uma tarefa de uma lista de candidatos (LC). Esta lista de candidatos é composta por todas as tarefas ainda não inseridas em s. A cada iteração da etapa de construção, a LC é ordenada considerando o incremento que a tarefa atribui à função objetivo se ela for inserida em s. Este processo é considerado adaptativo, pois o incremento à função objetivo associada a cada tarefa é atualizada para refletir a inserção feita na iteração anterior.

As tarefas de LC são ordenadas em ordem crescente em relação ao menor incremento no valor da função objetivo de s. Este incremento da tarefa j é obtido da seguinte maneira:

- Para cada máquina k em s faça:
  - Temporariamente insira a tarefa j como última tarefa da máquina k de s.
  - Calcule os tempos ótimos de início e conclusão das tarefas de s.
  - Calcule a função objetivo de s.
- seja o incremento da tarefa j o valor da função objetivo.

Este modo de ordenar tarefas também é usado na heurística construtiva DJASA (*Dynamic Job Assignment with Setups Resource Assignment*) Ruiz & C. (2007). A heurística DJASA é puramente gulosa. A cada iteração ela sempre adiciona a primeira tarefa da lista LC. No algoritmo de construção implementado neste trabalho a tarefa adicionada é escolhida aleatoriamente de uma lista restrita de candidatos (LRC). O tamanho da LRC, h, depende do valor do parâmetro de aleatoriedade α da heurística GRASP. A LRC é composta pelas primeiras h tarefas da LC, onde $h = max(1, \alpha \times |LC|)$. Se $\alpha = 0$, a LRC será composta apenas pela primeira tarefa de LC, tornando a etapa de construção um procedimento completamente guloso. No entanto, se $\alpha = 1$, a LRC será igual à LC tornando a etapa de construção completamente aleatória.

Ao fazer a inserção de uma tarefa na solução em construção, a lista LC deve ser reordenada. Este processo é repetido até todas as tarefas de LC serem inseridas na solução, resultando numa solução completa. O pseudocódigo da etapa de construção da heurística GRASP é apresentado no Algoritmo 3.

Algoritmo 3 Etapa de de construção(α)

$LC \leftarrow \{j_1, ..., j_n\}$ // Lista de candidatos;
$s \leftarrow \emptyset$;
while ($|LC| > 0$) **do**
  avaliar candidatos;
  ordenar (LC);
  $h \leftarrow max(1, \alpha \times |LC|)$;
  $i \leftarrow$Random$(1, h)$;
  $s \leftarrow Inserir(j_i, s)$;
  $LC \leftarrow LC - \{j_i\}$;
end while
retorne s;

3.3.2 Busca Local

A heurística GRASP e todas as heurísticas hibridas implementadas neste trabalho são processos iterativos, que utilizam o procedimento de busca local para o melhoramento de uma determinada solução. Este procedimento objetiva o melhoramento de uma solução s através da exploração de soluções semelhantes a ela, denominadas de soluções vizinhas. Estas soluções vizinhas são obtidas através de alterações ou pertubações da solução s.

Neste trabalho, as soluções vizinhas de uma solução são geradas realizando movimentos de inserção (ou realocação) de tarefas. Este método consiste em remover tarefas de suas posições originais e reinseri-las em outras posições, na mesma ou em outra máquina. Este método produz um conjunto de soluções vizinhas que possui estrutura denotada por N^R. A escolha dessa estrutura é devido ao trabalho de Ruiz et al. (2006), que destaca a estrutura de soluções vizinhas N^R como superior em relação a outras estruturas, como a estrutura N^T que utiliza troca de tarefas, em diversos problemas de sequenciamento de tarefas em máquinas paralelas.

Este procedimento de realocação de tarefas, quando aplicado a uma solução s, gera o conjunto de soluções vizinhas N_s^R. O tamanho do conjunto de soluções vizinhas $|N_s^R|$, se m for par, é de $(n^2 - n + m)$; no entanto, se m for ímpar é de $(n^2 - m)$. Por exemplo, para a solução $s = [[2, 5, 1, 4], [6, 11, 9, 10], [8, 7, 12, 3]]$, a solução $s' = [[2, 1, 4, 5], [6, 11, 9, 10], [8, 7, 12, 3]]$ e a solução $s'' = [[2, 1, 5], [6, 4, 11, 9, 10], [8, 7, 12, 3]]$, são consideradas duas soluções vizinhas. A solução s' é obtida inserindo a tarefa 5 na última posição da primeira subsequência (máquina 1), já para a solução s'' a obtenção se faz pela inserção da tarefa 4 na segunda posição da segunda subsequência (máquina 2). Este exemplo é ilustrado na Figura 3.9.

Além da estrutura N^R, o procedimento de busca local implementado utiliza o

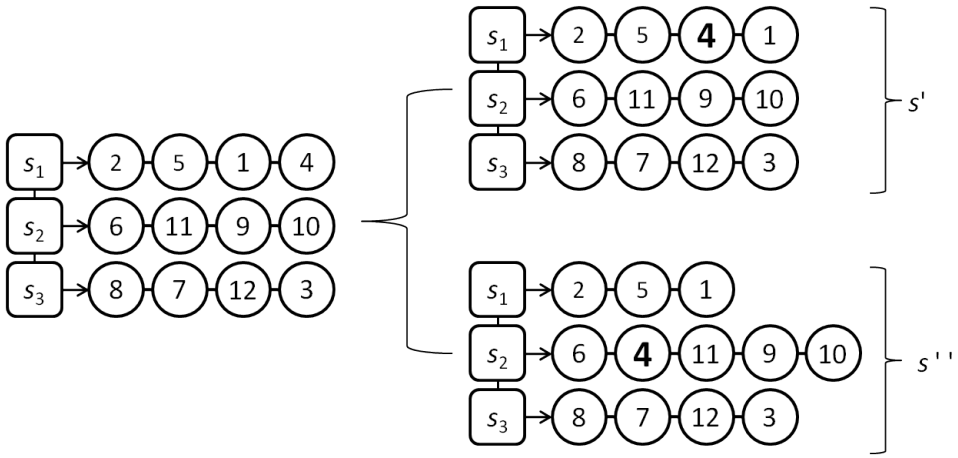


Figura 3.9: Exemplo da iteração de soluções utilizando movimentos de inserção

critério de escolha *best improvement*. Este critério analisa todas as possíveis soluções vizinhas de uma determinada solução s. Considerando s' como a solução de melhor qualidade entre as soluções vizinhas de s e a qualidade de s' sendo melhor que a de s ($f(s') < f(s)$), então o procedimento de busca local é aplicado em s', ou seja, ($s \leftarrow s'$). Lembrando que é necessário primeiramente calcular os tempos ótimos de início e conclusão de uma solução, para então ser possível definir sua qualidade com o uso da função 2.1 do modelo matemático. O procedimento é finalizado quando não é possível melhorar a solução atual s, ou seja, quando a de s é um ótimo local (melhor solução de um procedimento de busca local).

O Algoritmo 4 apresenta o pseudocódigo do procedimento de busca local implementado neste trabalho. Este algoritmo analisa cada solução de uma vizinhança, considerando a estrutura N_s^R, gerada a partir de uma solução s. Seja s'' a solução com melhor qualidade entre as soluções da vizinhança N_s^R, se a qualidade de s'' for melhor que s, ($f(s'') < f(s)$), então a busca local é aplicada em s''. Caso contrário o procedimento é finalizado, retornando a solução s.

3.4 Heurística ILS

A heurística *Iterated Local Search* (ILS) apresentada por Baxter (1990) pode ser aplicada em problemas de otimização combinatória, produzindo resultados satisfatórios Lourenço et al. (2002). Esta heurística começa com um procedimento de busca local em uma solução fornecida até obter uma solução ótima local inicial. A partir dessa solução, a heurística, iterativamente, constrói soluções geradas através da aplicação repetida de procedimentos de perturbação e busca local.

Algoritmo 4 Busca Local(s)

$f_{min} \leftarrow \infty$;
for (i até $|N_s^R|$) **do**
  $s' \leftarrow$ uma solução vizinha de s em $N^R(s)$ não analisada;
  if ($f(s') < f_{min}$) **then**
    $f_{min} \leftarrow f(s')$;
    $s'' \leftarrow s'$;
  end if
end for
if ($f(s'') < f(s)$) **then**
  $s \leftarrow$ Busca Local(s'');
end if
retorne s;

O propósito do procedimento de pertubação da solução ótima local inicial é permitir que a etapa de busca local seja feita em diferentes espaços de vizinhanças. Para que isso ocorra, o método de perturbação deve possuir uma intensidade suficiente para que o procedimento de busca local obtenha uma solução ótima local diferente. No entanto, a intensidade do procedimento de perturbação deve ser moderada para evitar um reinício aleatório do procedimento. Resumindo a intensidade da perturbação tem que ser suficiente para o procedimento de busca local obter uma solução ótima local diferente, porém não demasiadamente severa que torne o procedimento completamente estocástico.

A heurística ILS é composta de quatro etapas distintas:

1. Obtenção de uma solução ótima local inicial s;

2. Perturbação da solução s, obtendo uma solução s';

3. Busca local em s', obtendo um ótimo local s'';

4. Critério de aceitação;

As três últimas etapas são executadas iterativamente até que uma condição de parada pré-estabelecida seja atendida, resultando no retorno da solução s^*.

Neste trabalho a heurística ILS é utilizada na etapa de busca local da heurística GRASP. A condição de parada utilizada na heurística ILS consiste de um número de iterações sem melhoramento de s^*, assim como no trabalho de Ribeiro & Urrutia (2007). O pseudocódigo da heurística ILS implementada é apresentado no Algoritmo 5.

Algoritmo 5 <ILS(s, d)>

```
s* ← busca local(s);
while (condição de parada não satisfeita) do
  s' ← perturbação(s*, d) ;
  s'' ← busca local(s') ;
  // critério de aceitação
  if f(s'') < f(s*) then
    s* ← s'' ;
  end if
end while
retorne s*;
```

A heurística ILS possui como parâmetros de entrada a solução inicial s e um parâmetro d que determina a intensidade da etapa de perturbação. Neste trabalho a intensidade da etapa de perturbação é igual à quantidade de tarefas perturbadas.

A primeira etapa da heurística ILS é a obtenção de uma solução ótima local s^*, aplicando uma busca local na solução inicial s. Normalmente a solução s é construída pela heurística; no entanto, na implementação deste trabalho esta solução é fornecida como um parâmetro de entrada.

A etapa seguinte da heurística é o procedimento de perturbação da solução resultante da etapa anterior. O procedimento de perturbação consiste na modificação de s^*. Esse procedimento é composto por dois estágios distintos: destruição e reconstrução da solução.

O estágio de destruição consiste em remover aleatoriamente d tarefas da solução s^*, o que resulta na solução parcial s_p, que possui $n - d$ tarefas. As tarefas removidas são alocadas na ordem em que forem retiradas, em uma subsequência denominada de s_R, que possui tamanho d.

Após a remoção das tarefas é necessário reconstruir a solução parcial s_p. Esta etapa é constituída de d iterações nas quais os elementos de s_R serão reinseridos em s_p. Para cada iteração, a primeira tarefa de s_R é inserida em todas as posições possíveis de s_p, gerando $(n + m - |s_R| + 1)$ soluções parciais. Cada solução parcial é avaliada e a melhor é selecionada. O procedimento é encerrado após d iterações, ou seja, quando todas as tarefas de s_R forem inseridas em s_p. O procedimento de perturbação é apresentado no Algoritmo 6.

A solução s_p retornada pela perturbação é melhorada pelo método de busca local obtendo a solução s''.

A próxima etapa da heurística ILS é o critério de aceitação que tem como propósito determinar se a solução retornada pela busca local após a etapa de per-

Algoritmo 6 <perturbação(s, d)>

$s_R \leftarrow \emptyset$;
$s_p \leftarrow s$;
//estágio de destruição;
while $|s_R| < d$ **do**
  $j \leftarrow$ tarefa escolhida aleatoriamente de s_p ;
  $s_R \leftarrow s_R + j$;
  $s_p \leftarrow s_R - j$;
end while
//estágio de construção;
while $|s_R| > 0$ **do**
  Insira a tarefa $s_R[1]$ em todas as posições de s_p e determine a melhor posição i;
  $s_p \leftarrow$ inserir $s_R[1]$ na posição i; // insere o primeiro elemento de s_R na melhor posição de s_p
  $s_R \leftarrow s_R - s_R[1]$; // remove o primeiro elemento de s_R
end while
retorne s_p;

turbação será aceita. O critério de aceitação utilizado consiste em aceitar o ótimo local s'' se ele possuir um valor da função objetivo melhor do que valor da solução s^*, ou seja, $f(s'') < f(s^*)$.

3.5 Path Relinking

A técnica de *Path Relinking* (PR), originalmente introduzido por Glover (1996), é uma estratégia de intensificação que explora regiões potencialmente promissoras entre duas soluções. A partir de uma determinada solução, um caminho é gerado em direção à outra solução em busca de obter um melhoramento. Para gerar este caminho, são utilizados movimentos (troca ou inserção) para introduzir na solução origem, elementos presentes na solução guia. Os elementos a serem introduzidos podem ser a posição da tarefa ou a quantidades de tarefas em uma sequência.

O procedimento PR necessita de duas soluções distintas como parâmetros de entrada: a solução origem s_o e a solução guia s_g. O caminho entre s_o e s_g é gerado a partir dos movimentos de troca e inserção que introduzem elementos presentes da s_g na s_o, ou seja, a solução s_o é transformada gradativamente em s_g. Inicialmente é calculado o número de elementos diferentes entre s_o e s_g, denotado por $\Delta(s_o, s_g)$. Este valor é igual ao número de tarefas de s_o que estão em posições diferentes em s_g. Por exemplo, usando as soluções $s_o = [[9, 6, 7, 8, 3], [12, 1, 10, 4], [2, 5, 11]]$ e $s_g = [[9, 6, 2, 8], [3, 12, 10, 1, 4], [7, 5, 11]]$, o valor de $\Delta(s_o, s_g) = 6$, pois s_o possui 6 tarefas em

posições diferentes quando comparados a s_g, estas tarefas são $7, 3, 12, 1, 4, 2$. Como o valor $\Delta(s_o, s_g)$ representa o número de elementos diferente entre s_o e s_g, ele será igual ao número de passos necessários para criar o caminho entre s_o a s_g.

Cada passo do procedimento PR constrói um conjunto de soluções semelhantes à s_o. Este conjunto é denominado de vizinhança restrita de s_o e é denotado por $N_g(s_o)$. As soluções $N_g(s_o)$ são construídas utilizado movimentos diferentes de troca ou inserção de tarefas de s_o.

O movimento de troca é utilizado para aproximar a sequência da solução origem à sequência da solução guia. Este movimento é feito pela troca de uma tarefa de s_o com a tarefa na posição que contém o mesmo valor em s_g. Por exemplo, em s_o, a tarefa 12 está na primeira posição da segunda máquina. No entanto, em s_g, a tarefa 12 esta na segunda posição da segunda máquina. Em s_o, a segunda posição da segunda máquina contém a tarefa 1. O movimento troca a tarefa 12 com a tarefa 1 de s_o gerando uma nova solução $s = [[9, 6, 7, 8, 3], [1, 12, 10, 4], [2, 5, 11]]$. Uma vez que o movimento troca a posição de duas tarefas, é possível que, com um movimento as duas tarefas fiquem na posição correta em relação à s_g. Esta situação acontece se o movimento de troca for feito na tarefa 7 e na tarefa 2. Em tal caso, a vizinhança restrita irá possuir duas soluções com sequências de tarefas idênticas.

O movimento de inserção é feito para que o número de tarefas em cada máquina da solução origem seja igual a da solução guia. Este movimento remove a última tarefa da máquina de s_o que possui um quantidade maior de tarefas comparado a sua respectiva máquina em s_g. A tarefa removida é inserida na última posição da máquina de s_o que possui uma quantidade menor de tarefas comparado a sua respectiva máquina em s_g. Utilizando o exemplo, a primeira máquina de s_o possui 5 tarefas ($[9, 6, 7, 8, 3]$) e a primeira máquina de s_g possui 4 tarefas ($[9, 6, 2, 8]$), ou seja, a última tarefa da primeira máquina (3) será inserida em outra máquina. Como a segunda máquina de s_o possui 4 tarefa ($[12, 1, 10, 4]$) e a segunda máquina de s_g contem 5 tarefas ($[3, 12, 10, 1, 4]$) então a tarefa removida (3) será inserida na última posição da segunda máquina de s_o resultando na solução $s = [[9, 6, 7, 8], [12, 1, 10, 4, 3], [2, 5, 11]]$.

Após a construção de $N_g(s_o)$ utilizando todos os possíveis movimentos, uma nova solução origem será escolhida. A solução da vizinhança restrita que possui o maior melhoramento do valor da função objetivo (ou a menor piora) será a nova solução origem.

A Figura 3.10 ilustra o primeiro passo do procedimento PR utilizando os dados do exemplo apresentado na secção 3.2.2. Neste exemplo, a solução origem possui a sequência de tarefas $s_o = [[9, 6, 7, 8, 3], [12, 1, 10, 4], [2, 5, 11]]$ com o valor da função objetivo $f = 1761$ e a solução guia possui a sequência $s_g = [[9, 6, 2, 8], [3, 12, 10, 1, 4], [7, 5, 11]]$

com o valor de $f = 1293$. Observa-se que na Figura a solução gerada pelo movimento de troca da tarefa 2 não esta presente. Tal fato se deve ao movimento de troca da tarefa 2 que gera uma solução idêntica aquela produzida pelo movimento de troca da tarefa 7. Nessa Figura, a solução com a melhor qualidade encontrada é a segunda solução da vizinhança restrita $s = [[9, 6, 7, 8, 12], [3, 1, 10, 4], [2, 5, 11]]$, obtido pelo movimento de troca da tarefa 3, na qual possui o valor da função objetivo de $f = 1396$. Esta solução será a nova solução origem para o próximo passo do procedimento.

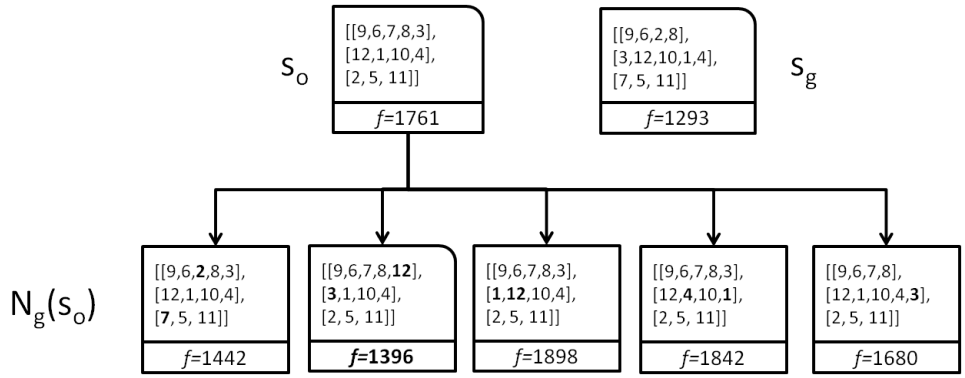


Figura 3.10: Exemplo da estratégia *Path Relinking* do primeiro passo em uma instância com $n = 12$ e $m = 3$

O procedimento é repetido até haver apenas uma diferença entre os elementos de s_o e s_g, ou seja, se mais um movimento for feito, a solução origem e a solução guia serão idênticas ($\Delta(s_o, s_g) = 1$). Após o encerramento do procedimento, a solução com a melhor qualidade encontrada durante todo o procedimento é retornada. A Figura 3.11 ilustra por completo o procedimento PR utilizando o exemplo abordado previamente.

Existem várias formas de implementar o *Path Relinking*, tais como:

- *forward relinking*: o caminho é construído de uma solução s_o, que possui a pior qualidade, em direção a s_g;
- *backward relinking*: o caminho é construído de uma solução s_o, na qual possui a melhor qualidade, em direção a s_g;
- *bidirecional relinking*: dois caminhos são construídos, sendo o primeiro de s_o em direção a s_g e outro de s_g em direção a s_o;
- *mixed relinking*: dois caminhos são explorados simultaneamente originados da s_o e da s_g, e se encontram em uma solução intermediária;

Uma descrição mais aprofundada sobre as diversas estratégias do *Path Relinking* é apresentada no trabalho de Resende & Ribeiro (2005).

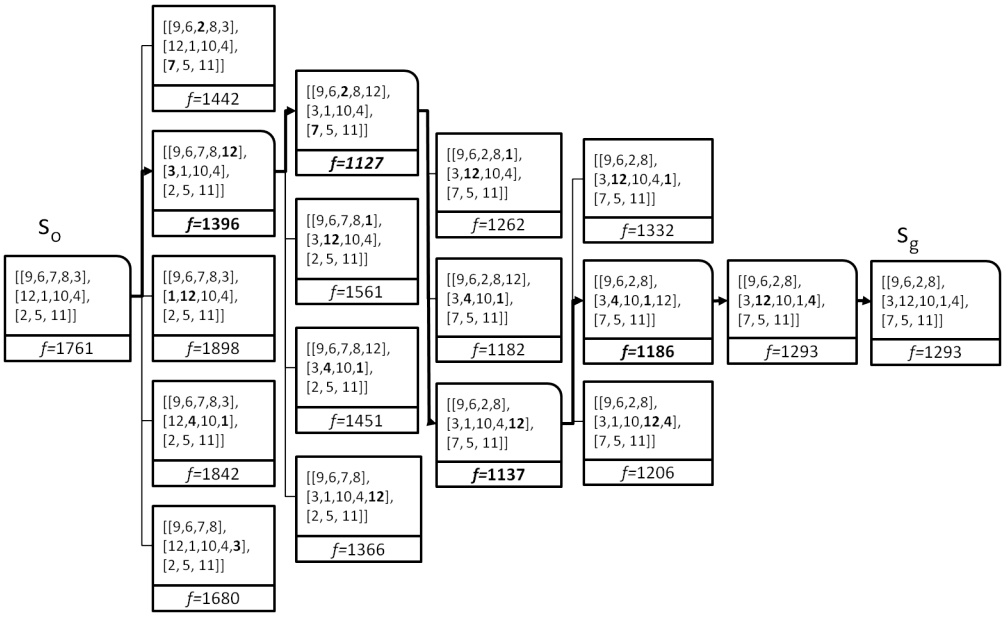


Figura 3.11: Exemplo da estratégia *Path Relinking* em uma instância com $n = 12$ e $m = 3$

Nesta dissertação foi implementada a estratégia de *mixed relinking*, a qual foi proposta por Ribeiro & Rosseti (2004). Esta estratégia gera caminhos derivados de ambas as soluções, origem e guia, que se encontram formando um único caminho contínuo. Estes caminhos são gerados através do intercâmbio entre as soluções s_o e s_g a cada passo do procedimento. Quando os dois caminhos se encontram, a melhor solução encontrada pelo procedimento é retornada. Esta estratégia foi utilizada, pois ela explora as vizinhanças restritas perto de s_o e s_g assim como a estratégia *back-and-forward relinking*. No entanto, como *mixed relinking* só constrói os caminhos simultaneamente, ela gasta menos tempo. A Figura 3.12 ilustra o procedimento de *Mixed Path Relinking* entre a solução origem e a solução guia utilizados no exemplo anterior. Nota-se que os dois caminhos se encontram na solução $s = [[9, 6, 2, 8, 4], [3, 1, 10, 12], [7, 5, 11]]$ que possui o valor da função objetivo $f = 1073$.

3.6 Heurísticas Híbridas

Utilizando os três métodos apresentados (GRASP, ILS e PR) foram implementadas heurísticas híbridas com objetivo de resolver o problema de sequenciamento em máquinas paralelas não relacionadas com tempo de preparação e penalidades por atraso e adiantamento. A combinação dos métodos produziu três heurísticas híbridas: GRASP+ILS, GRASP+PR e GRASP+ILS+PR. O procedimento de cada método

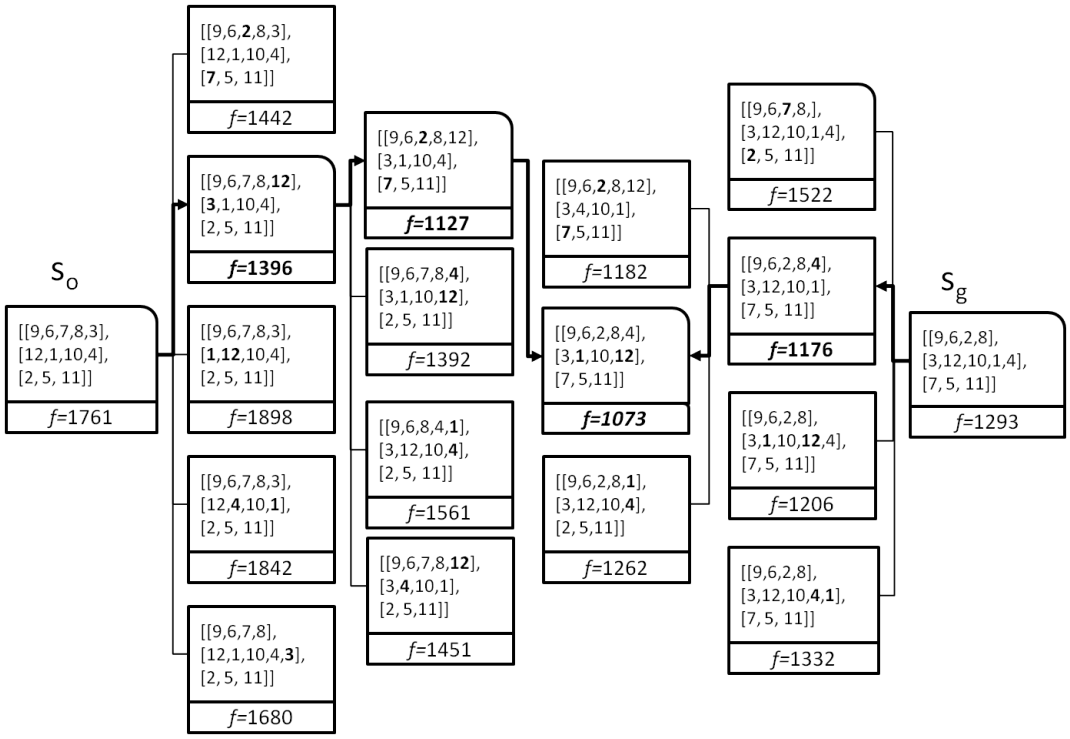


Figura 3.12: Exemplo da estratégia *Mixed Path Relinking* em uma instância com $n = 12$ e $m = 3$

híbrido é detalhado nas subseções seguintes.

3.6.1 GRASP com ILS

Neste método a heurística GRASP é utilizada em conjunto com a heurística ILS, formando um procedimento híbrido denominado de GRASP+ILS. Esta hibridização é feita semelhantemente ao trabalho apresentado por Resende (2008). A hibridização dos dois métodos é feita através da substituição da etapa de busca local do GRASP pela heurística ILS. Nessa heurística híbrida a solução resultante da etapa de construção do GRASP é fornecida como solução inicial do procedimento ILS. A condição de parada para o procedimento ILS é a mesma utilizada no trabalho de Ribeiro & Urrutia (2007), ou seja, 50 iterações do método ILS sem melhoramento da melhor solução atual.

3.6.2 GRASP com Path Relinking

Neste método é desenvolvida uma heurística híbrida GRASP que utiliza a técnica de PR como um procedimento de intensificação. A combinação das heurísticas GRASP com PR foi proposta por Laguna & Martí (1999).

Neste trabalho, a implementação da heurística híbrida que utiliza GRASP com a estratégia *Mixed Path Relinking* é denominada de GRASP+PR. Esta hibridização é baseada no trabalho de Aiex & Resende (2006). A implementação da heurística híbrida utiliza como parâmetros de entrada o valor α, utilizado na etapa de construção do GRASP, e o tamanho do conjunto de soluções elite E (soluções de alta qualidade) utilizado pela técnica PR, denotado de E_{size}. O procedimento de PR é aplicado em um par de soluções (s_o, s_g), no qual s_o é um ótimo local resultante da etapa de busca local do GRASP e s_g é uma solução escolhida aleatoriamente do conjunto de soluções Elites E obtidas durante a execução da heurística.

O procedimento de escolha da solução s_g do conjunto de soluções elite E é implementado baseado no trabalho de Resende & Werneck (2004). A estratégia utilizada favorece a escolha de soluções do conjunto de elite E que, durante o procedimento de *Path Relinking*, possuem um caminho mais comprido entre s_o e s_g, ou seja, favorece soluções com uma maior diferença simétrica a s_o.

A razão pelo uso de E é conservar um conjunto de soluções com boa qualidade e diversidade para ser utilizado no procedimento. Inicialmente este conjunto está vazio. Cada solução obtida pela etapa de busca local do método GRASP é considerada como um candidato a ser inserido no conjunto E se não estiver presente nele ($s \notin E$). Se a quantidade de soluções no conjunto E for menor que E_{size} ($|E| < E_{size}$) a solução será inserida em E. No entanto, se a quantidade de soluções do conjunto E for igual à E_{size} ($|E| = E_{size}$) então a solução a ser inserida em E irá somente substituir a solução do conjunto mais similar que possuir uma qualidade pior que a dela. Esta estratégia é utilizada para manter a diversidade do conjunto E durante a execução da heurística hibrida GRASP+PR.

Assim sendo, o procedimento PR é executado a partir da segunda iteração da heurística híbrida, GRASP+PR. Isto se deve ao fato de que o conjunto E deve conter pelo menos um elemento para que seja escolhida uma solução para o procedimento de PR. O pseudocódigo da heurística híbrida GRASP+PR é apresentado no Algoritmo 7.

3.6.3 GRASP com ILS e Path Relinking

Neste método é feita a hibridização da heurística GRASP+PR com a heurística ILS mencionada neste capítulo. Esta heurística híbrida é denotada por GRASP+ILS+PR. O método ILS é introduzido na heurística GRASP+PR através da substituição da etapa de busca local do GRASP pela heurística ILS na forma como foi abordado na heurística híbrida GRASP+ILS. Este procedimento utiliza os mesmos parâmetros de ambas as heurísticas híbridas mencionadas (α, d, E_{size}). O pseudocódigo da heurística

Algoritmo 7 GRASP+PR(α, Critério-de-Parada)

```
E ← ∅;
f(s*) ← ∞;
i ← 1;
while Critério-de-Parada do
   s ← Construção(α);
   s' ← Busca Local(s);
   Atualizar conjunto Elite(E, s');
   if i ≥ 2 then
      s_g ← escolha aleatoriamente uma solução de E;
      s' ← Mixed − PathRelinking(s', s_g);
      Atualizar conjunto Elite(E, s');
   end if
   if f(s') < f(s*) then
      f(s*) ← f(s');
      s* ← s';
   end if
   i ← i + 1;
end while
retorne s*;
```

híbrida GRASP+ILS+PR é apresentado no Algoritmo 8.

Algoritmo 8 GRASP+ILS+PR(α, d, Critério de Parada)

$E \leftarrow \emptyset$;
$f(s^*) \leftarrow \infty$;
$i \leftarrow 1$;
while (Critério de Parada não satisfeito) **do**
  $s \leftarrow$ Construção(α);
  $s' \leftarrow$ ILS(s, d);
  Atualizar conjunto Elite(E, s');
  if $i \geq 2$ **then**
    $s_g \leftarrow$ escolha aleatoriamente uma solução de E;
    $s' \leftarrow Mixed - PathRelinking(s', s_g)$;
    Atualizar conjunto Elite(E, s');
  end if
  if $f(s') < f(s^*)$ **then**
    $f(s^*) \leftarrow f(s')$;
    $s^* \leftarrow s'$;
  end if
  $i \leftarrow i + 1$;
end while
retorne s^*;

Capítulo 4

EXPERIMENTOS COMPUTACIONAIS

Neste capítulo é apresentado o ambiente computacional e o procedimento de geração de instâncias dos problemas teste. Em seguida é descrita a metodologia de avaliação dos resultados obtidos pelos métodos propostos e o critério de parada utilizado. Continuando, é descrito como foi realizada a calibração dos parâmetros de entrada de cada heurística. Por fim, são apresentados os resultados e a comparação das médias do desvio percentual relativo (RPD) do valor da função objetivo das heurísticas implementadas.

4.1 Ambiente Computacional e Geração de Instâncias de Problemas Teste

Todos os procedimentos heurísticos foram implementados na linguagem de programação Java versão 1.6, e compilados com IDE Eclipse 3.5.1 em uma máquina que utiliza o sistema operacional Windows 7 v.64. Os testes foram executados usando o compilador JDK 6.0 em um computador com processador Intel Core2 Quad CPU Q9550 @ 2.83GHz, com 6 GB de RAM. Todos os testes foram executados sem a utilização de multiprocessamento, ou seja, todos os testes utilizaram apenas um processador.

As instâncias dos problemas de sequenciamento foram geradas de forma semelhante às de Lee & Choi (1997). Ao todo foram geradas 400 instâncias com tamanhos variados, que são agrupadas em 3 conjuntos ordenados pelo número de tarefas do problema. Estes conjuntos de instâncias são denotados por: pequeno, médio e grande porte. O conjunto de pequeno porte é dividido em 6 grupos de instâncias tendo cada grupo 10 instâncias perfazendo um total de 60. As instâncias

terão a combinação de número de tarefas e de número de máquinas definido por $(n = \{8, 9, 10\}) \times (m = \{3, 5\})$. O conjunto de instâncias de médio porte é composto de quatro grupos com 10 instâncias em cada grupo sendo a relação de tarefas e máquinas seguindo a proporção $n \times m$ onde $(n = \{20, 30\}) \times (m = \{3, 5\})$. O conjunto de instâncias de grande porte é composto de 30 grupos com 10 instâncias em cada grupo sendo a relação de tarefas e máquinas seguindo a proporção $(n = \{50, 60, 70, 80, 90, 100\}) \times (m = \{10, 15, 20, 25, 30\})$. A Tabela 4.1 mostra as possíveis combinações do número de tarefas e máquinas utilizadas para gerar os grupos de instâncias de cada conjunto.

Tabela 4.1: Possíveis combinações nos valores n e m

Conjunto	**Grupos**	$\boldsymbol{n}$	$\boldsymbol{m}$	**Instâncias**
Pequeno porte	6	8, 9, 10	3, 5	60
Médio porte	4	20, 30	3, 5	40
Grande porte	30	50,60,70,80,90,100	10,15,20,25,30	300

Para gerar as instâncias dos problemas teste são necessários três parâmetros: o fator de dispersão das datas de entrega R, o fator de aperto das datas de entrega τ e o rigor dos tempos de preparação η. Todas as instâncias foram geradas utilizando-se $\tau = 0,3$, $R = 0,25$ e $\eta = 0,25$, assim como encontrado em Lee & Choi (1997). O tempo de processamento da tarefa i na máquina k, $(p_{i,k})$, é escolhido aleatoriamente do intervalo $[50, 100]$. As penalidades por atraso α_i e adiantamento β_i são escolhidas aleatoriamente no intervalo $[1, 100]$. Os tempos de preparação são gerados no intervalo $\left[\frac{2}{3}\eta\overline{p}, \frac{4}{3}\eta\overline{p}\right]$, sendo , $\overline{p}$ a média dos tempos de processamento. Para gerar as datas de entrega é necessário calcular o intervalo no qual estas serão geradas. Para tal é necessário obter uma estimativa do *makespam* C_{max}, que é obtida pela seguinte fórmula:

$$C_{max} = \frac{n}{m}\left(\overline{p} + \overline{s}\left(0,4 + \frac{10m^2}{n^2} - \frac{\eta}{7}\right)\right)$$

sendo o valor $\overline{s}$ a média dos tempos de preparação. A mediana da data de entrega $\overline{d}$ é calculada da seguinte forma:

$$\overline{d} = C_{max}(1 - \tau)$$

As datas de entrega estão distribuídas uniformemente no intervalo $[(1 - R)\overline{d}, \overline{d}]$ com probabilidade τ e no intervalo $[\overline{d}, ((C_{max} - \overline{d})R) + \overline{d}]$ com probabilidade $(1 - \tau)$.

4.2 Metodologia de Avaliação dos Métodos Propostos

Para avaliar a eficiência dos resultados das heurísticas implementadas é computada a média do desvio relativo percentual (*Relative Percentual Deviation*) (RPD) como apresentado no trabalho de Vallada & Ruiz (2009). O RPD é calculado pela seguinte fórmula:

$$RPD = \frac{Metodo_{sol} - Melhor_{sol}}{Melhor_{sol}} \times 100$$

sendo, $Metodo_{sol}$ o valor da função objetivo da solução obtida por um dos métodos e $Melhor_{sol}$ é o valor da função objetivo da melhor solução encontrada entre todos os métodos comparados. Sendo assim, o valor do RPD para um método definirá a sua qualidade em relação a melhor solução obtida. Quanto menor for este valor, melhor será a qualidade do método.

O RPD de um método é calculado utilizando a média do valor da função objetivo de 5 execuções desse método em cada instância de um conjunto.

4.3 Critério de Parada

As heurísticas apresentadas neste trabalho possuem a característica comum de serem procedimentos iterativos, ou seja, são executados várias vezes até que o critério da condição de parada seja satisfeito. Todas heurísticas implementadas utilizam a condição de parada baseada em um tempo máximo de execução. Este critério define o tempo limite no qual o procedimento será executado. Este critério de parada foi escolhido com objetivo de comparar os resultados das heurísticas implementadas dentro do mesmo intervalo temporal. Este tempo limite é definido de forma que varia de acordo com o número de tarefas e máquinas de um problema de sequenciamento.

O método utilizado para calcular o tempo limite de um problema é semelhante ao encontrado no trabalho de Vallada & Ruiz (2009). O tempo limite utilizado em cada instância consiste de $(n \times m \times 50)$ milissegundos, de forma que um problema com 50 tarefas e 20 máquinas é executado em 50 segundos (50,000 milissegundos).

4.4 Análise dos Parâmetros das Heurísticas Implementadas

As heurísticas implementadas nesta dissertação utilizam alguns parâmetros de entrada que afetam o desempenho dos seus procedimentos. Estes parâmetros foram submetidos

a um processo de calibração, para definir o melhor valor a ser utilizado nos experimentos das heurísticas.

Nas subseções seguintes é apresentada a metodologia utilizada para a calibração desses parâmetros. Existem algumas situações nas quais os parâmetros utilizados são aqueles definidos de acordo com a literatura.

4.4.1 Calibração do parâmetro α do GRASP

A heurística GRASP possui apenas um parâmetro que necessita ser calibrado, que vem a ser relativo à aleatoriedade da etapa de construção denotado por α. Com objetivo de definir um valor de α que forneça bons resultados, é feita uma calibração utilizando todos os conjuntos de instâncias.

Para a calibração da heurística GRASP foram escolhidos 4 valores para o parâmetro α. Estes valores foram escolhidos levando em consideração que valores de α próximo a 0 tornam a etapa de construção da heurística GRASP um procedimento guloso e valores próximo de 1 tornam a etapa de construção um procedimento muito aleatório. Foram então adotados os valores de α = (0,1; 0,2; 0,3; 0,4) para serem testados.

Os resultados dos testes utilizando cada valor de α descrito no parágrafo anterior são comparados entre si pelos valores médios de RPD para cada conjunto de instâncias. Nesses testes, a condição de parada adotada foi de $n \times m \times 50$ milissegundos de execução, conforme mencionado anteriormente.

A Figura 4.1 apresenta a distribuição das médias de RPD segundo os valores testados do parâmetro α no conjunto de instâncias de pequeno porte. Analisando esta figura é observado que os testes com valores de $\alpha = 0,4$ possuem uma média de RPD em geral menor que nos testes com os demais valores. Na Figura 4.1 é também observado que os testes utilizando valores de $\alpha = 0,4$ obtiveram uma dispersão menor da média de RPD. Este resultado é consistente com o fato de que, no conjunto de instâncias de pequeno porte, os valores menores de α fazem com que a etapa de construção seja praticamente gulosa afetando a diversidade de soluções.

Os resultados da calibração de α para o conjunto de instâncias de médio e grande porte são ilustrados, respectivamente, nas Figuras 4.2 e 4.3.

Nas Figuras 4.2 e 4.3 observa-se, respectivamente, a distribuição das médias de RPD do valor da função objetivo para as instâncias de médio e grande porte. Nos *boxplots* apresentados nessas figuras, nota-se que o aumento do valor do parâmetro α resultou na situação exatamente oposta a ocorrida com as médias de RPD no conjunto de pequeno porte. Os testes com valores mais elevados de α resultam em um aumento

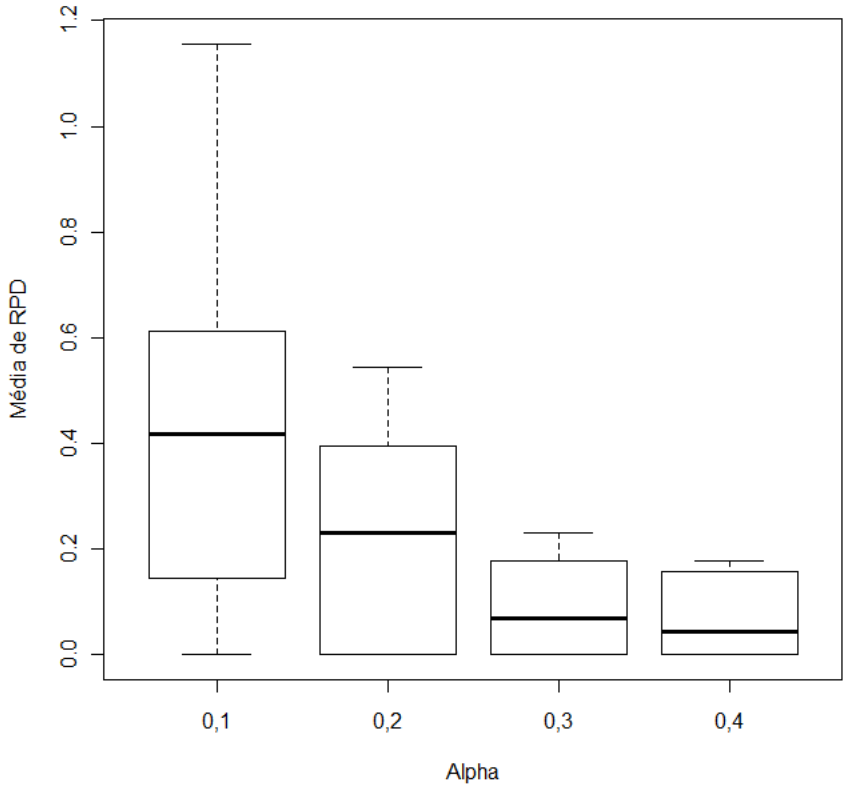


Figura 4.1: Boxplot da distribuição das médias de RPD segundo os valores de α do conjunto de pequeno porte

das médias de RPD. O procedimento GRASP para os conjuntos de médio porte apresenta soluções de melhor qualidade com a utilização de $\alpha = 0,1$. Esta situação também ocorreu nos testes utilizando instâncias do conjunto de grande porte, Figura 4.3. A dispersão das médias de RPD em ambos os conjuntos de instâncias (médio e grande porte) não sofreu alterações notáveis. A Tabela 4.2 apresenta os resultados dos testes de calibração do parâmetro α nos três conjuntos de instâncias.

Tabela 4.2: Média do RPD aos valores de α para cada conjunto de instâncias

Conjuntos	$\alpha = 0,1$	$\alpha = 0,2$	$\alpha = 0,3$	$\alpha = 0,4$
Pequeno	0,54	0,35	0,18	**0,11**
Médio	**1,60**	1,66	1,77	1,98
Grande	**4,07**	4,29	4,48	4,71

O parâmetro de entrada α do método GRASP foi calibrado e definido igual ao melhor valor apresentado na Tabela 4.2, ou seja, $\alpha = 0,4$ para experimentos que

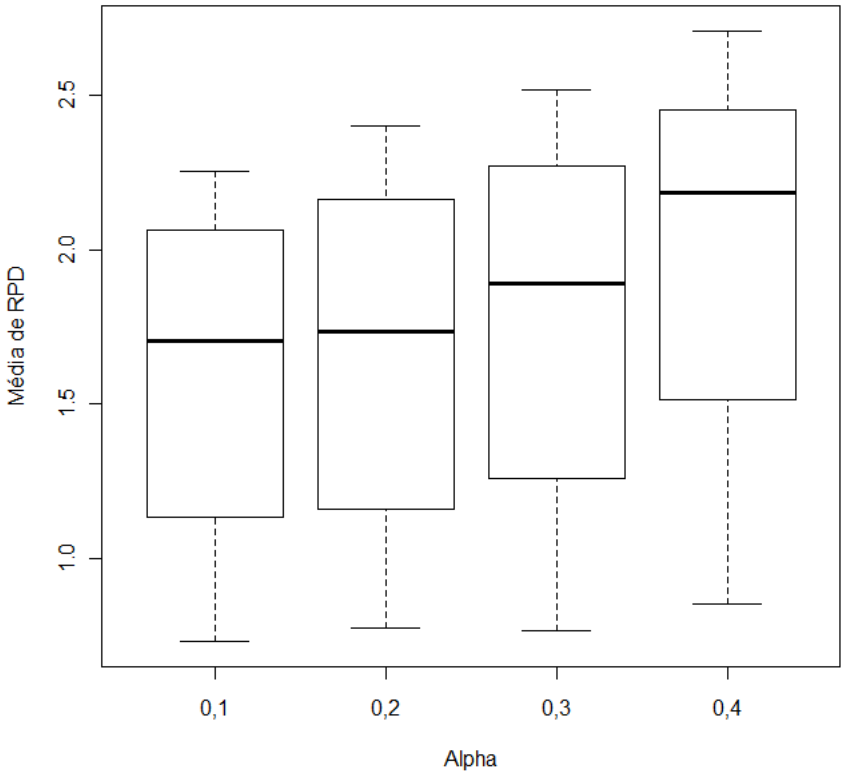


Figura 4.2: Boxplot da distribuição das médias de RPD segundo os valores de α do conjunto de médio porte

utilizam o conjunto de pequeno porte e $\alpha = 0,1$ para experimentos que utilizam o conjunto de médio e grande porte.

4.4.2 Calibração do parâmetro d do ILS

A implementação da heurística ILS utilizada nesta dissertação utiliza o parâmetro de entrada d, o qual necessita ser calibrado. Este parâmetro representa o grau de intensidade da etapa de perturbação da heurística. Essa etapa não deve ser suficientemente intensa para gerar um reinício aleatório da heurística, e nem moderada demais que impossibilite melhorar o ótimo local.

O procedimento ILS é utilizado nas heurísticas híbridas GRASP+ILS e GRASP+ILS+PR. No entanto, para calibrar este parâmetro, os testes foram executados na heurística mais simples entre as duas: GRASP+ILS. Todos os testes executados nesta heurística foram feitos utilizando o melhor valor de α encontrado na subseção anterior (conjunto de pequeno porte $\alpha = 0,4$; conjunto médio e grande porte

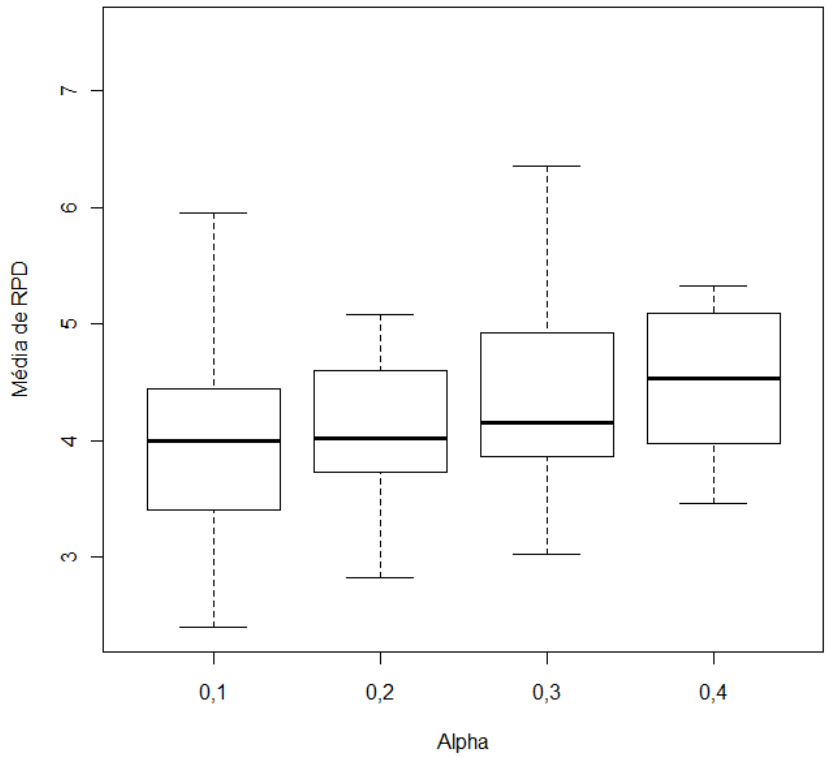


Figura 4.3: Boxplot da distribuição das médias de RPD segundo os valores de α do conjunto de grande porte

$\alpha = 0,1$). A condição de parada do ILS é a mesma mencionada no Capítulo 3, ou seja, de 50 iterações sem melhoramento da melhor solução achada. Os possíveis valores utilizados na calibração do parâmetro teste d em cada conjunto de instâncias foram $d = (4, 6, 8, 10)$ com a exceção do conjunto de pequeno porte. As instâncias de pequeno porte foram executadas com $d = 2$, pois valores acima deste tornaram os resultados da heurística muito aleatórios. As Figuras 4.4 e 4.5 apresentam, respectivamente, as médias do RPD para instâncias de médio e grande porte obtidas durante os testes de calibração de d.

Observa-se na Figura 4.4 que os testes realizados no conjunto de médio porte geralmente obtiveram a menor média de RPD utilizando $d = 4$. No entanto, os testes de calibração do parâmetro d feito nas instâncias de grande porte obtiveram a melhor média de RPD utilizando $d = 10$, assim como ilustrado na Figura 4.5. Nos teste utilizando o conjunto de grande porte, nota-se também que a menor distribuição de RPD foi obtida nos testes com $d = 10$. Também é interessante notar, nos testes utilizando o conjunto de médio porte, a menor mediana é obtida nos testes utilizando

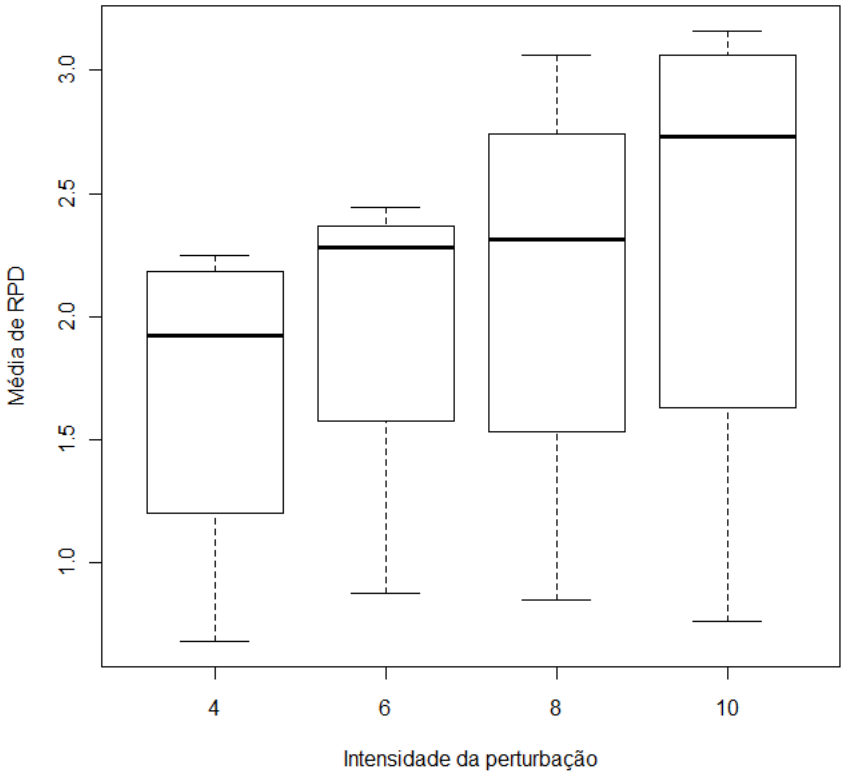


Figura 4.4: Boxplot das médias do RPD em relação a d do conjunto de médio porte

$d = 4$.

A Tabela 4.3 apresenta a média dos valores de RPD encontrados nos testes de calibração do parâmetro d para cada conjunto de instâncias. As médias do RPD do conjunto de instâncias apresentadas na Tabela 4.3 indicam que os testes realizados com $d = 10$ no conjunto de grande porte e $d = 4$ no conjunto de médio porte apresentaram uma média de RPD menor em relação aos testes utilizando outros valores. Por tal fato, os testes utilizando as heurísticas híbridas GRASP+ILS e GRASP+ILS+PR utilizaram $d = 4$ nas instâncias do conjunto de médio porte e $d = 10$ nas instâncias do conjunto de grande porte.

Tabela 4.3: Média do RPD em relação aos valores de d para cada conjunto de instâncias

Conjunto	$d = 4$	$d = 6$	$d = 8$	$d = 10$
Médio	**1,69**	1,97	2,14	2,35
Grande	5,60	5,27	4,49	**3,81**

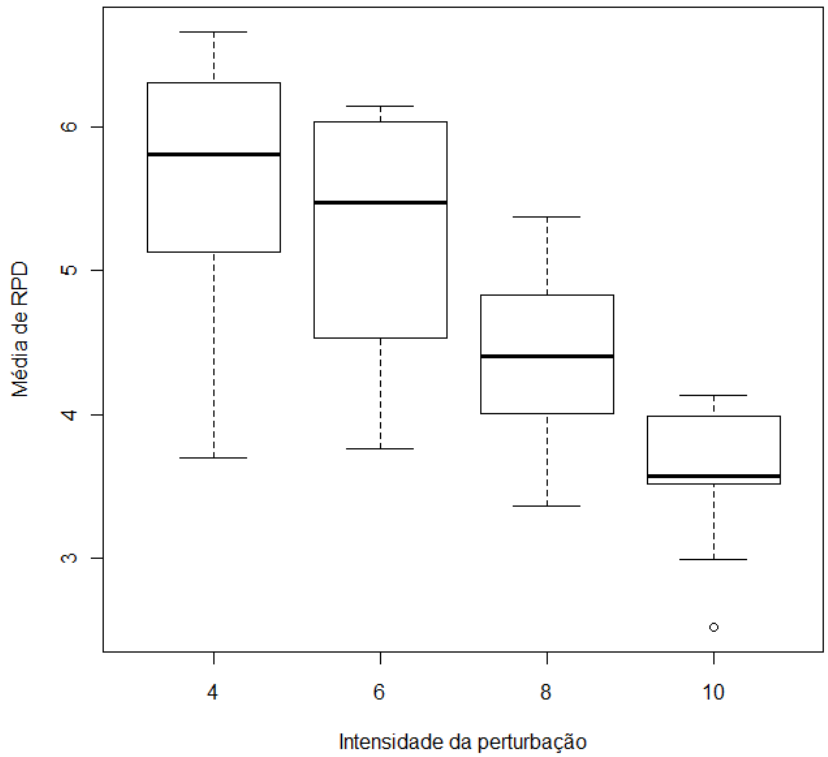


Figura 4.5: Boxplot das médias do RPD em relação a d do conjunto de grande porte

4.4.3 Calibração do parâmetro E_{size} do Path Relinking

A implementação da técnica de PR necessita apenas de um parâmetro de entrada, o tamanho do conjunto de soluções Elite E_{size}. Assim como os parâmetros α e d, o valor do parâmetro E_{size} foi calibrado utilizando-se os conjuntos de instâncias de pequeno, médio e grande porte. Os testes de calibração para o parâmetro do PR foram feitos utilizando a heurística híbrida GRASP+PR, sendo que, o parâmetro α utilizado para a etapa de construção método GRASP é o melhor valor obtido, apresentado na Seção 4.3.1. Durante os testes de calibração foram utilizados os valores $E_{size} = (5, 10, 15, 20)$. Os resultados dos testes utilizando cada valor de E_{size} são comparados entre si pelos valores médios de RPD para cada conjunto de instâncias. Lembra-se que a condição de parada adotada nesses testes é a mesma utilizada para a calibração de α e d, ou seja, cada instância do conjunto é executada por $m \times n \times 50$ milissegundos. A Figura 4.6 apresenta um *boxplot* da distribuição das médias de RPD obtidas pelos testes utilizando as instâncias de pequeno porte.

Nos resultados apresentados na Figura 4.6, os testes que utilizaram $E_{size} = 20$

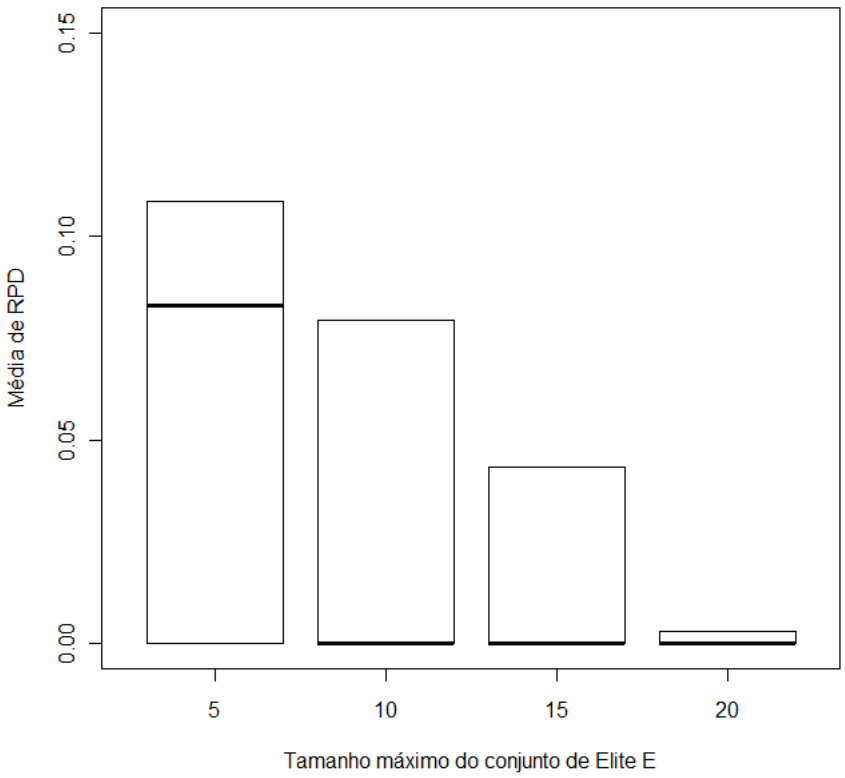


Figura 4.6: Boxplot das médias do RPD em relação a E_{size} do conjunto de pequeno porte

obtiveram os melhores resultados. Normalmente, nas instâncias de pequeno porte, a heurística GRASP+PR obtém a mesma solução obtida pelo *software* CPLEX, ou seja, a solução ótima. Consequentemente, na maioria das execuções a distribuição da média dos RPD obtida pelos testes é muito perto de zero. As Figuras 4.7 e 4.8 apresentam, respectivamente, os *boxplot* das distribuições das médias de RPD obtidas na calibração do parâmetro E_{size} nos conjuntos de médio e grande porte.

Analogamente aos resultados do conjunto de pequeno porte, os resultados do conjunto de médio e grande porte geralmente apresentam a melhor média de RPD nos testes utilizando $E_{size} = 20$. Nota-se que, nos conjuntos de médio e grande porte, a dispersão das médias de RPD é muito pouco afetada pela variação do parâmetro E_{size}. Os valores das médias de RPD de cada conjunto em relação ao parâmetro E_{size} são apresentados na Tabela 4.4.

Nos testes verificou-se que para todos os conjuntos de instâncias, o menor valor da média de RPD foi obtido em testes que possuem $E_{size} = 20$. Assim sendo, os

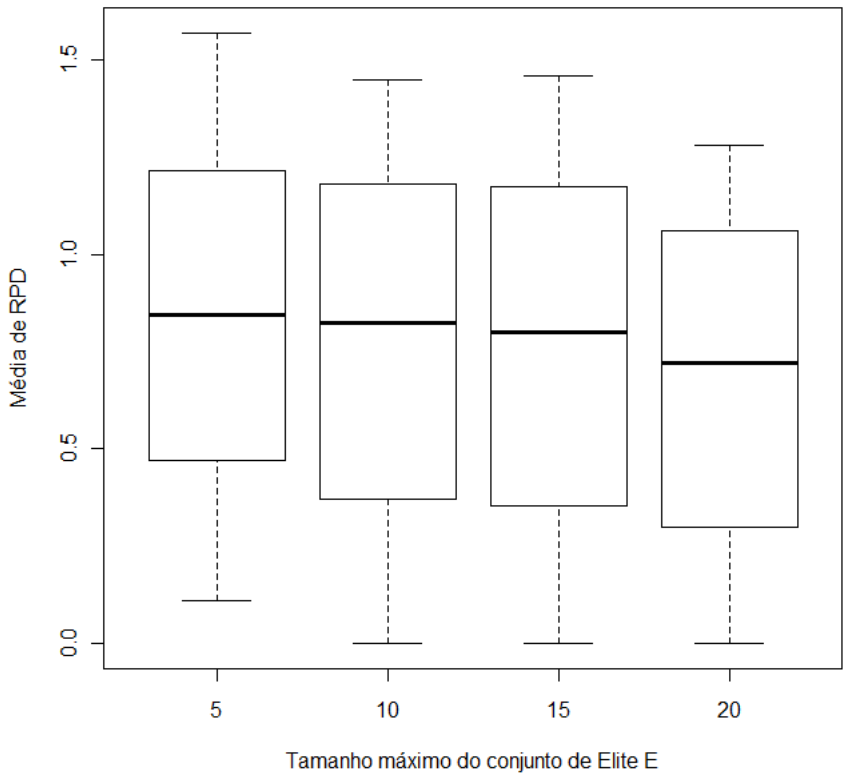


Figura 4.7: Boxplot das médias do RPD em relação a E_{size} do conjunto de médio porte

Tabela 4.4: Média do RPD em relação aos valores de E_{size} para cada conjunto de instâncias teste.

Conjunto	$E_{size} = 5$	$E_{size} = 10$	$E_{size} = 15$	$E_{size} = 20$
Pequeno	0,38	0,30	0,28	**0,25**
Médio	0,84	0,78	0,76	**0,68**
Grande	3,41	3,27	3,05	**2,93**

testes executados com heurísticas híbridas que utilizam a técnica de *Path Relinking* (GRASP+PR e GRASP+ILS+PR) utilizaram $E_{size} = 20$ em todos os conjuntos de instâncias.

4.5 Resultados e Comparações das Heurísticas

Os experimentos computacionais foram divididos em três partes. Na primeira, é feito a comparação entre os resultados das heurísticas com as soluções obtidas através do *software* de otimização CPLEX 12.1 da IBM (2009). Ressalta-se que este *software*

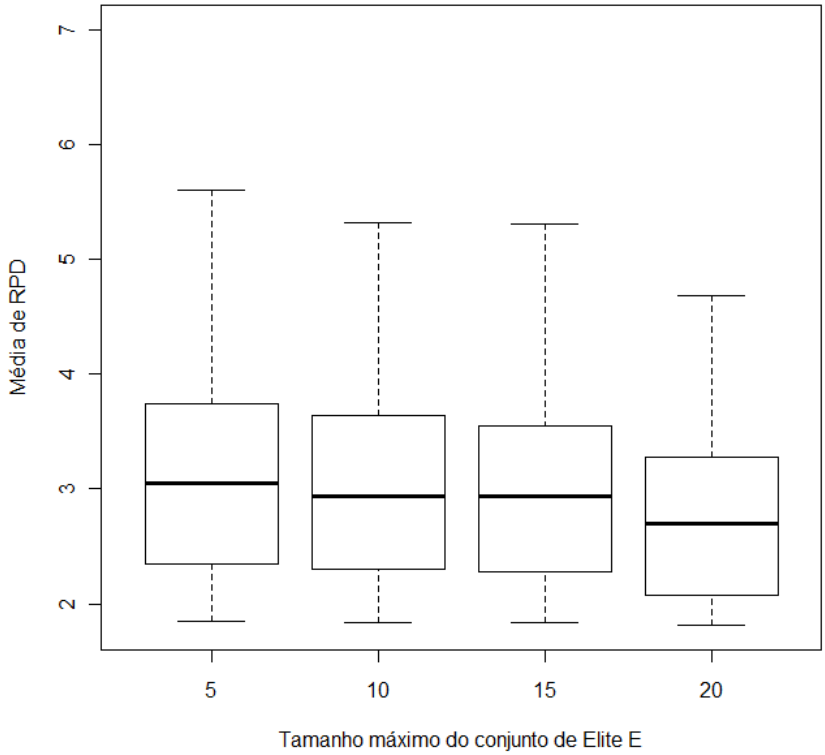


Figura 4.8: Boxplot das médias do RPD em relação a E_{size} do conjunto de grande porte

obtém soluções através da resolução do modelo matemático apresentado no Capítulo 2. Na segunda parte dos experimentos, as heurísticas são testadas na resolução de instâncias de médio e grande porte. Neste experimento os resultados das heurísticas implementadas são comparados utilizando um teste de análise de variância (ANOVA) para determinar se existe uma diferença estatisticamente significante entre elas. No terceiro experimento é avaliada a convergência das heurísticas implementas. Para comparar tais algoritmos é realizado um teste de probabilidade empírica, apresentados em Aiex & Resende (2006).

4.5.1 Experimento 1: Comparação com soluções ótimas

No primeiro experimento é utilizado o conjunto de instâncias de pequeno porte. Este conjunto foi resolvido com o modelo de PLIM utilizando do software CPLEX. Este *software* foi capaz de obter a solução ótima em baixo tempo computacional (veja Tabela 4.5).

Ressalta-se que o *software* CPLEX não conseguiu encontrar a solução ótima para todas as instâncias de um grupo com mais de 11 tarefas em um tempo limite de 3 horas. A Tabela 4.5 contém as médias do RPD das heurísticas implementadas e do CPLEX.

Tabela 4.5: Média do desvio relativo percentual dos algoritmos propostos (instâncias de pequeno porte)

$n \times m$	CPLEX Tempo(s)	GRASP Tempo(s)	CPLEX	GRASP	GRASP +ILS	GRASP +PR	GRASP +ILS+PR
8×3	8,47	1,20	0,00	**0,00**	**0,00**	**0,00**	**0,00**
8×5	8,03	2,00	0,00	0,18	**0,00**	**0,00**	**0,00**
9×3	9,44	1,35	0,00	0,61	0,51	**0,00**	0,35
9×5	12,54	2,25	0,00	0,25	0,25	0,08	**0,01**
10×3	7,54	1,50	0,00	0,14	0,07	**0,00**	**0,00**
10×5	6,54	2,50	0,00	1,16	1,11	1,04	**0,83**

Note que na Tabela 4.5 a média de RPD para a coluna CPLEX, em todos os grupos de instâncias, é igual a zero, pois o CPLEX determinou a solução ótima para todas as instâncias de grupos com menos de 11 tarefas. A primeira e segunda colunas da Tabela 4.5 apresentam, respectivamente, o tempo de execução, em segundos, para o CPLEX e as heurísticas implementadas concluírem suas execuções. Além disso, é observado nessa tabela que as heurísticas hibridas (GRASP+ILS,GRASP+PR e GRASP+PR) apresentaram, na maioria dos grupos de instâncias, um resultado melhor que o do simples GRASP.

O método GRASP em todos os grupos de instâncias, com exceção do grupo com o menor número de tarefas e máquinas ($n = 8, m = 3$), não foi possível obter a solução ótima encontrada pelo CPLEX. No entanto, as heurísticas híbridas implementadas foram capazes de determinar tais soluções ótimas para os grupos de instâncias de tamanho 8×3, 8×5, 9×3 e 10×3. Para os outros grupos, as soluções obtidas foram bem próximas à solução ótima.

4.5.2 Experimento 2: Resultados para instâncias de médio e grande porte

No segundo conjunto de experimentos, as heurísticas implementadas foram testadas em 4 grupos de instâncias de médio porte e 30 grupos de instâncias de grande porte. Cada grupo possui 10 instâncias distintas. Para o conjunto de instâncias de médio porte, a Tabela 4.6 apresenta a comparação dos RPDs entre: CPLEX, GRASP, GRASP+ILS,

GRASP+PR e GRASP+ILS+PR. O CPLEX foi executado utilizando um tempo de CPU máximo de 3 horas e em nenhum caso o *software* foi capaz de obter a solução ótima. Sendo assim, a solução retornada pelo *software* é um limite superior (*upper bound*).

Tabela 4.6: Média do desvio relativo percentual para as heurísticas propostas (instâncias de médio porte)

$n \times m$	CPLEX	GRASP	GRASP+ILS	GRASP+PR	GRASP+ILS+PR
20x3	19,38	0,76	0,69	0,19	**0,16**
20x5	29,63	2,22	2,13	0,78	**0,59**
30x3	66,60	1,42	1,38	**0,98**	**0,98**
30x5	111,54	2,29	1,85	0,94	**0,74**

Na Tabela 4.6 observa-se que as soluções obtidas pelas heurísticas implementadas são bem melhores em comparação às soluções geradas pelo CPLEX. Observe que todas as soluções obtidas pelo CPLEX são notavelmente piores quando o número de tarefas e máquinas são incrementadas. Esta diferença fica mais notável em testes utilizando grupos de instâncias que possuem um número maior de tarefas e máquinas ($n = 30, m = 5$). Devido à diferença entre os RPDs das heurísticas implementadas e dos resultados obtidos pelo CPLEX, o *software* não for utilizado em testes aplicados ao conjunto de grande porte.

Neste grupo de instâncias de médio porte, a heurística GRASP+ILS+PR mostrou melhor desempenho em comparação às heurísticas testadas. Vale lembrar que em todos os testes as heurísticas utilizaram o mesmo tempo de execução de $(n \times m \times 50)$ milissegundos, ou seja, um instância que pertence ao grupo com $n = 30$ e $m = 5$ é resolvido em 7,5 segundos.

As heurísticas são também testadas utilizando os grupos de instâncias de grande porte ($n = 50, 60, 70, 80, 90, 100$) e ($m = 10, 15, 20, 25, 30$). Cada grupo possui 10 instâncias distintas. A Tabela 4.7 mostra a média dos RPD pra as heurísticas GRASP, GRASP+ILS, GRASP+PR e GRASP+ILS +PR.

Na Tabela 4.7 observa-se que os RPDs das heurísticas híbridas são menores que os RPDs da GRASP convencional em todos os 30 grupos de instâncias. Isto mostra que as hibridizações da heurística GRASP com ILS e *Path Relinking* produzem ganhos consideráveis. Para cada instância do grupo de tamanho 100×30 (100 tarefas e 30 máquinas), as heurísticas foram executadas por 150 segundos.

Na Figura 4.9 mostra-se o *boxplot* das distribuições de médias de RPD das heurísticas implementadas. Nesta figura observa-se que a heurística GRASP+ILS+PR apresenta uma média de RPD geralmente menor que as demais heurísticas. Esta figura

Tabela 4.7: Média do desvio relativo percentual dos algoritmos propostos (instância de grande porte)

$n \times m$	GRASP	GRASP + ILS	GRASP + PR	GRASP +ILS+PR
50×10	6,04	5,12	2,88	**2,85**
50×15	8,87	6,38	3,92	**2,53**
50×20	10,78	8,38	5,63	**4,01**
50×25	14,15	12,11	7,15	**4,56**
50×30	13,12	12,27	6,97	**5,88**
60×10	5,72	4,11	3,46	**2,78**
60×15	8,08	5,47	3,84	**2,59**
60×20	7,94	5,26	2,57	**2,57**
60×25	10,17	7,86	4,97	**3,07**
60×30	15,7	11,47	8,10	**4,81**
70×10	5,55	3,14	3,77	**2,13**
70×15	6,07	4,23	3,22	**1,75**
70×20	8,91	6,35	5,09	**3,19**
70×25	10,08	7,46	4,54	**2,90**
70×30	10,65	7,96	5,26	**3,25**
80×10	4,83	2,72	3,58	**2,66**
80×15	6,21	3,68	4,03	**2,38**
80×20	8,21	6,17	4,76	**3,16**
80×25	9,78	7,24	4,97	**2,95**
80×30	10,14	7,24	5,28	**2,63**
90×10	5,39	2,17	3,88	**2,17**
90×15	6,53	3,12	4,65	**2,10**
90×20	8,66	5,32	5,27	**3,01**
90×25	8,25	5,31	4,48	**2,46**
90×30	9,21	6,40	4,61	**2,47**
100×10	5,40	2,04	4,59	**2,04**
100×15	4,63	1,77	3,03	**1,70**
100×20	7,70	3,94	4,65	**2,57**
100×25	8,16	4,07	4,60	**1,83**
100×30	8,71	6,52	4,86	**3,03**
Média	8,45	5,84	4,62	**2,87**

também apresenta que a heurística híbrida GRASP+ILS+PR possui a menor dispersão da média de RPD.

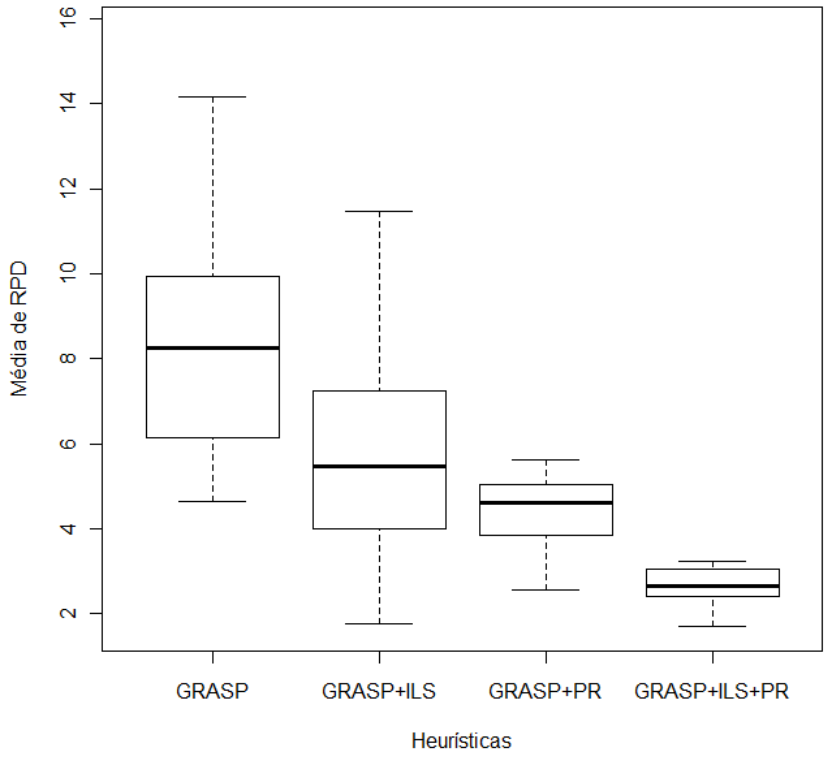


Figura 4.9: Boxplot das médias do desvio de percentagem relativa(RPD) das heurísticas

Para validar os resultados das heurísticas é interessante verificar se as diferenças entre as médias de RPD são estatisticamente significativas. Os testes estatísticos foram feitos utilizando o *software* estatístico R 2.13 Venables & M. (2011) com um nível de significância de 95% ($threshold = 0,05$), que determina a confiabilidade do teste. Foi realizada uma análise de variância seguindo o procedimento ANOVA preconizado por Montegomery (2007) para testar a significância estatística das diferenças entre as médias do RPD através da comprovação ou rejeição de uma hipótese nula. O resultado do teste ANOVA é apresetado na Tabela 4.8. As colunas df, **F** e *p-value* representam, respectivamente, o grau de liberdade, o valor de **F** do ANOVA e a probabilidade de aceitação da hipótese nula. Esta hipótese propõe que não existe diferença entre as médias do RPD das heurísticas implementadas. Porém como o *p-value* (probabilidade de aceitação da hipótese nula) é menor que o *threshold* ($2,2^{-16} < 0,05$) a hipótese nula será rejeitada. Este resultado do teste ANOVA comprova que a diferença entre médias é estatisticamente significante.

Tabela 4.8: Resultados do teste ANOVA das médias de RPD

df	**F**	*p-value*
3	40.74	$2,2^{-16}$

Para localizar quais heurísticas possuem esta diferença foi utilizado um teste de Tukey para comparação de múltiplas médias. A Tabela 4.9 apresenta os resultados do teste de Tukey.

Tabela 4.9: Resultados do teste de Tukey da comparação de médias de RPD

Heurísticas	diferença de RPD	*p-value*
GRASP+ILS ↔ GRASP	-2.61	0.0000106
GRASP+ILS+PR ↔ GRASP	-5.58	0.0000000
GRASP+PR ↔ GRASP	-3.83	0.0000000
GRASP+ILS+PR ↔ GRASP+ILS	-2.97	0.0000005
GRASP+PR ↔ GRASP+ILS	-1.22	0.0925874
GRASP+PR ↔ GRASP+ILS+PR	1.75	0.0055797

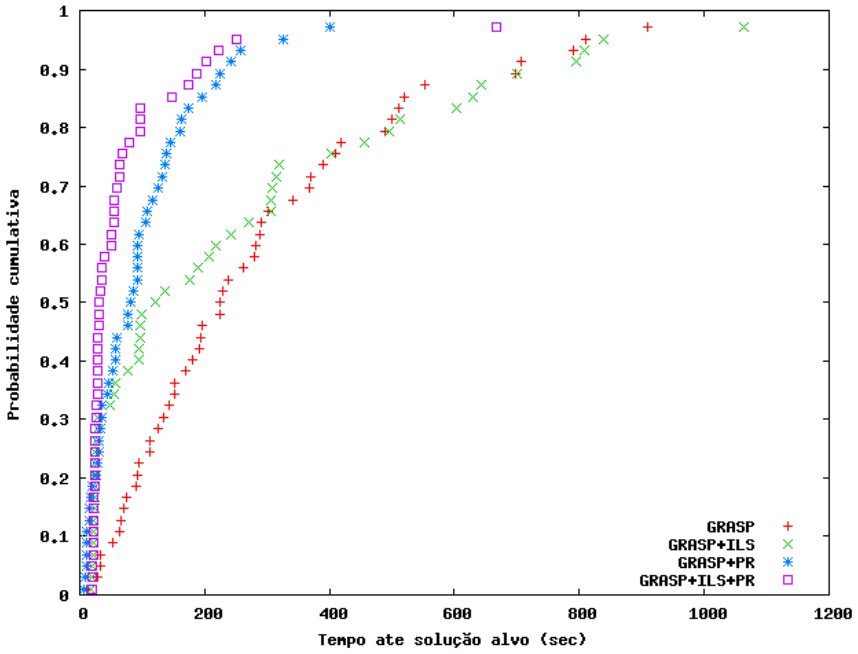
Na Tabela 4.9 são mostrados os resultados estatísticos da comparação de médias de RPD das heurísticas implementadas. A coluna *diferença de RPD* e *p-value* representam, respectivamente, a diferença entre as médias obtidas pelo teste de Turkey e o valor de probabilidade de aceitação da hipótese nula.

Analisando a tabela é possível afirmar que a média de RPD da heurística híbrida GRASP+ILS é menor que a do GRASP em 2.61. É possível afirmar também que esta diferença é estatisticamente significante, pois seu valor *p-value* é menor que o $threshold = 0,05$, ou seja, a heurística híbrida GRASP+ILS é significativamente melhor que a heurística GRASP. Assim sendo, todas as heurísticas híbridas são significativamente melhores que a heurística GRASP.

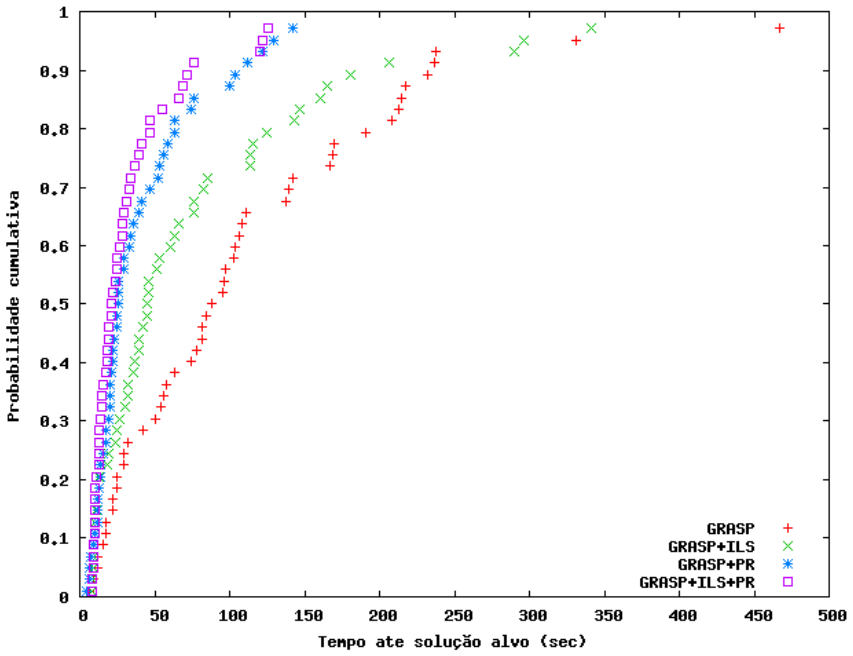
Nota-se que na Tabela 4.9, a comparação das médias entre as heurísticas híbridas GRASP+PR e GRASP+ILS, o valor *p-value* está acima do $threshold = 0,05$. Este fato significa que a hipótese nula não pode ser rejeitada. Por tal fato, não é possível afirmar que uma heurística é significamente melhor que a outra. Também nesta tabela, observa-se que a diferença das médias do RPD das duas heurísticas híbridas GRASP+PR e GRASP+ILS+PR é positiva, ou seja, a média do GRASP+PR é maior que a média do GRASP+ILS+PR. Considerando que o valor do *p-value* está abaixo do $threshold = 0,05$ para GRASP+PR e GRASP+ILS+PR, então pode-se afirmar que a média do GRASP+ILS+PR é significativamente melhor que a do GRASP+PR.

4.5.3 Experimento 3: Testes de probabilidade empírica

O propósito deste experimento é analisar a convergência das heurísticas propostas. Cada heurística (GRASP, GRASP+ILS, GRASP+PR, GRASP+ILS+PR) foi executada 50 vezes. A condição de parada considerada é atingir um valor objetivo-alvo, ou seja, os algoritmos concluem a execução quando encontram uma solução pelo menos tão boa quanto a solução alvo. A solução alvo considerada para todas as execuções é a melhor solução obtida pelo algoritmo GRASP depois de $n \times m \times 50$ milissegundos de tempo de CPU. Neste experimento são feitos gráficos dos testes de probabilidade empírica que são interessantes para plotar tempos de CPU. Na heurística GRASP, e demais procedimentos implementados com base nela, estes tempos são assumidos para se ajustarem a uma distribuição exponencial, Aiex & Resende (2006). Normalmente este é o caso para heurísticas baseadas em busca local para problemas de otimização combinatória, Aiex et al. (2002). Para mais informações sobre os testes de probabilidade empírica o leitor deve consultar Aiex & Resende (2006). Na Figura 4.10 são ilustrados os testes para duas instâncias (70×10 e 100×10) de grande porte. Estas figuras mostram que as heurísticas híbridas GRASP+ILS, GRASP+PR e GRASP+ILS+PR possuem uma maior probabilidade para encontrar a solução alvo gastando o menor tempo de CPU, ou seja, estas heurísticas possuem a melhor convergência. Entre as heurísticas implementadas o GRASP+PR apresentou possuir uma convergência melhor que o GRASP+ILS e o GRASP+ILS+PR exibiu ter a melhor convergência entre todos os métodos. Nesta Figura também pode ser observado que e a heurística GRASP possui a pior convergência. Dos resultados obtidos, pode-se concluir que a hibridização da heurística GRASP foi vantajosa para o bom desempenho da heurística. Os testes de probabilidade empírica para algumas instâncias são encontrados no apêndice A.



(a) Instância 70 × 10



(b) Instância 100 × 10

Figura 4.10: Teste de probabilidade empírica para analisar a convergência das heurísticas.

Capítulo 5

CONCLUSÕES

Neste trabalho foi abordado o problema de sequenciamento de tarefas em máquinas paralelas não relacionadas considerando tempo de preparação de máquina com o objetivo de minimizar a soma de penalidades por adiantamento e atraso da produção.

Foi utilizado um modelo de programação linear inteira mista do problema e através desse modelo foram resolvidas instâncias de pequeno porte com o objetivo de validar as heurísticas desenvolvidas. Também foram implementadas três heurísticas hibridas, adaptadas do procedimento GRASP. Na heurística híbrida, denotada por GRASP+ILS, a etapa de busca local é substituída pela heurística ILS. A heurística híbrida GRASP+PR aplica a técnica de intensificação *Path Relinking* na solução obtida pela busca local do GRASP, com uma solução escolhida de um conjunto de soluções elites. O último método híbrido implementado, denotado por GRASP+ILS+PR, combina o procedimento GRASP com a heurística ILS e a técnica PR. Vale ressaltar a complexidade do problema, pois a avaliação de soluções necessita do cálculo de datas ótimas de inicio e conclusão.

Foram gerados três conjuntos de instâncias de diferentes tamanhos. Inicialmente, estas instâncias foram utilizadas para calibrar os parâmetros de entrada de todas as heurísticas. O objetivo dessa calibração foi obter o melhor valor dos parâmetros utilizados nas heurísticas, levando em consideração os diferentes conjuntos de instâncias do problema.

Foram executados três tipos de experimentos para avaliar os resultados das heurísticas. O primeiro experimento utilizou as instâncias de pequeno porte para comparar os resultados obtidos pelo *software* CPLEX com os resultados obtidos pelas heurísticas híbridas implementadas. As heurísticas implementadas conseguiram resultados, na maioria dos casos, iguais ou muito semelhantes aos resultados ótimos obtidos pelo CPLEX.

O segundo experimento realizado teve como objetivo comparar os resultados obtidos através do CPLEX com os das heurísticas implementadas utilizando instâncias de médio porte. Os resultados obtidos pelo CPLEX, utilizando 3 horas como tempo máximo de execução, foram consideravelmente piores quando comparados com os das heurísticas implementadas. Em seguida, as heurísticas implementadas também foram testadas em instâncias de grande porte. Os resultados obtidos foram comparados entre si através de um teste de análise de variância ANOVA. Nesse teste foi verificado que as diferenças entre as médias do desvio relativo percentual das heurísticas são estatisticamente significantes. Em seguida, é feito um teste de Tukey para determinar se os resultados de cada heurística são estatisticamente significantes em relação às demais heurísticas implementadas. Os resultados mostraram que as heurísticas híbridas GRASP+ILS e GRASP+PR possuem um desempenho estatisticamente melhor que o GRASP implementado, porém, não é possível estabelecer qual dos dois procedimentos (GRASP+ILS e GRASP+PR) é estatisticamente melhor. O teste Tukey também mostrou que a heurística híbrida GRASP+ILS+PR é estatisticamente melhor que todas as demais heurísticas implementadas.

O terceiro experimento comparou a convergência das heurísticas utilizando testes de probabilidade empírica. Este experimento mostrou que as heurísticas híbridas GRASP+PR e GRASP+ILS+PR possuem convergências mais rápidas quando comparadas aos demais métodos. Este experimento também mostrou que a heurística GRASP possui uma convergência mais lenta entre as heurísticas implementadas neste trabalho.

Finalizando, pode-se concluir que os resultados obtidos com as heurísticas híbridas baseadas no procedimento GRASP implementadas nesta dissertação tiveram um desempenho superior aos do GRASP mais simples. Os resultados mostraram que as heurísticas híbridas apresentam bom desempenho em termos de qualidade de soluções obtidas, assim como possuem uma convergência mais rápida em relação ao método GRASP convencional.

Como trabalho futuro, sugere-se a hibridização das heurísticas GRASP com outras heurísticas como *Simulated Annealing* e *Variable Neighborhood Search*. Também se sugere incluir um procedimento de busca local utilizando uma vizinhança gerada por movimentos de troca entre tarefas. Finalmente, sugere-se a comparação de resultados com outros procedimentos, tais como ILS e Algoritmos Genéticos.

Referências Bibliográficas

Aiex, R. & Resende, M. (2006). Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366.

Aiex, R.; Resende, M. & Ribeiro, C. (2002). Probability distribution of solution time in grasp. *Jornal of Heuristics*, 8:343–373.

Arenales, M.; Morabito, R.; Armentano, V. A. & Yanasse, H. (2006). *Pesquisa operacional - Modelagem e algoritmos*. Campus.

Baker, K. R. (1990). Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 28:22–36.

Balakrishnan, N.; Kanet, J. J. & Sridharan, S. V. (1999). Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers and Operations Research*, 26:127–141.

Baxter, J. (1990). Probability distribution of solution time in grasp. *Journal of the Operational Research Society*, 32:815–819.

Cheng, T. & Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 9:271–292.

Du, J. & Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research*, 15:483–495.

Feo, T. & Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problems. *Operations Research Letters*, 8:67–71.

França Filho, M. F. (2007). *GRASP e Busca Tabu aplicados a problemas de programação de tarefas em maquinas paralelas*. PhD thesis, Faculdade de Engenharia Elétrica e de Computação – Universidade Estadual de Campinas. vtls 000437102.

Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completness.* W. H. Freeman.

Glover, F. (1996). Tabu search and adaptive memory programing. *Interface in Computer Science and Operational Reaserch*, pp. 1–75.

Hirano, H. & Makota, F. (2006). *JIT is Flow: Practice and Principles of Lean Manufacturing.* P C S Press.

IBM (2009). *IBM ILOG CPLEX V12.1 User's Manual for CPLEX.* IBM.

James, R. J. W. & Buchanan, J. T. (1997). A neighborhood scheme with a compressed solution space for the early/tardy scheduling problem. *European Journal of Operational Research*, 102:513–527.

Kim, D. W.; Na, D. G. & Chen, F. F. (2003a). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing*, 19:173–181.

Kim, S. S.; & Eom, D. H. (2003b). A due date density-based categorising heuristic for parallel machines scheduling. *International Journal of Advanced Manufacturing Technology*, 22:753–760.

Laguna, M. & Martí, R. (1999). Grasp with path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52.

Lee, C. Y. & Choi, J. Y. (1995). A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. *Computers and Operations Research*, 22:857–869.

Lee, C. Y. & Choi, J. Y. (1997). Scheduling jobs on parallel machines with sequence dependent setup times. *European Journal of Operational Research*, 100:464–474.

Lenstra, J. K.; Rinnooy, K. A. H. G. & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

Liaee, M. & Emmons, H. (1997). Scheduling families of jobs with setup times. *International Journal of Production Economics*, 51:165–176.

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

Logendran, R. & Saputra, H. (2007). Real-time scheduling of jobs with sequence-dependent setups on unrelated parallel machines. *Proceedings of the Eighth Industrial Engineering Research Conference*, 51.

Lourenço, H.; Martin, O. & Stutzle, T. (2002). *Optimization and heuristics of scheduling.* Kluwer Academic Publishers. Handbook of Metaheuristics.

Montegomery, D. (2007). *Design and Analysis of Experiments.* John Willy and Sons.

Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production.* Productivity Press.

Pinedo, M. (1995). *Scheduling Theory.* Prentice Hall.

Potts, C. & M.Y, K. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, 120:228–249.

Radhakrishnan, S. & Ventura, J. A. (2000). Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, 38:2233–2252.

Rangel, M. & Abreu, N. (2000). Grasp para o pqa: um limite de aceitação para soluções iniciais. *Pesquisa Operacional*, 20:45–58.

Resende, M. (2008). Metaheuristic hybridization with grasp. *Tutorials in Operations Research.* `http://www.optimization-online.org/DB_HTML/2008/04/1954.html`.

Resende, M. & Werneck, R. (2004). A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10:59–88.

Resende, M. G. C. & Ribeiro, C. C. (2005). *Metaheuristics: Progress as real problem solvers.* Kluwer Academic Publishers.

Ribeiro, C. & Rosseti, I. (2004). Parallel grasp for the 2-path network design problem. *Lecture Notes in Computer Science*, 2400:922–926.

Ribeiro, C. & Urrutia, S. (2007). Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, 127:775–787.

Ruiz, R. & C., A. (2007). Unrelated parallel machines scheduling with resource-assignable sequence dependent setup times. *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling Theory and Applications*, pp. 439–446.

Ruiz, R.; Maroto, C. & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34:461–476.

Serifoglu, F. S. & Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research*, 26:773–787.

Vallada, E. & Ruiz, R. (2009). Genetic algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times. *Boletin de la Sociedad Española de Cerámica y Vidrio*, 44:39–44.

Venables, W. N. & M., S. (2011). *An Introduction to RNotes on R: A Programming Environment for Data Analysis and Graphics*. R Development Core Team.

Wan, G. & Yen, B. P. C. (2002). Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *European Journal of Operational Research*, 142:129–146.

Zhu, Z. & Heady, R. B. (1998). Minimizing the sum of job earliness and tardiness in a multimachine system. international journal of production research. *International Journal of Production Research*, 36:1619–1632.

Apêndice A

TESTES DE PROBABILIDADE EMPÍRICA

As Figuras A.1 até A.12 ilustram os testes de probabilidade empírica para algumas instâncias de diferente número de tarefas e diferente número de máquinas.

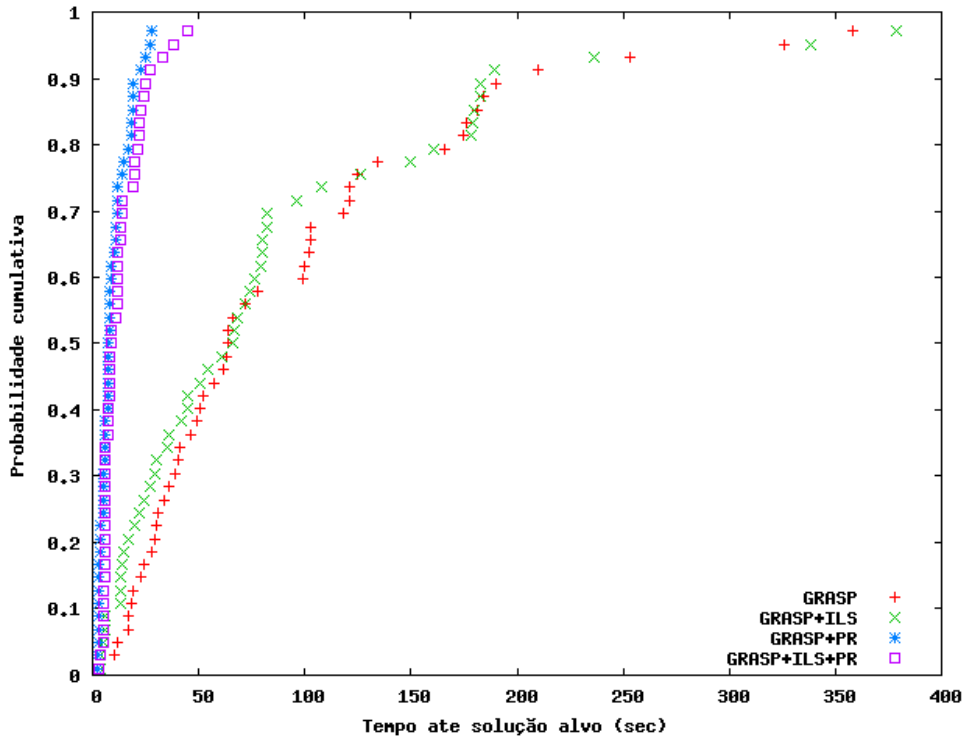


Figura A.1: Instância 50×10

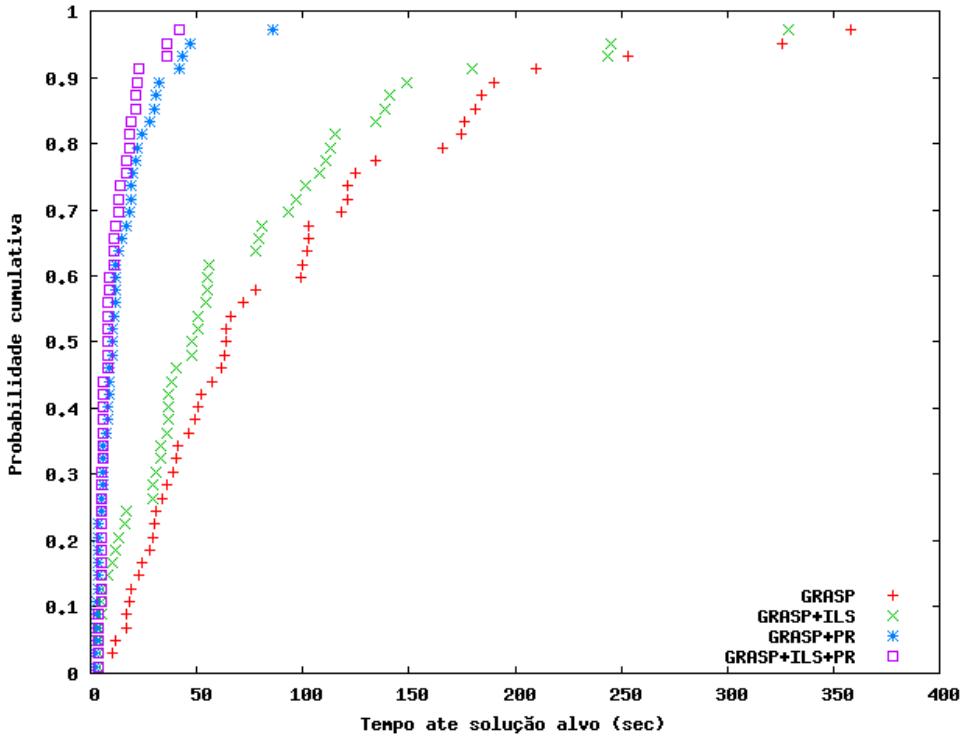


Figura A.2: Instância 50×30

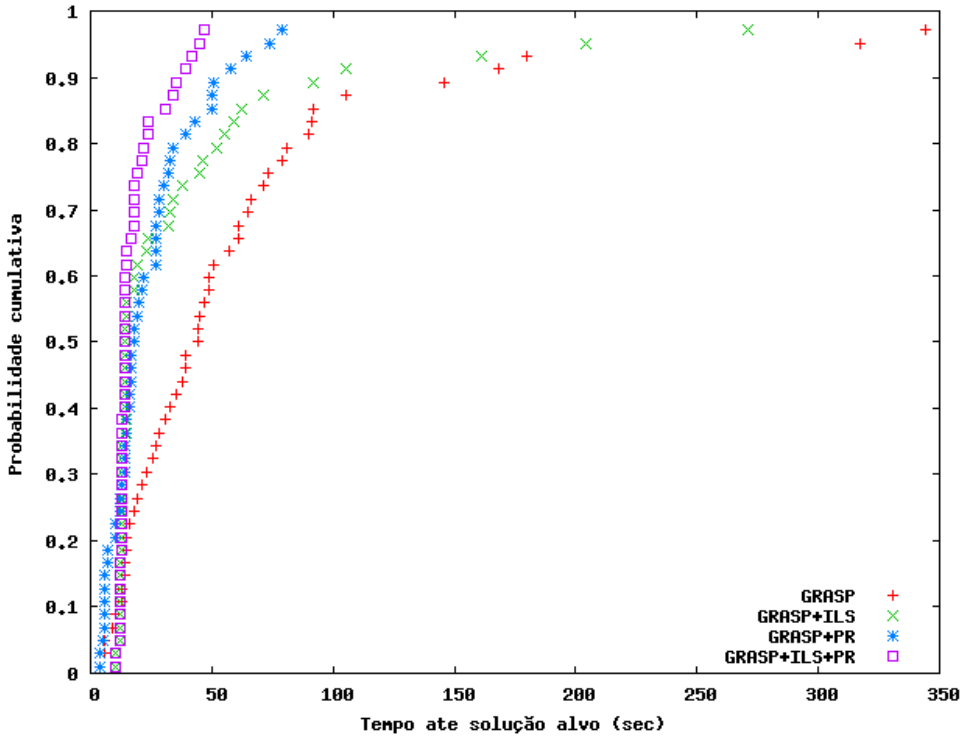


Figura A.3: Instância 60×10

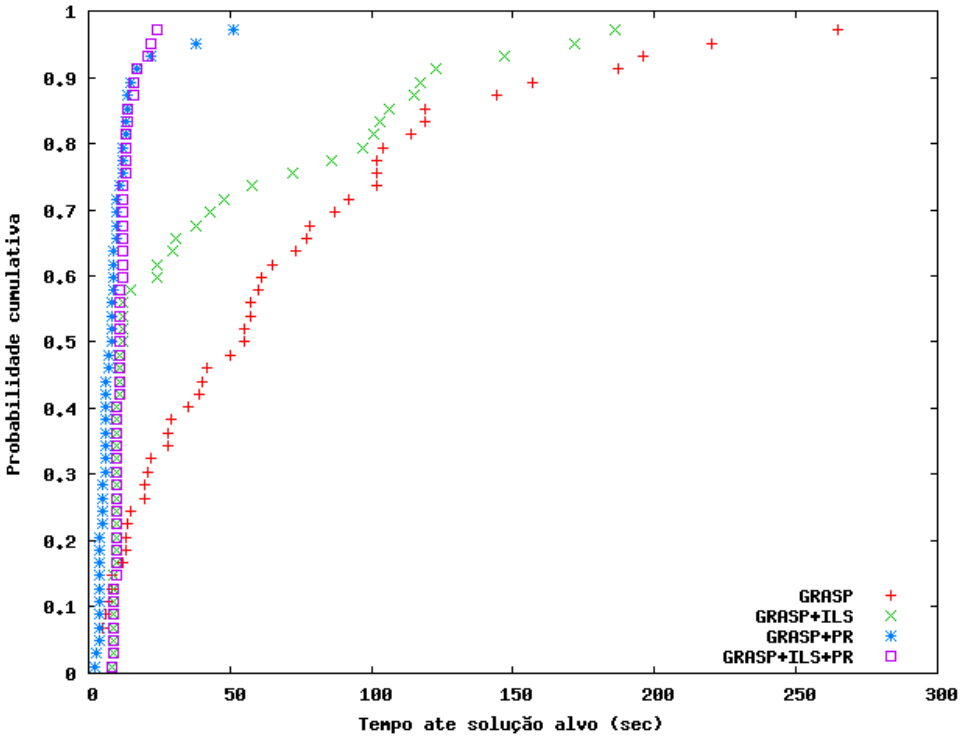


Figura A.4: Instância 60 × 30

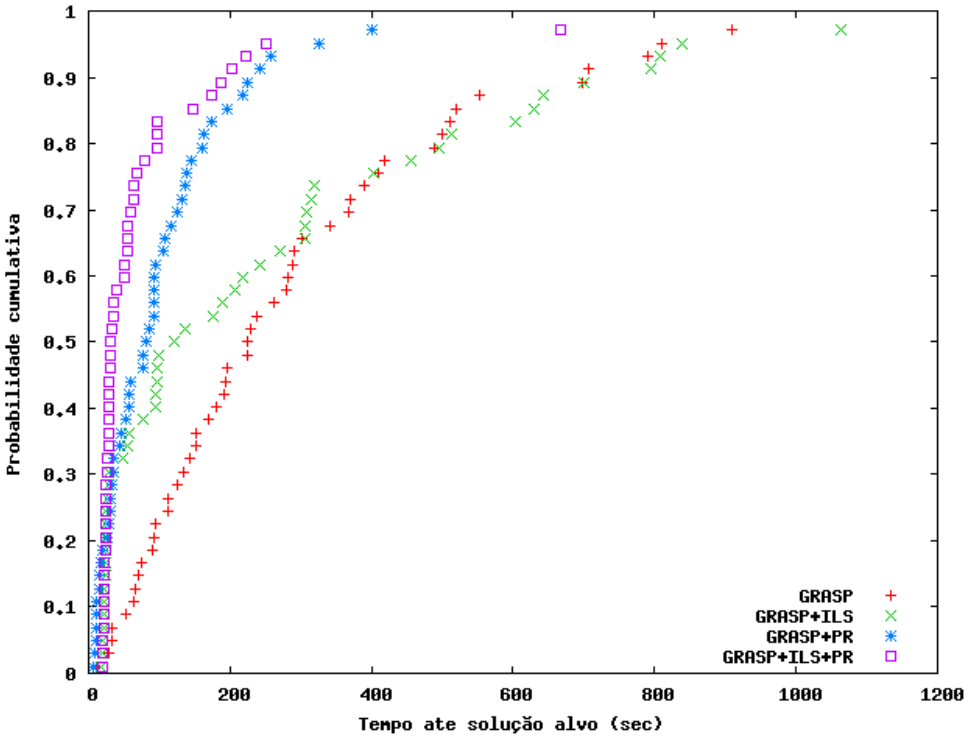


Figura A.5: Instância 70 × 10

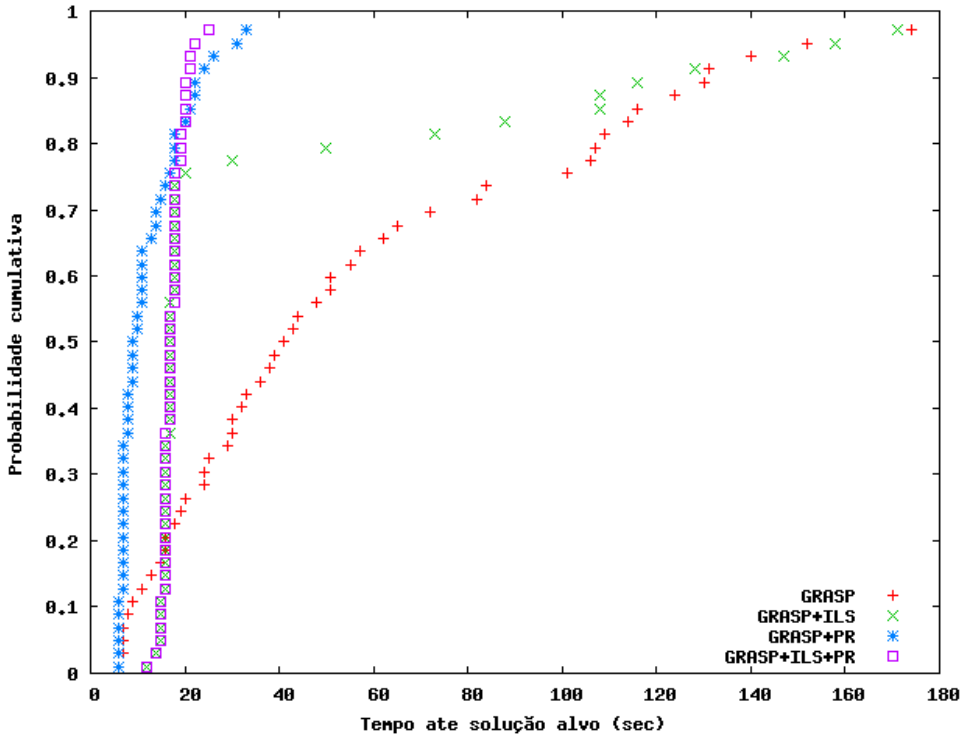


Figura A.6: Instância 70×30

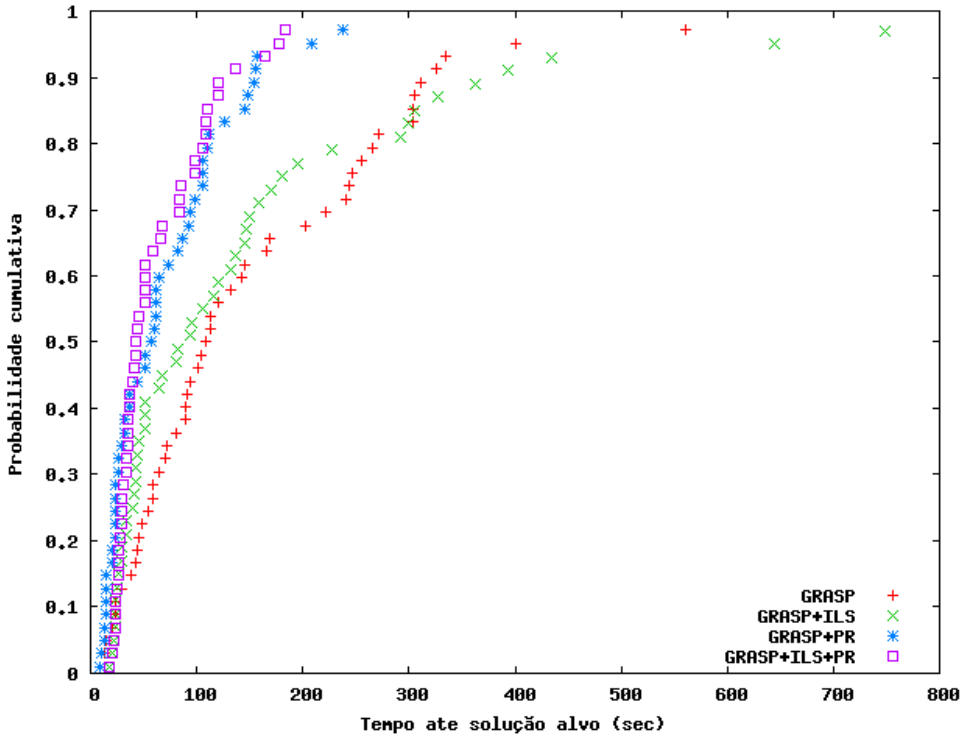


Figura A.7: Instância 80×10

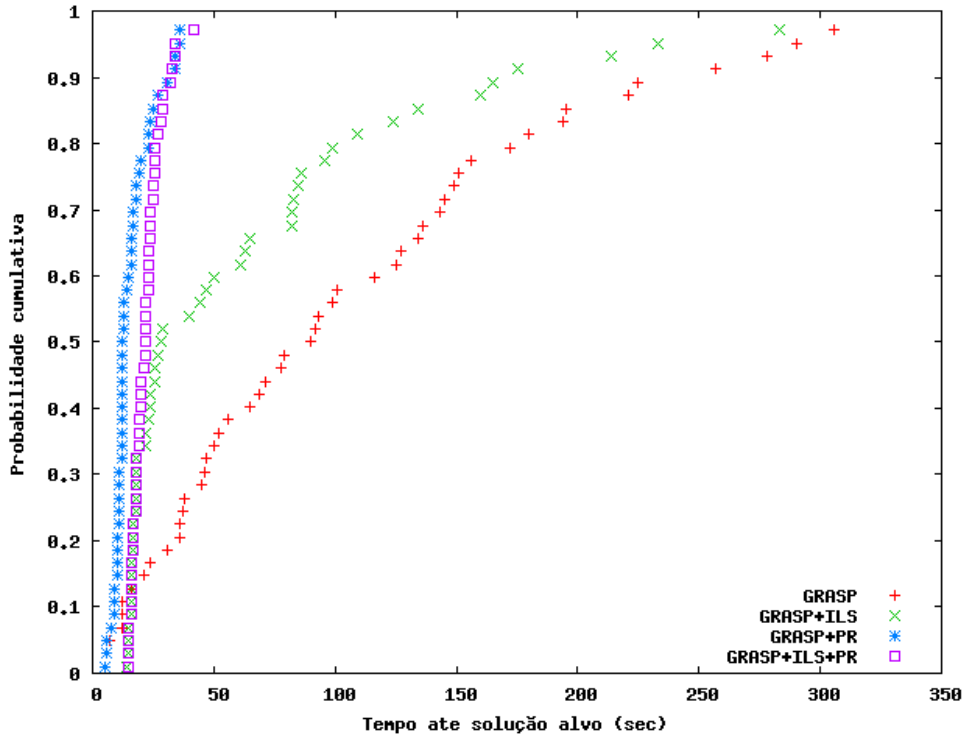


Figura A.8: Instância 80 × 30

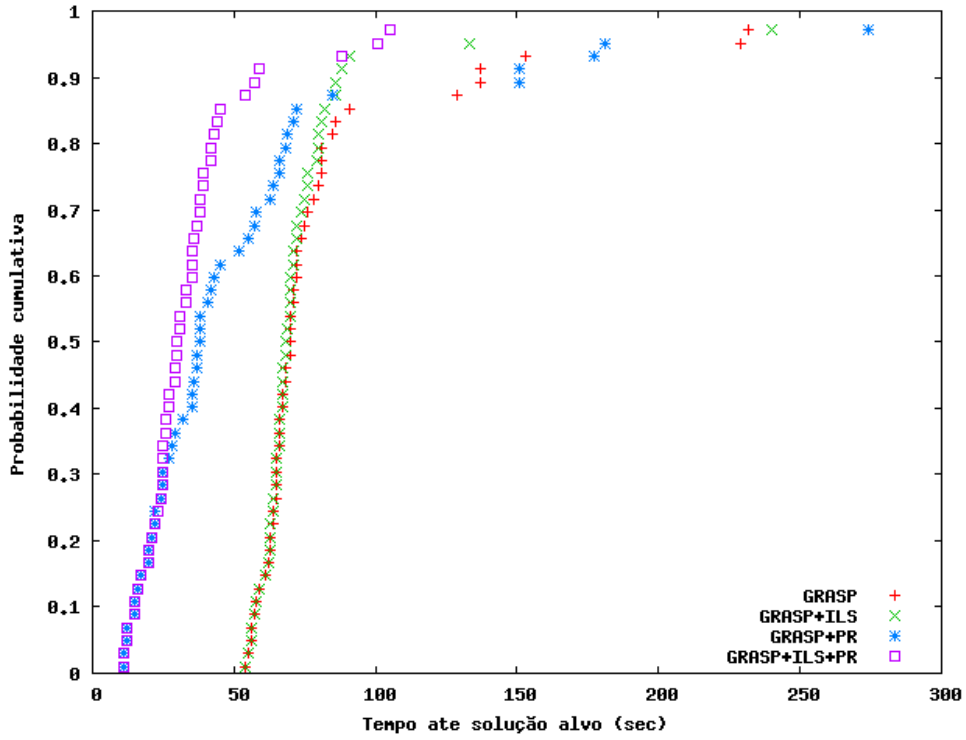


Figura A.9: Instância 90 × 10

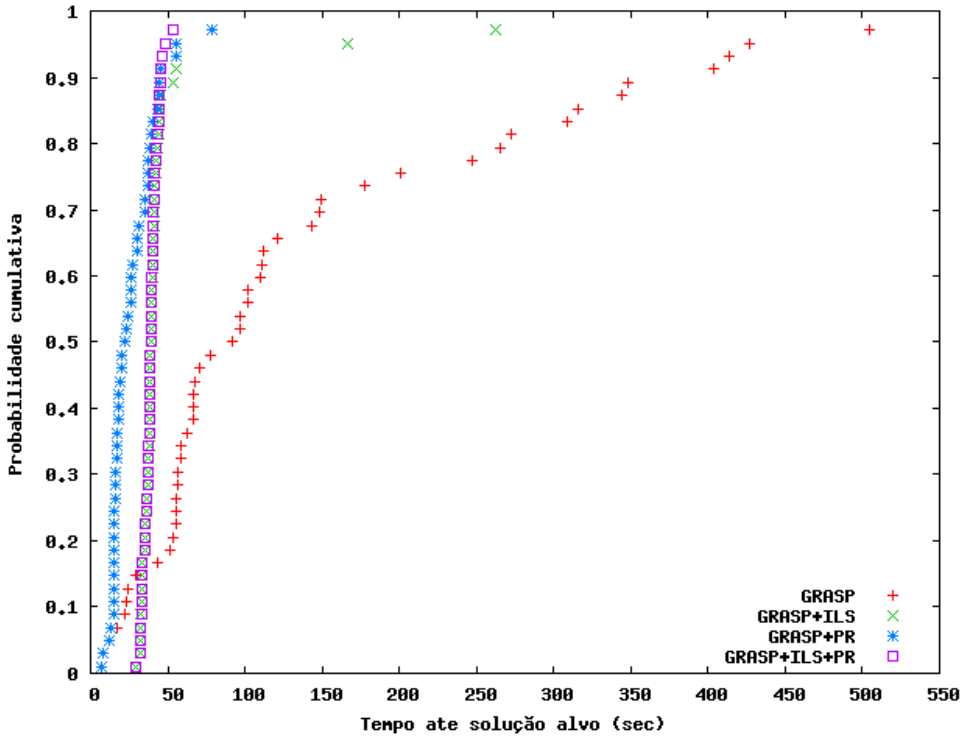


Figura A.10: Instância 90 × 30

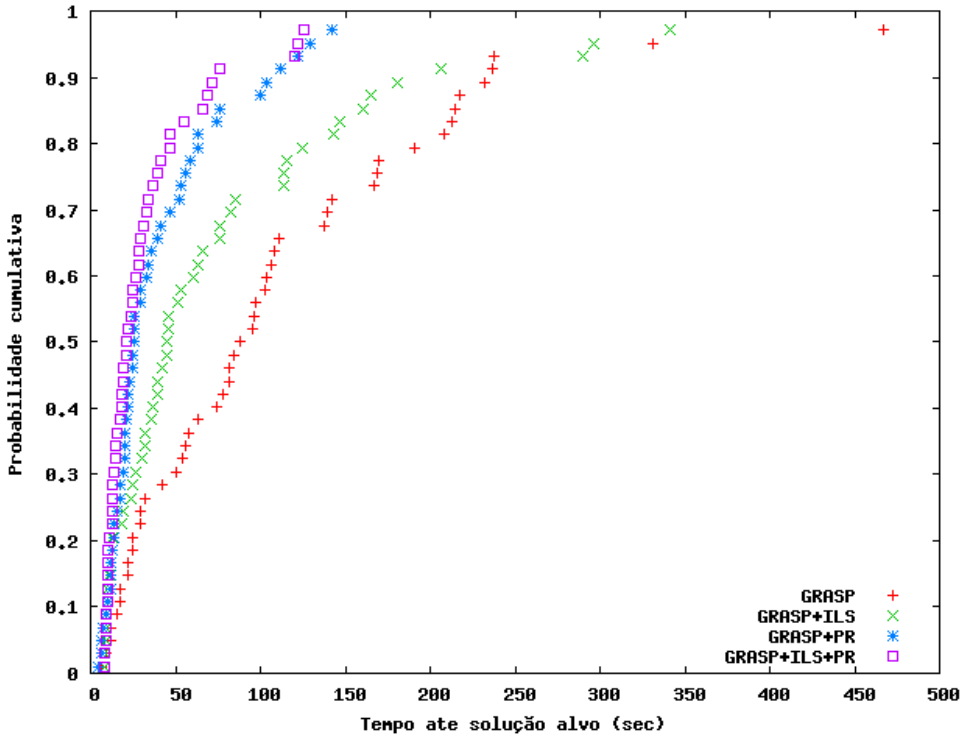


Figura A.11: Instância 100 × 10

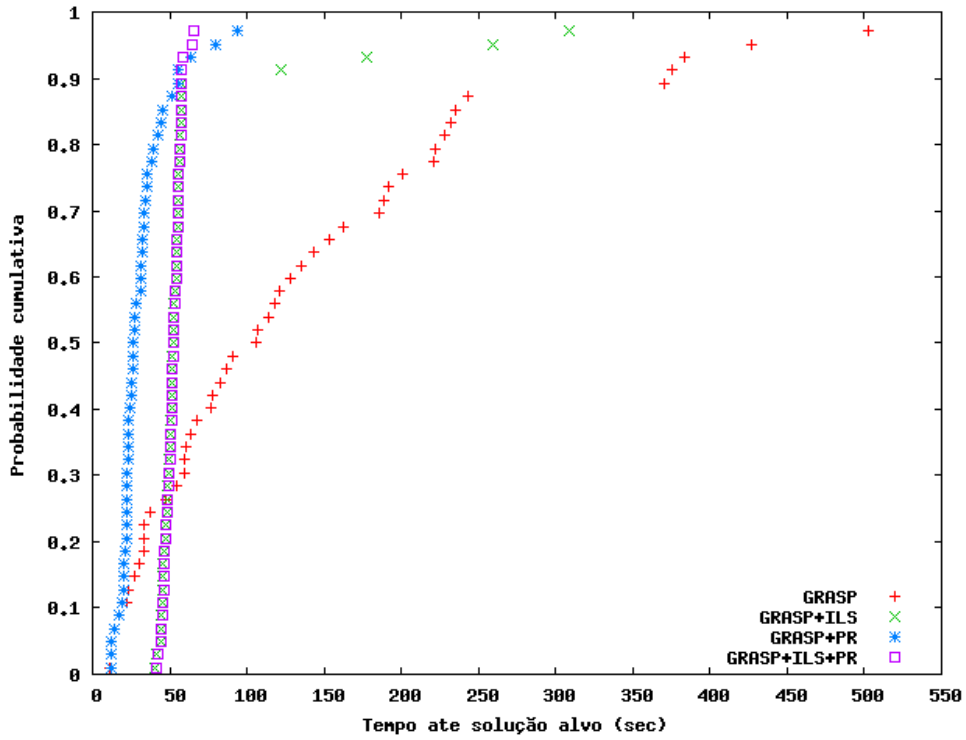


Figura A.12: Instância 100×30