EDMAR HELL KAMPKE

METAHEURÍSTICAS PARA O PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA E DE RECURSOS

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
2010

EDMAR HELL KAMPKE

METAHEURÍSTICAS PARA O PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA E DE RECURSOS

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: 09 de Março de 2010

André Gustavo dos Santos
(Co-Orientador)

Mauro Nacif Rocha
(Co-Orientador)

Heleno do Nascimento Santos

Marcone Jamilson Freitas Souza

José Elias Cláudio Arroyo
(Orientador)

Dedico essa dissertação aos meus pais
Helmar e Ilza

Aos meus irmãos
Vera e Edgar

E em memória dos meus amados e eternos amigos
Lelê e Valquíria

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus que me deu força e coragem para realizar esse trabalho. Agradeço a Ele por sempre colocar verdadeiros amigos em meu caminho, que sempre me auxiliaram, não só nesse trabalho, mas em toda a vida.

Agradeço também aos meus pais, Helmar e Ilza, e aos meus irmãos Vera e Edgar, por todo amor e incentivo concedido ao longo de todo esse tempo. Agradeço por sempre acreditarem em mim e pelo apoio em todos os momentos; sem vocês, nada disso seria possível. Aos meus pais, de forma especial, agradeço por toda a educação que recebi e pela oportunidade que me deram de realizar meus sonhos. Vocês são exemplos de vida para mim. Agradeço também a todos os familiares: tios(as) e primos(as), pelo grande incentivo.

Agradeço aos meus amigos, por todos os momentos divididos, pelo companheirismo, pelo carinho e pela paciência. Vocês também fazem parte dessa conquista. Obrigado por tudo que sempre fizeram e fazem por mim. Um agradecimento especial, através do Pastor Jony Wagner de Almeida, à Igreja Presbiteriana de Viçosa que sempre foi minha segunda família. Aos amigos colaboradores e diretores das empresas que tive o prazer de compartilhar um pouco do meu conhecimento, meus sinceros agradecimentos por acreditarem em mim e por terem sido parceiros desta conquista.

Agradeço ao Professor José Elias Cláudio Arroyo, por tamanha paciência em me receber na sua sala mesmo sem agendar. Você, além de orientador, é também um amigo que sempre me mostrou a direção correta dos passos a serem tomados. Agradeço também aos meus co-orientadores, Professor André Gustavo dos Santos e Professor Mauro Nacif Rocha, pelos conselhos e opiniões sobre o trabalho.

A todos do corpo docente do Departamento de Informática, da Universidade Federal de Viçosa, pela formação e conhecimento recebidos. Muito obrigado, pois os ensinamentos não foram somente para o crescimento profissional, mas para a vida toda. Um agradecimento especial aos professores: Heleno Nascimento dos Santos,

José Luís Braga e Ricardo Santos Ferreira, por estarem sempre de prontidão pra me auxiliar.

Ao Altino Alves de Souza Filho, secretário da Pós-graduação, pelas ótimas conversas e por sempre se prontificar a ajudar perante as burocracias do mestrado.

A todos os colegas, do mestrado e da graduação, que me incentivaram, ajudaram e contribuíram direta ou indiretamente para a realização desse trabalho. De forma especial, aos amigos Antônio Almeida de Barros Júnior e Reuryson Fidelis de Morais, companheiros de trabalhos e de estudos durante o mestrado e graduação, respectivamente.

Aos meus amigos que estiveram longe de mim nesses momentos, mas que sempre deram um jeito para disfarçar a distância. Especialmente Radical®, Débs, Jê e Fred por me darem a honra, mesmo não merecendo, de ser chamado de "padrinho". Obrigado por sempre se preocuparem comigo e pelas terapias gratuitas. Aos hereges mais queridos do mundo, PG e Fernanda, por estarem sempre presentes nos melhores e nos piores momentos. Obrigado pela cobertura jornalística da defesa. Agradeço também de forma especial aos meus amigos, que de tão especiais, Deus achou melhor levá-los para junto de si: Lelê e Valquíria!

Ao pessoal da república dos Frávios, pelos vários anos de convivência e de amizade, pela paciência com meu stress e com as minhas maluquices. Na correria do dia-a-dia, vocês sempre demonstravam, através de palavras e atitudes, motivos para continuar prosseguindo. Estendo os meus agradecimentos aos familiares e namoradas dos Frávios por também me suportarem com carinho.

A Universidade Federal de Viçosa.

BIOGRAFIA

EDMAR HELL KAMPKE, filho de Helmar Kampke e Ilza Hell Kampke, brasileiro nascido em 08 de Agosto de 1982 na cidade de Vila Velha, no estado do Espírito Santo.

No ano de 2001, após concluir o ensino médio na cidade de Vila Velha, ingressou no curso de graduação em Ciência da Computação na Universidade Federal de Viçosa, concluindo em Julho de 2005.

Após a conclusão da graduação trabalhou em empresas da área de informática, nas cidades de Belo Horizonte-MG, Rio de Janeiro-RJ e Vitória-ES.

Em 2007, foi aprovado na seleção do programa de pós-graduação em Ciência da Computação do Departamento de Informática – DPI, onde, em março deste mesmo ano, iniciou o mestrado na Universidade Federal de Viçosa – UFV, defendendo sua dissertação em março de 2010.

Trabalhou como Analista de Sistemas na ATAN Ciência da Informação de setembro de 2005 a março de 2007 e recentemente deixou o cargo de Analista de Sistemas na BMS LTDA, que ocupava desde Abril de 2007, para se dedicar exclusivamente a esta dissertação.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CPLEX – Software matemático de otimização

DJASA – *Dynamic Job Assignment with Setups resource Assignment*

EAs – *Evolutionary Algorithms*

GB – Método GRASP Básico

GR – Método GRASP com mecanismo reativo

GRASP – *Greedy Randomized Adaptive Search Procedures*

GRPR – Método GRASP com mecanismo reativo e técnica *path relinking*

ILS – *Iterated Local Search* e Método ILS Básico

ILSPR – Método ILS com técnica *path relinking*

LC – Lista de Candidatos

LRC – Lista Restrita de Candidatos

LSMs – *Local Search Metaheuristics*

MMPI – Modelo Matemático de Programação Inteira

PMD – Percentual Médio de Desvio

PO – Pesquisa Operacional

PPTMPSR – Problema de Programação de Tarefas em Máquinas Paralelas com *Setup Times* e Recursos

PTMP – Programação de Tarefas em Máquinas Paralelas

SPT – *Shortest Processing Time*

SPTSA – *Shortest Processing Time with Setups resource Assignment*

SPSTSA – *Shortest Processing and Setup Time with Setups resource Assignment*

WSPT – *Weighted Shortest Processing Time*

RESUMO

KAMPKE, Edmar Hell, M.Sc., Universidade Federal de Viçosa, Março de 2010. **Metaheurísticas para o problema de programação de tarefas em máquinas paralelas com tempos de preparação dependentes da sequência e de recursos**. Orientador: José Elias Cláudio Arroyo. Co-Orientadores: André Gustavo dos Santos e Mauro Nacif Rocha.

Os problemas de programação de tarefas em máquinas paralelas são importantes na área de otimização combinatória, pois quase sempre envolvem problemas rotineiros em indústrias de pequeno e grande porte. Este trabalho aborda o problema de sequenciamento de tarefas em máquinas paralelas, com tempos de preparação das máquinas dependentes da sequência e do número de recursos utilizados. A característica deste problema é que o tempo de preparação não é determinado apenas pela máquina e pela sequência das tarefas, mas também pela quantidade de recursos associados, que varia entre um valor mínimo e um máximo. Dada a complexidade combinatória do problema, inicialmente propõe-se um algoritmo baseado na metaheurística GRASP, no qual o parâmetro de aleatoriedade utilizado na fase de construção é auto-ajustado de acordo com as soluções previamente encontradas (GRASP Reativo). Em seguida, propõe-se um algoritmo baseado na metaheurística *Iterated Local Search* (ILS). Esta metaheurística foi proposta recentemente na literatura e está sendo aplicada satisfatoriamente em diversos problemas de otimização combinatória. Em ambos os algoritmos, é utilizada a estratégia de intensificação baseada na técnica de Reconexão de Caminhos, que explora trajetórias que conectam soluções de alta qualidade encontradas pelos algoritmos. Os resultados obtidos pelos algoritmos propostos são comparados entre si e com os melhores resultados disponibilizados na literatura. A análise e a discussão dos resultados mostram que as metaheurísticas aplicadas, apresentaram resultados satisfatórios, sendo possível melhorar, em média, 9,14%, os resultados da literatura, o que comprova a viabilidade do uso destes algoritmos na resolução de problemas práticos existentes nas indústrias.

ABSTRACT

KAMPKE, Edmar Hell, M.Sc., Universidade Federal de Viçosa, March, 2010. **Metaheuristics for the parallel machines scheduling problem with resource-assignable sequence dependent setup times**. Adviser: José Elias Cláudio Arroyo. Co-Advisers: André Gustavo dos Santos and Mauro Nacif Rocha.

The problems of scheduling jobs on parallel machines are important problems of combinatorial optimization. They involve routine problems in industries of small and large. This work addresses an unrelated parallel machine problem with machine and job sequence dependent setup times. The characteristic of this problem is that the amount of setup times do not only depend on the machine and job sequence, but also on a number of resources assigned, which can vary between a minimum and a maximum. Due to the combinatorial complexity of this problem, initially we propose an algorithm based on the GRASP metaheuristic, in which the basic parameter that defines the restrictiveness of the candidate list during the construction phase is self-adjusted according to the quality of the solutions previously found (reactive GRASP). Then, we propose an algorithm based on the Iterated Local Search (ILS) metaheuristic. This metaheuristic was proposed recently in the literature and is being applied successfully to several combinatorial optimization problems. In both algorithms, an intensification strategy based on the path relinking technique is used. This consists in exploring paths between elite solutions found by the algorithms. The results obtained by the proposed algorithms are compared with each other and with the best results available in literature. The analysis and discussion of results obtained with the algorithms, leads us to conclude that the metaheuristics show good performance, it is possible to improve, on average, 9,14% the results of the literature, which demonstrates the feasibility of using these algorithms to solve real problems existing in industries.

1 INTRODUÇÃO

A Pesquisa Operacional (PO) é um conjunto de métodos científicos utilizados na resolução de diferentes problemas de tomada de decisões, que estão envolvidos em determinar o melhor aproveitamento de recursos escassos. A PO se comporta como uma ferramenta auxiliar nos processos de tomada de decisões, de planejamento, e em quaisquer setores e níveis da economia, visando à maior satisfação dos usuários, definidos num determinado contexto. Dessa forma, a PO é uma área da Ciência da Computação que possui grande interseção com outras áreas do conhecimento humano.

Com isso, nas últimas décadas observamos um grande impulso da PO na área de administração, com o apoio a tomada de decisões e também visando aumento da qualidade dos processos de produção e atendimento.

Dentre os problemas que podem ser resolvidos pela PO destacam-se os Problemas de Otimização Combinatória. A Otimização Combinatória é uma disciplina de tomada de decisões no caso de problemas discretos que pode ser encontrada em diversas áreas, tais como, problemas de planejamento e programação (*scheduling*) da produção, problemas de corte e empacotamento, roteamento de veículos, redes de telecomunicação, sistemas de distribuição de energia elétrica, problemas de localização, dentre outras. Em muitos destes problemas surgem frequentemente vários critérios de desempenho (funções objetivos), em geral, conflitantes entre si. Por exemplo, o controle de chão de fábrica envolve a execução do plano de produção e para tal é necessário programar a produção em cada centro de trabalho de forma a satisfazer objetivos globais da empresa. Como destacado por VOLMANN *et al*. (1988), existem três objetivos básicos na programação de tarefas (*jobs*) na produção. O primeiro objetivo está relacionado com datas de entrega das tarefas: basicamente, não se deseja atraso em relação a essas datas, e quando o custo de estoque é relevante, tenta-se evitar que as tarefas sejam finalizadas muito antes dessas datas. O segundo objetivo está relacionado com o tempo de fluxo de tarefas no chão de fábrica: deseja-se que esse tempo seja curto, ou equivalentemente, que o

estoque em processamento seja baixo. O terceiro objetivo envolve a utilização dos centros de trabalho: deseja-se maximizar a utilização de equipamentos e de mão de obra. Portanto, o analista de produção (decisor) deve optar por uma solução que pondere os objetivos globais da empresa.

Grande parte dos problemas de otimização combinatória não possuem um método de solução eficiente e fazem parte da classe de problemas designados como *NP-difíceis*. A principal característica desta classe de problemas é que apesar de nunca ter sido provado a inexistência, nenhum algoritmo eficiente para estes problemas foi encontrado. Uma abordagem interessante para a solução destes tipos de problema é a utilização de algoritmos aproximados ou heurísticos, que buscam soluções aproximadas gastando um baixo tempo de processamento.

1.1 Motivação

A concorrência acirrada tem levado as empresas a utilizar técnicas cada vez mais sofisticadas para gerir as atividades de produção, cujo objetivo consiste no atendimento das necessidades de seus consumidores, que primam por produtos e serviços com qualidade, rapidez, confiabilidade, flexibilidade e custo adequado.

As estratégias de produção objetivam dar suporte à empresa na obtenção de vantagem competitiva, que pode ser obtida através do gerenciamento adequado dos recursos produtivos, beneficiando a empresa com um conjunto de características de desempenho adequado as suas necessidades estratégicas.

Portanto, a gestão da produção desempenha um importante papel no auxílio ao cumprimento dos objetivos estratégicos da empresa. Uma das várias funções executadas pelos gestores é a atividade de programação da produção, que tem como objetivo comandar e gerenciar o processo produtivo, e caracteriza uma das atividades mais complexas no gerenciamento dos sistemas produtivos, uma vez que lida com diversos tipos diferentes de recursos e tarefas simultaneamente.

A programação da produção faz parte da rotina diária de qualquer empresa e envolve uma enorme quantidade de detalhes que afetam diretamente todo o processo na empresa. As decisões envolvidas no nível de programação são: designação de tarefas a máquinas e programação das tarefas em cada máquina, isto é, a sequência de processamento das tarefas e o instante de início e término do processamento.

Nesse contexto os problemas de programação de tarefas em máquinas têm destaque, já que ocorrem em grande escala nas indústrias e nem sempre a alocação de tarefas é feita da melhor maneira possível, gerando com isso perdas.

Encontrar uma solução ótima para um problema de programação de tarefas em máquinas é considerado um grande desafio. Porém, já existem na literatura diversas heurísticas para gerar soluções de alta qualidade, de acordo com o objetivo que se quer minimizar (ou maximizar).

Neste trabalho é abordado um problema de Programação de Tarefas em Máquinas Paralelas (PTMP), no qual são considerados tempos de preparação (*setup times*) de máquinas, que dependem de recursos disponíveis (por exemplo, mão-de-obra). Um dos primeiros trabalhos propostos para o problema PTMP foi apresentado por MARSH e MONTGOMERY (1973). Eles estudaram o problema considerando a minimização de *setup times*. GUINET (1991) modelou matematicamente o problema e aplicou métodos heurísticos para minimizar outros critérios tais como média dos tempos de conclusão (*completion times*) das tarefas, média dos atrasos das tarefas e o tempo máximo de conclusão (*makespan*).

O problema de PTMP considerado neste trabalho foi abordado pela primeira vez por RUIZ e ANDRÉS (2007) e por relacionar tempos de preparação e recursos é denotado por PPTMPSR (Problema de Programação de Tarefas em Máquinas Paralelas com *Setup Times* e Recursos). Este problema é provado ser NP-Difícil (RUIZ e ANDRÉS, 2007). Esta prova consiste em simplesmente reduzir o problema a um clássico problema de PTMP com *setup times* dependentes apenas da sequência, que é NP-Difícil (WEBSTER, 1997). Portanto, a utilização de heurísticas é uma das alternativas para se encontrar soluções de boa qualidade para casos do PPTMPSR de tamanho real.

Nos trabalhos mais recentes sobre problemas de PTMP, nota-se uma tendência de aplicação de metaheurísticas, tais como Algoritmos Genéticos (TAMINI e RAJAN, 1997; RUIZ e MAROTO, 2004), *Simulated Annealing* (RADHAKRISHNAN e VENTURA, 2000; KIM *et al.*, 2002), GRASP (FEO *et al.*, 1996; AIEX *et al.*, 2003) e *Iterated Local Search* (CONGRAM *et al.*, 2002; RUIZ e STÜTZLE, 2007).

Este problema é de suma importância, principalmente para as indústrias de manufatura, pois ocorre, por exemplo, na fabricação de cerâmicas, onde existem diferentes máquinas de polimento. Cada máquina inicia o polimento de um tipo de

cerâmica, e ao final a máquina precisa ser limpa e preparada para realizar o polimento de outro tipo de cerâmica. Este tempo de limpeza e preparação varia de acordo com a máquina, com o tipo da cerâmica e com o número de recursos utilizados, que neste caso, pode ser o número de pessoas que realizarão a limpeza e ajuste da máquina.

Portanto, a importância do problema e a dificuldade de encontrar soluções de qualidade em tempo computacional aceitável fornecem a motivação para a pesquisa baseada na utilização de métodos heurísticos.

1.2 Objetivos

O objetivo geral do trabalho é estudar, compreender e descrever os comportamentos e resultados obtidos na aplicação das metaheurísticas GRASP (*Greedy Randomized Adaptive Search Procedures*) e ILS (*Iterated Local Search*) para resolução do PPTMPSR, de forma a minimizar a soma dos tempos de conclusão das tarefas e o número de recursos utilizados. A técnica reconexão de caminhos (*path relinking*) é utilizada, em ambas metaheurísticas, para intensificar a busca de soluções, com a tentativa de encontrar soluções mais refinadas.

Os objetivos específicos desse trabalho são:

- Analisar um conjunto de casos do problema em busca de características comuns para sua melhor compreensão;
- Implementar algumas heurísticas construtivas, propostas na literatura, que encontrem uma solução inicial satisfatória;
- Aplicar as metaheurísticas GRASP e ILS, verificando, em seguida, se apresentam resultados satisfatórios;
- Utilizar a técnica *path relinking* nas metaheurísticas GRASP e ILS, para construir um caminho de busca, entre duas soluções de elite, a fim de tentar encontrar uma nova solução que melhore o resultado;
- Comparar os desempenhos das metaheurísticas com o desempenho de heurísticas propostas na literatura;
- Comparar o desempenho das metaheurísticas GRASP e ILS entre si, além de analisar se a utilização da técnica *path relinking* apresenta bom desempenho;

- Compreender e analisar os resultados obtidos, com o intuito de descrever padrões, parâmetros e situações do problema em que a utilização das metaheurísticas tenha mostrado resultados satisfatórios;
- Apresentar os resultados obtidos que comprovem: a viabilidade do uso de metaheurísticas ao problema e as vantagens que estas podem proporcionar na solução problemas reais.

O segundo desafio daqueles propostos pela SBC - Sociedade Brasileira de Computação, identificado no seminário "Grandes Desafios da Pesquisa em Computação no Brasil – 2006 – 2016" (SBC, 2006), está relacionado com a modelagem computacional de sistemas complexos artificiais, naturais e sócio-culturais e da iteração homem-natureza. O grande número de dados e variáveis envolvidas nos problemas de otimização combinatória, como o que é abordado neste trabalho, contribui para a resolução do segundo desafio, no que tange à modelagem computacional de sistemas complexos, já que utiliza algoritmos e estruturas para um processo de otimização mais eficiente.

1.3 Organização deste documento

Os resultados alcançados neste trabalho estão fortemente embasados em dados e informações publicadas na literatura. Para isso, foi realizada uma extensa pesquisa bibliográfica buscando trabalhos científicos que estivessem relacionados ao contexto do problema abordado e à metodologia proposta nesse trabalho. O Capítulo 2 apresenta a definição e a descrição detalhada do problema, incluindo o modelo matemático que é composto pela função objetivo e suas restrições. Antes disso, são relatados as variáveis e os dados de entrada do problema, e como eles estão relacionados. Esta descrição, inclusive com exemplo, fornece a visão geral do problema.

O Capítulo 3 descreve as heurísticas construtivas propostas na literatura para o problema e suas variações. É descrita também uma forma de tentar minimizar ainda mais a função objetivo durante a escolha da quantidade de recursos e tempos de preparação. Por fim é exemplificada a utilização de uma das heurísticas construtivas.

O Capítulo 4 apresenta os métodos heurísticos propostos: GRASP e ILS. No início os métodos são descritos na sua forma básica; em seguida, detalha-se cada uma das etapas dos métodos implementados. Por fim, é relatada a técnica *path*

relinking e a forma em que ela foi acoplada às metaheurísticas. Todos os procedimentos são apresentados juntamente com os seus respectivos pseudocódigos.

No Capítulo 5 são apresentados os resultados da aplicação das metaheurísticas propostas e da técnica de intensificação e diversificação *path relinking* ao conjunto de problemas disponibilizados na internet. São detalhados também os parâmetros usados nos métodos. Os resultados mostram como a metodologia pode ser utilizada para obter soluções de alta qualidade que minimizam o tempo de conclusão do processamento das tarefas e o número de recursos utilizados na preparação das máquinas. Para realizar a análise comparativa dos métodos com as heurísticas construtivas foram utilizados dados disponibilizados na literatura.

Finalmente, o Capítulo 6 apresenta as conclusões obtidas e as principais contribuições disponibilizadas por este trabalho, destacando a economia que a utilização dos métodos propostos proporciona nas indústrias em que o problema de programação de tarefas, abordado neste trabalho, ocorre. São disponibilizadas também sugestões para trabalhos futuros, ressaltando a possibilidade de aplicação de outras metaheurísticas ou novos critérios, na função objetivo, a serem minimizados. Além disso, são sugestionadas outras formas de busca por soluções, principalmente no procedimento de Busca Local, e também critérios de parada na execução das metaheurísticas.

Ao final do documento são apresentadas as referências utilizadas para o embasamento teórico do trabalho.

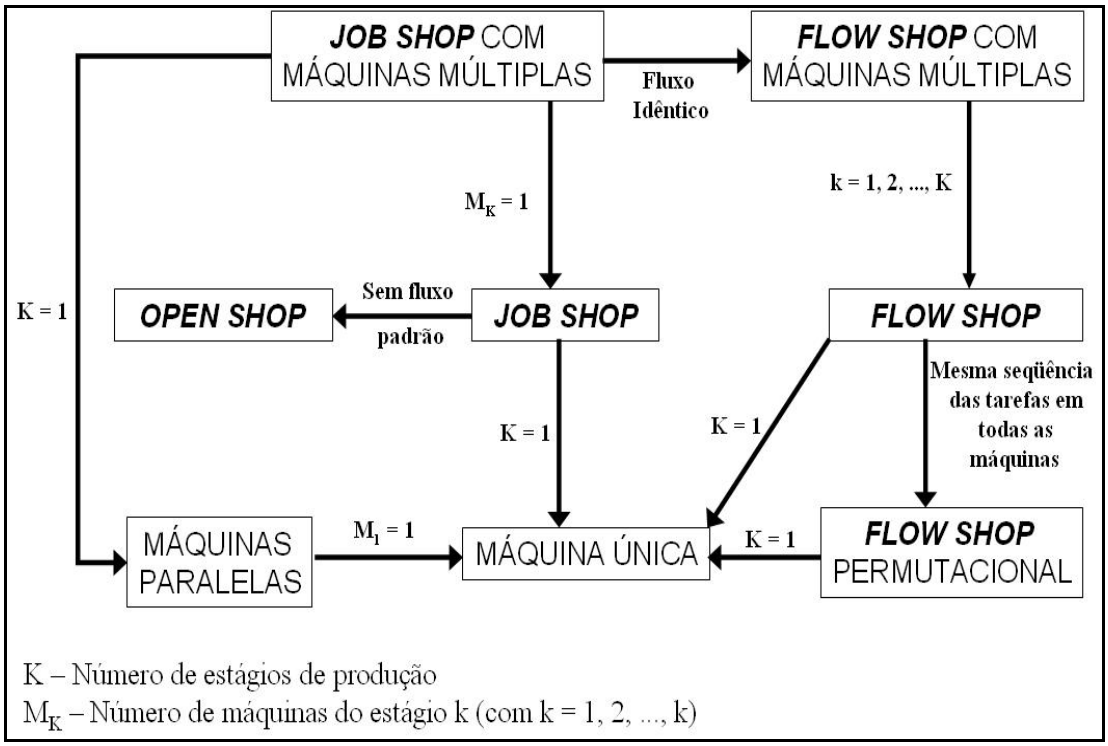
2 DEFINIÇÃO DO PROBLEMA

Os problemas de programação (sequenciamento) de tarefas em máquinas têm sido objeto de vários trabalhos nas últimas seis décadas. Da mesma forma, os tópicos relacionados a esses problemas têm sido de contínuo interesse de pesquisadores e profissionais da área. Conforme foi descrito no capítulo anterior, o eficiente sequenciamento de tarefas em máquinas possui um fator econômico associado, que tornou-se essencial para a sobrevivência das indústrias no ambiente de negócios altamente competitivo.

Nos problemas de programação de tarefas em máquinas, as restrições tecnológicas das tarefas e os objetivos da programação devem ser especificados. As restrições tecnológicas são determinadas pelo padrão do fluxo de tarefas nas máquinas levando a uma classificação conforme segue.

- ***Job Shop***: Cada tarefa tem sua própria ordem de processamento nas máquinas;
- ***Flow Shop***: Todas as tarefas têm o mesmo fluxo de processamento nas máquinas;
- ***Open Shop***: Não há fluxo definido (específico) para as tarefas a serem processadas nas máquinas;
- ***Flow Shop Permutacional***: Trata-se de *flow shop* no qual a ordem de processamento das tarefas deve ser a mesma em todas as máquinas;
- **Máquina Única**: Existe apenas uma máquina a ser utilizada;
- **Máquinas Paralelas**: São disponíveis mais de uma máquina, geralmente idênticas, para as mesmas operações;
- ***Job Shop* com Múltiplas Máquinas**: *Job shop* no qual em cada estágio de produção existe um conjunto de máquinas paralelas;
- ***Flow Shop* com Múltiplas Máquinas**: *Flow shop* no qual em cada estágio de produção existe um conjunto de máquinas paralelas.

Esses diversos tipos de problema são ilustrados na figura 2.1.



Fonte: (MACCARTHY e LIU, 1993)

Figura 2.1. Relação entre as classes de problemas de sequenciamento de tarefas em máquinas

Dos problemas acima, o problema *flow shop* numa única máquina é o caso mais simples, pois consiste na ordenação de *n* tarefas. Ou seja, o número de soluções possíveis neste caso é *n*! (*n* fatorial).

Existem problemas (geralmente de pequeno porte) que são relativamente simples de resolver através de programação linear inteira para se obter a solução ótima. Entretanto, a grande maioria dos problemas de programação da produção é intrinsecamente muito difícil de ser resolvida, de tal forma, que a grande maioria desses problemas é NP-difícil, ou seja, não foi encontrado ainda um algoritmo eficiente para sua resolução, mesmo não tendo sido provado a inexistência desses algoritmos. De acordo com RUIZ e ANDRÉS (2007), o problema abordado neste trabalho é NP-difícil, pois pode ser reduzido ao clássico problema de programação de tarefas com tempos de preparação dependentes apenas da sequência, que foi provado ser NP-Difícil por WEBSTER (1997).

2.1 Problemas de programação de tarefas em máquinas paralelas

Conforme a revisão de literatura apresentada por ALLAHVERDI *et al.* (2008), nas últimas décadas muitos estudos têm se dedicado na análise de problemas de

programação de tarefas em máquinas paralelas (PTMP). Estes estudos são consequência dos grandes desafios teóricos e da ampla aplicação industrial que os problemas de PTMP possuem.

Os problemas de PTMP se diferenciam daqueles que possuem apenas uma única máquina, ou um fluxo de produção unidirecional, já que nesses casos a programação é determinada apenas pelo sequenciamento das atividades a serem executadas. No caso dos problemas de PTMP, além da ordem de execução das tarefas, é necessário também definir em qual máquina cada tarefa será alocada, antes de fazer a alocação propriamente dita.

Um número interessante de características define os problemas de PTMP. Nestes problemas há um conjunto $N = 1, 2, ..., n$ de tarefas, onde cada uma tem que ser processada exatamente em uma máquina do conjunto $M = 1, 2, ..., m$. Inicialmente as tarefas devem ser associadas às máquinas e, em seguida, as tarefas associadas a cada máquina formarão uma sequência de processamento.

De acordo com FRENCH (1982), há três classes de problemas com máquinas paralelas:

- **Máquinas Idênticas** – Os tempos de processamento das tarefas são os mesmos em todas as máquinas;
- **Máquinas Uniformes (Proporcionais)** – Os tempos de processamento das tarefas variam de acordo com um padrão simples entre as máquinas. Esta variação está associada a um fator de proporcionalidade;
- **Máquinas Não-Relacionadas** – Os tempos de processamento das tarefas variam entre as máquinas, mas em um padrão completamente arbitrário.

Na programação da produção, os tomadores de decisão precisam definir múltiplos, muitas vezes conflitantes, critérios de decisão. Uma ampla revisão sobre problemas de PTMP que abordam os critérios convencionais de desempenho, como a data de entrega, o tempo de conclusão (*Completion Time*) e o tempo de fluxo das tarefas (*Flow Time*), foi apresentada por CHENG e SIN (1990).

Um algoritmo de programação dinâmica para problemas de programação de n tarefas em duas máquinas foi desenvolvido por ALIDAEE (1993). O objetivo desse era minimizar o tempo de fluxo ponderado das tarefas e discutir similaridades entre o problema investigado e os pesos dos adiantamentos e dos atrasos em uma única máquina. Posteriormente, WEBSTER (1995) apresentou limitantes superiores e

inferiores para o problema de programação de *n* tarefas em *m* máquinas de diferentes capacidades (máquinas não-relacionadas) e, da mesma forma que ALIDAEE (1993), o objetivo era minimizar o tempo de fluxo ponderado das tarefas.

Em meados da década anterior, GUPTA e RUIZ (2005) consideraram o problema de geração de um conjunto de eficientes programações em máquinas paralelas idênticas, envolvendo o tempo total de fluxo e o número total de tarefas em atraso. No mesmo ano, AMBUHL e MASTROLILLI (2005) investigaram o problema de minimização do tempo de fluxo máximo para programação de tarefas em *m* máquinas idênticas, quando interrupções (*preemptions*) são permitidas e quando não são permitidas.

Dentre os problemas de PTMP, o caso mais geral e complexo é quando as máquinas paralelas são independentes (não-relacionadas) entre si. Ou seja, o tempo de processamento de cada tarefa depende da máquina em que ela será associada (KIM *et al.*, 2002). Os tempos de processamento são determinísticos e conhecidos de antemão, sendo denotados por p_{ij}.

2.2 Conceitos de tempo de preparação (*setup time*)

Tempo de preparação, do inglês *setup time*, é o tempo requerido para preparar uma máquina para processar uma tarefa. Isto inclui obter ferramentas, posicionar o trabalho no processo de materiais, limpeza, recolocação de ferramental, posicionamento de acessórios, ajuste de ferramentas e inspeção de materiais, de acordo com CAO e BEDWORTH (1992).

Segundo ALLAHVERDI *et al.* (1999), grande parte das pesquisas em programação em máquinas, considera os *setup times* como não relevantes ou de pequena variação e, geralmente, os incluem nos tempos de processamento das tarefas. Esse procedimento simplifica muito a análise em determinadas aplicações, principalmente quando os *setup times* são consideravelmente menores que os tempos de processamento, ou em casos em que o *setup time* destina-se à produção de lotes e é executado somente uma vez para um grande lote de produção.

No entanto, para casos em que os *setup times* apresentam razão significativa diante dos tempos de processamento, há necessidade de tratá-los diferencialmente, uma vez que eles têm relação direta com a disponibilidade de equipamentos e acarretam custos específicos, como a necessidade de pessoal especializado para sua execução. De acordo com BARROS e MOCCELLIN (2004), o tratamento em

separado dos *setup times* pode levar, com a otimização do critério de desempenho adotado, a melhorias no atendimento à demanda e à facilidade no gerenciamento do sistema de produção.

Para o *setup time* separado dos tempos de processamento das tarefas, existem dois tipos de problemas. No primeiro, o *setup time* depende somente da tarefa a ser processada. Nesse caso ele é denominado independente da sequência de execução das tarefas. No segundo, o *setup time* depende de ambos, da tarefa a ser processada e da tarefa anterior, sendo denominado dependente da sequência de execução das tarefas.

Em ambos os casos, o *setup time* é tido como uma atividade que não agrega valor e, por isso, muitas pesquisas nas últimas décadas têm se voltado para técnicas de sequenciamento que minimizem o tempo total de *setup* (REDDY e NARENDRAN, 2003). Como pode ser visto em BURBIDGE (1975), ROBINSON (1990) e SHINGO (1996), há várias vantagens em reduzir o *setup time* no processo produtivo.

Segundo BAKER (1974), os *setup times* dependentes da sequência são comumente encontrados onde uma só instalação produz muitos tipos diferentes de itens, ou onde máquinas de múltiplos propósitos realizam uma variedade de tarefas.

O *setup time* tem relação direta com o grau de similaridade entre duas tarefas processadas sucessivamente em uma máquina. Portanto, se duas tarefas a serem processadas em sequência são similares, o *setup time* requerido é relativamente pequeno. Entretanto, se forem completamente diferentes, o tempo será proporcionalmente maior.

Quando o *setup time* da tarefa *i* para a tarefa *j* é diferente do *setup time* da máquina na sequência inversa, ou seja, da tarefa *j* para a tarefa *i* se diz que os *setup times* são assimétricos.

Conforme já citado anteriormente, um dos primeiros trabalhos considerando o *setup time* dependente da sequência, tanto em máquinas idênticas como em máquinas não-relacionadas, foi o de MARSH e MONTGOMERY (1973). O objetivo do problema era a minimização do total de *setup time*. A maior parte dos trabalhos na literatura aborda problemas em que o *setup time* é dependente da sequência em máquinas idênticas. Por exemplo, os trabalhos de BALAKRISHNAN *et al.* (1999), SIVRIKAYA-SERIFOGLU e ULUSOY (1999) ou RADAKRISHMAN e VENTURA (2000) propõem diferentes técnicas para resolver problemas de

programação de tarefas em máquinas idênticas, ou uniformes, cujo *setup time* é dependente da sequência.

No entanto, existem algumas exceções. ZHU e HEADY (2000) propuseram um modelo misto de programação inteira para o problema de programação de tarefas em máquinas não-relacionadas, com *setup time* dependente da sequência. O objetivo do modelo era minimizar o adiantamento e o atraso na conclusão das tarefas. WENG *et al.* (2001) modelou o problema com o objetivo de minimizar a média do tempo de conclusão das tarefas. Além disso, outra diferença com relação ao trabalho de ZHU e HEADY (2000) foi que o *setup time* não dependia das máquinas, apenas da sequência de tarefas.

Neste trabalho, estuda-se um complexo e real problema de programação de tarefas em máquinas paralelas e não-relacionadas. O *setup time* é dependente da sequência e da máquina, considerado assim assimétrico. Portanto, S_{ijk} denota o *setup time* utilizado para limpeza e preparação da máquina *i* após o processamento da tarefa *j* e, imediatamente, antes do processamento da tarefa *k*.

2.3 Relação entre *setup times* e recursos

Na prática é difícil definir o *setup time*. Uma das razões é que o tempo gasto na operação de *setup* depende, na grande maioria dos casos, de recursos a serem aplicados nessa operação. Por exemplo, considere o caso de máquinas que, ao finalizarem o processamento de uma tarefa, precisarão ser limpas por trabalhadores que, neste exemplo, serão os recursos utilizados no *setup*. Para limpar uma máquina, um único trabalhador (recurso) poderá gastar um tempo elevado. Entretanto, se mais trabalhadores limparem a máquina, o tempo gasto será menor. Obviamente, se associarmos mais trabalhadores (recursos) o *setup time* será menor. No entanto, como os recursos (trabalhadores) possuem um custo (valor da mão-de-obra), quanto mais recursos associarmos, maior será o custo deste *setup time*. Dessa forma, poderá haver um momento em que a associação de mais recursos tornará o custo do *setup* muito elevado e, consequentemente, inviável.

RUIZ e ANDRÉS (2007) realizaram uma extensa revisão de literatura na busca de problemas de sequenciamento de tarefas em máquinas cujo *setup time* estivesse relacionado com a quantidade de recursos; porém, nenhum trabalho foi encontrado. Na literatura, entretanto, existe uma grande quantidade de trabalhos em que os problemas de sequenciamento de tarefas possuem os tempos de processamento

controlados que, por sua vez, são os problemas que mais se aproximam dos que possuem o *setup time* relacionado com recursos. Para esse tipo de problemas de sequenciamento, onde o tempo de processamento pode ser controlado ou reduzido (com um custo associado), uma excelente revisão de literatura é apresentada por NOWICKI e ZDRZALKA (1990), sendo que um dos trabalhos mais recentes foi apresentado por ZHANG *et al.* (2001), onde foi estudado esse caso do problema em máquinas paralelas não-relacionadas, e com o objetivo de minimizar o tempo total ponderado dos tempos de conclusão das tarefas e o total dos custos gerados na redução dos tempos de processamento.

Resumidamente, conforme foi descrito nas seções anteriores, o problema abordado neste trabalho considera n tarefas que serão processadas em m máquinas paralelas, não-relacionadas e continuamente disponíveis. Cada máquina pode processar apenas uma tarefa por vez e cada tarefa pode ser processada em apenas uma máquina. Cada tarefa i possui, determinado, um tempo de processamento p_{ij} na máquina j. Considera-se, também, neste trabalho, o *setup time* S_{ijk} utilizado na limpeza e preparação da máquina i, após o processamento da tarefa j e antes do processamento da tarefa k. Neste trabalho, os *setup times* são dependentes da sequência e da máquina e, na maioria dos casos, são assimétricos, ou seja, $S_{ijk} \neq S_{ikj}$.

No problema abordado neste trabalho, além do *setup time* ser dependente da sequência e da máquina, também se consideram os recursos utilizados na limpeza e preparação das máquinas. Por isso, a quantidade de recursos atribuídos afeta a definição do *setup time* e, consequentemente, denota-se por R_{ijk} o número de recursos associados ao *setup time* S_{ijk}. Ou seja, R_{ijk} representa o número de recursos utilizados na limpeza e preparação da máquina i após o processamento da tarefa j e imediatamente antes do processamento da tarefa k.

Para este problema, cada possível *setup time* S_{ijk} possui um conjunto de quatro valores. Os dois primeiros especificam a quantidade máxima e mínima de recursos que podem ser associados ao *setup time*. Portanto, R_{ijk}^{+} (R_{ijk}^{-}) denota o máximo (mínimo) de recursos que se pode associar ao *setup time* S_{ijk}. Esses valores são exemplificados na prática através de algumas linhas de produção, cujo *setup time* será usado para a remoção de itens e/ou ferramentas pesadas e, assim, um número mínimo de trabalhadores (recursos) é necessário. Da mesma forma, num determinado momento pode não ser possível atribuir mais recursos devido a restrições físicas e/ou operacionais.

Os outros dois valores disponíveis, para cada possível *setup time*, representam o valor máximo e mínimo que definem o intervalo em que o *setup time* poderá ser considerado. Neste trabalho, a relação entre o *setup time* e o número de recursos associados é considerada linear. Portanto, conforme descrito anteriormente, se o mínimo de recursos (R_{ijk}^{-}) é associado, o *setup time* será o máximo possível (denotado por S_{ijk}^{+}). Caso contrário, se o número máximo de recursos é associado (R_{ijk}^{+}), o *setup time* será o mínimo possível (denotado por S_{ijk}^{-}). A relação entre recursos e *setup time* é apresentada na Figura 2.2.

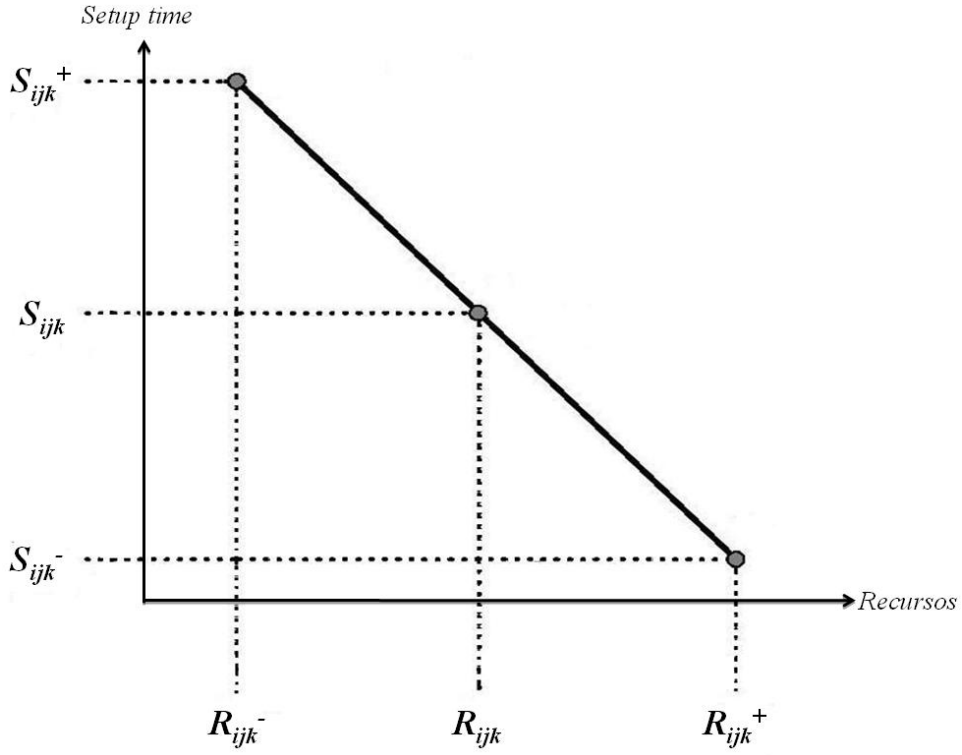


Figura 2.2. Relação entre o *setup time* S_{ijk} e o número de recursos associado R_{ijk}

Portanto, da relação linear descrita acima, deduzimos que a expressão que determina o *setup time* S_{ijk} está em função da quantidade de recursos associados R_{ijk}, conforme detalhado abaixo:

$$S_{ijk} = S_{ijk}^{+} - \frac{S_{ijk}^{+} - S_{ijk}^{-}}{R_{ijk}^{+} - R_{ijk}^{-}}(R_{ijk} - R_{ijk}^{-}) = S_{ijk}^{+} + \frac{S_{ijk}^{+} - S_{ijk}^{-}}{R_{ijk}^{+} - R_{ijk}^{-}}R_{ijk}^{-} - \frac{S_{ijk}^{+} - S_{ijk}^{-}}{R_{ijk}^{+} - R_{ijk}^{-}}R_{ijk} \qquad (1)$$

Usando K_1 para os primeiros termos da expressão e K_2 para o multiplicador de R_{ijk}, temos como resultado a equação da reta que define a relação entre S_{ijk} e R_{ijk}. Além disso, podemos encontrar o coeficiente angular da reta, que indica a intensidade do "declive".

$$S_{ijk} = K_1 - K_2 R_{ijk} \quad (2)$$

Portanto, K_2 é o "declive" apresentado na Figura 2.2 que relaciona R_{ijk} com S_{ijk}. Em outras palavras, K_2 indica quanto o *setup time* S_{ijk} é reduzido para a adição de um recurso. Por exemplo, dados $S_{ijk}^{+} = 20$, $S_{ijk}^{-} = 4$, $R_{ijk}^{+} = 10$ e $R_{ijk}^{-} = 2$, a expressão (2) poderá ser definida por $S_{ijk} = 20 - ((20\text{-}4)\div(10\text{-}2))\times(R_{ijk} - 2) = 24 - 2R_{ijk}$. Ou seja, para cada recurso adicionado, o *setup time* reduzirá em duas unidades.

2.4 Modelagem matemática do problema

Com a definição do *setup time* e sua relação com os recursos, é possível definir um modelo de programação matemática inteira, da mesma forma que foi proposto por RUIZ e ANDRÉS (2007). Nesse modelo, pretende-se descrever o problema de forma realista, com o interesse de definir um critério de otimização que minimize o tempo total de produção (do inglês, *flow time*).

Se denotarmos por C_j o tempo gasto para conclusão da tarefa j, o tempo total de produção, ou *flow time*, será dado por:

$$\sum_{j=1}^{n} C_j \quad (3)$$

Uma observação importante do problema é que ao minimizar a expressão (3), os *setup times* utilizados serão os menores possíveis e, consequentemente, o número de recursos associados, juntamente com seus custos, serão maximizados. Por isso, afirma-se que todas estas medidas são importantes na análise do problema e na sua formulação matemática, já que a minimização do *flow time* acarreta um aumento nos custos do processamento, através da utilização de mais recursos, o que também não é interessante e precisa ser minimizado.

Portanto, o número de recursos associados também precisa ser considerado e minimizado. Considerando $R_{ijk} = 0$, caso a tarefa j não preceda a tarefa k na máquina i e, denotando-se por T_{rec} o número total de recursos utilizados, temos que:

$$T_{rec} = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1, k\neq j}^{n} R_{ijk} \quad (4)$$

Sejam λ e δ os valores que modularizam a importância, ou custo, de cada unidade de recurso e tempo de conclusão, respectivamente. Além disso, seja C_{ij} o tempo de conclusão da tarefa j na máquina i. Dessa forma, a função objetivo do problema considerado neste trabalho pode então ser definida por:

$$\lambda T_{rec} + \delta \sum_{j=1}^{n} C_j \tag{5}$$

Destaca-se que $C_{ij} = 0$ se a tarefa j não for processada na máquina i. A partir desse momento, é possível detalhar a formulação do problema. Abaixo descreve-se as variáveis de decisão do problema.

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

C_{ij} = Tempo de conclusão da tarefa j na máquina i

R_{ijk}= Recursos associados ao *setup time* entre as tarefas j e k, na máquina i

Observa-se que $R_{ijk} = 0$ se a tarefa j não preceder a tarefa k na máquina i. Dessa forma, a função objetivo do problema denotada por Z e apresentada anteriormente, pode ser detalhada como:

$$Z = \lambda \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} R_{ijk} + \delta \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} \tag{6}$$

A função objetivo deste modelo, conforme observado, tenta minimizar a combinação linear do total de recursos associados (T_{rec}) e o tempo total de conclusão das tarefas. No problema abordado neste trabalho, a função objetivo terá algumas restrições para garantir a correta minimização. A seguir são descritas as restrições do modelo matemático do problema.

$$\sum_{i \in M} \sum_{\substack{j \in N \cup \{0\} \\ j \neq k}} X_{ijk} = 1, \qquad k \in N \tag{7}$$

$$\sum_{i \in M} \sum_{\substack{k \in N \\ j \neq k}} X_{ijk} \leq 1, \qquad j \in N \tag{8}$$

$$\sum_{k \in N} X_{i0k} \leq 1, \qquad i \in M \tag{9}$$

$$\sum_{\substack{h \in N \cup \{0\} \\ h \neq k, h \neq j}} X_{ihj} \geq X_{ijk}, \qquad \begin{matrix} j,k \in N, \\ j \neq k, i \in M \end{matrix} \tag{10}$$

$$C_{ik} + V(1 - X_{ijk}) \geq C_{ij} + K_1 - K_2 R_{ijk} + p_{ik}, \qquad \begin{matrix} j \in N \cup \{0\}, k \in N \\ j \neq k, i \in M \end{matrix} \tag{11}$$

$$R_{ijk} \geq R_{ijk}^{-}, \qquad \begin{matrix} j,k \in N, \\ j \neq k, i \in M \end{matrix} \tag{12}$$

$$R_{ijk} \leq R_{ijk}^{+}, \qquad \begin{matrix} j,k \in N, \\ j \neq k, i \in M \end{matrix} \qquad (13)$$

$$C_{i0} = 0, \qquad i \in M \qquad (14)$$

$$C_{ij} \geq 0, \qquad j \in N, i \in M \qquad (15)$$

$$X_{ijk} \in \{0,1\}, \qquad \begin{matrix} j \in N \cup \{0\}, k \in N \\ j \neq k, i \in M \end{matrix} \qquad (16)$$

$$R_{ijk} \in \mathbb{N} \qquad (17)$$

O conjunto de restrições (7) garante que cada tarefa tenha exatamente uma antecessora. Note, que para isso é usada uma tarefa fictícia 0, de tal forma que X_{i0k} ($i \in M$, $k \in N$) possui o valor 1 se a tarefa k é a primeira a ser processada na máquina i, e valor 0 no caso contrário. Inversamente, o conjunto de restrições (8) garante que cada tarefa terá uma tarefa sucessora se não for a última tarefa sequenciada na máquina, e nenhuma tarefa sucessora no caso contrário. O conjunto (9) limita o número de sucessoras das tarefas fictícias em uma para cada máquina, ou seja, as tarefas fictícias 0 podem ter apenas uma tarefa sucessora em cada máquina. Com o conjunto de restrições (10) garante-se que cada tarefa j sequenciada na máquina i tenha uma tarefa antecessora h sequenciada na mesma máquina. Observe que o conjunto de restrições (10) garante que cada tarefa tenha alguma tarefa antecessora, enquanto o conjunto de restrições (7) garante que cada tarefa tenha exatamente uma tarefa antecessora. O conjunto de restrições (11) é primordial para o controle dos tempos de conclusão das tarefas nas máquinas. Basicamente, se uma tarefa k é associada à máquina i após o processamento de j (ou seja, $X_{ijk} = 1$), então o seu tempo de conclusão C_{ik}, deverá ser maior que o tempo de conclusão de j (C_{ij}), somado com *setup time* (entre j e k) e o tempo de processamento de k (p_{ik}). Note que o *setup time* está expresso conforme a equação (2). Se $X_{ijk} = 0$, então a constante V, que possui um valor elevado, torna a restrição redundante. Observe que o valor de V neste caso é uma constante, no entanto, este valor poderia ser variável (V_{ijk}) e em cada caso, definido por uma heurística, por exemplo. Esta heurística deve garantir apenas que V_{ijk} atenda o restrição. Uma questão importante surge se $R_{ijk}^{-} = R_{ijk}^{+}$. Nessa situação, $S_{ijk}^{-} = S_{ijk}^{+} = S_{ijk}$ e, portanto o termo K_1-K_2R_{ijk} no conjunto de

restrições (11) será simplesmente substituído pelo valor S_{ijk}. Os conjuntos de restrições (12) e (13) limitam os possíveis valores de recursos. O conjunto (14) define o tempo de conclusão das tarefas fictícias em 0, enquanto que o conjunto (15) garante que o tempo de conclusão das demais tarefas sejam positivos ou nulos. Por fim, o conjunto de restrições (16) define as variáveis binárias e o (17) indica que a quantidade de recursos deve possuir valores no conjunto de números naturais ($\mathbb{N}$). De forma global, o modelo possui n^2m variáveis binárias, $n(n\text{-}1)m + nm + m$ variáveis contínuas e um total de $n^2m + 3n(n\text{-}1)m + nm + 2n + 2m$ restrições, onde n é o número de tarefas e m o número de máquinas.

A partir desta definição, pode-se afirmar que os dados de entrada do problema são, em primeiro lugar, o número de tarefas (n) e o número de máquinas (m). Além disso, outro dado de entrada é uma matriz de tamanho $n \times m$ com os tempos de processamento que cada tarefa utilizará se for processada na respectiva máquina. Por fim, cada máquina também terá 4 matrizes de tamanho $n \times n$. Duas delas representam o número mínimo e máximo de recursos e as outras duas os valores mínimo e máximo de *setup time* para cada possível sequência formada pelos pares de tarefas, caso elas sejam sequenciadas nesta ordem, na respectiva máquina. Observa-se ainda que nessas matrizes os valores da diagonal principal são nulos.

2.5 Exemplo prático de aplicação do problema

Apesar de ser um problema cotidiano, principalmente nas indústrias de manufatura, o problema de programação de tarefas em máquinas paralelas com *setup times* e recursos (PPTMPSR) somente foi abordado recentemente na literatura, por RUIZ e ANDRÉS (2007), sendo que até o momento nenhum outro trabalho, além desse, foi encontrado.

Além das já citadas indústrias de cerâmica, onde ocorre o PPTMPSR diariamente, podemos citar outras indústrias em que também ocorre o mesmo problema. Por exemplo, nas gráficas de impressão, onde diversas máquinas de impressão diferentes podem realizar todos os pedidos (tarefas). Porém algumas máquinas são especializadas em imprimir em determinados “tipos de papel”, por isso os tempos de processamento dos pedidos podem variar conforme a máquina em que serão executados. Ao final da execução do pedido, a máquina terá que ser limpa e preparada para a execução do próximo pedido, já que o excesso de tinta utilizado terá que ser removido e a matéria-prima, do próximo pedido, disponibilizada nos

respectivos locais. Nestes momentos, a limpeza e preparação das máquinas poderão ser realizadas por mais de um trabalhador (recurso), sendo que em alguns casos um número mínimo e máximo de trabalhadores será especificado, já que a limpeza da máquina pode incluir atividades que necessitam ser executadas simultaneamente e por haver restrições físicas na execução dessas atividades.

Para ilustrar um exemplo do problema, abaixo são apresentados os dados de entrada de um problema com $n = 4$ tarefas e $m = 2$ máquinas.

Tabela 2.1. Tempos de processamento (p_{ij}) para o exemplo do problema

	i	
j	1	2
1	79	45
2	51	78
3	32	27
4	43	90

Fonte: (RUIZ e ANDRÉS, 2007)

Conforme descrito anteriormente, um dos dados de entrada do PPTMPSR é a matriz $n \times m$ com os tempos de processamento que cada tarefa utilizará se for processada na respectiva máquina. Essa matriz, neste exemplo do problema, é apresentada na Tabela 2.1, onde cada tarefa *j* possui um tempo de processamento na máquina *i*. Por exemplo, se a tarefa 3 for processada na máquina 1, o tempo gasto no processamento dessa tarefa será de 32 unidades de tempo, pois na Tabela 2.1 é dado que $p_{13} = 32$.

Na Tabela 2.2 são apresentados para cada máquina, as quatro matrizes de tamanho $n \times n$, que apresentam, respectivamente, os números mínimo e máximo de recursos, e os *setup times* mínimo e máximo. Para facilitar a visualização, condensamos as duas primeiras e as duas últimas matrizes na Tabela 2.2, separando os elementos delas por ponto-e-vírgula (;). Observe que todas as matrizes possuem a diagonal principal com valores nulos. Além disso, alguns elementos das matrizes de recursos possuem o mesmo valor, ou seja, os números de recursos, mínimo e máximo, são iguais. Neste caso, os respectivos valores, mínimo e máximo, do *setup time* também serão iguais. Por exemplo, da Tabela 2.2 é dado que se na máquina 2 ($i = 2$), a tarefa 2 ($j = 2$) for processada antes da tarefa 1 ($k = 1$), então o número de recursos a ser utilizado neste *setup time* será de 2 unidades, já que $R_{221}^{-} = R_{221}^{+} = 2$.

Da mesma forma, o setup time será de 30 unidades de tempo, uma vez que $S_{221}^{-} = S_{221}^{+} = 30$.

Tabela 2.2. Mínimo e máximo recursos e *setup times* (R_{ijk}^{-}, R_{ijk}^{+}, S_{ijk}^{-}, S_{ijk}^{+}) para o exemplo do problema

Recursos, (R_{ijk}^{-} ; R_{ijk}^{+})

	$i = 1$				$i = 2$			
	k				k			
j	1	2	3	4	1	2	3	4
1	0;0	3;4	1;4	3;5	0;0	3;4	3;4	2;4
2	1;5	0;0	1;5	1;3	2;2	0;0	3;4	2;4
3	3;3	1;3	0;0	2;5	2;4	1;5	0;0	2;5
4	1;3	3;4	3;5	0;0	2;4	2;5	2;3	0;0

Setup Times, (S_{ijk}^{-} ; S_{ijk}^{+})

	$i = 1$				$i = 2$			
	k				k			
j	1	2	3	4	1	2	3	4
1	0;0	47;95	28;51	17;63	0;0	34;58	41;52	29;53
2	10;51	0;0	35;58	50;57	30;30	0;0	44;55	31;60
3	28;28	5;57	0;0	18;50	28;99	44;75	0;0	19;71
4	20;89	36;61	22;93	0;0	40;97	23;76	21;83	0;0

Fonte: (RUIZ e ANDRÉS, 2007)

Ao sequenciar a tarefa nas máquinas estará, em outras palavras, definindo a sequência em que as tarefas serão processadas em cada máquina. Neste exemplo, após sequenciarmos, escolhe-se o número de recursos que serão utilizados na limpeza e preparação da máquina, para cada par de tarefas adjacentes que foram sequenciadas na mesma máquina. Através da expressão (1), definida na seção anterior, e com o número de recursos escolhidos, pode-se calcular cada *setup time*.

Note que, ao escolhermos o número de recursos, devemos escolher sempre um valor inteiro definido entre os valores mínimo e máximo. Neste exemplo, iremos sempre utilizar o maior valor inteiro do número médio de recursos, ou seja,

$$R_{ijk} = \left\lceil \frac{R_{ijk}^{+} + R_{ijk}^{-}}{2} \right\rceil$$

Imagine que neste exemplo sequenciamos as tarefas 4 e 2, nesta ordem, na máquina 1, e sequenciamos as tarefas 3 e 1, nesta ordem, na máquina 2. Pelos dados

de entrada do problema (Tabelas 2.1 e 2.2), podemos representar a solução do problema graficamente como na Figura 2.3.

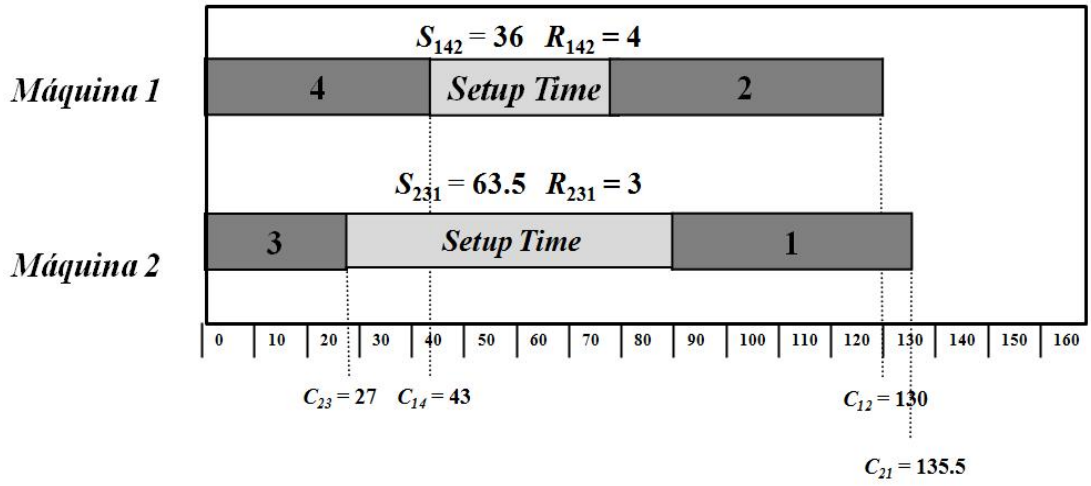


Figura 2.3. Solução do exemplo do problema utilizando o número médio de recursos em cada *setup time*

Na análise da Figura 2.3, observa-se que as tarefas 4 e 3 são as primeiras tarefas processadas, respectivamente, nas máquinas 1 e 2. Por isso os tempos de conclusão coincidem com os respectivos tempos de processamento ($C_{14} = p_{14} = 43$ e $C_{23} = p_{23} = 27$). O número de recursos utilizados no *setup time* entre as tarefas 4 e 2 na máquina 1 é o valor médio entre o máximo e o mínimo. Da tabela 2.2 é obtido que $R_{142}^{-} = 3$ e $R_{142}^{+} = 4$ e, portanto $R_{142} = 3.5 \rightarrow 4$, pois a quantidade de recursos deve ser um valor inteiro. Substituindo esse valor na expressão (1) e observando que $S_{142}^{-} = 36$ e $S_{142}^{+} = 61$ temos que $S_{142} = 36$. Realizando os mesmos procedimentos para o *setup time* entre as tarefas 3 e 1 na máquina 2, obtém-se que $R_{231} = 3$ e $S_{231} = 63.5$. Por fim, note que as tarefas 2 e 1 são as últimas tarefas a serem processadas, respectivamente, nas máquinas 1 e 2, sendo que da Tabela 2.1 obtemos que os tempos de processamento dessas tarefas são $p_{12} = 51$ e $p_{21} = 45$. Com essas informações é possível definir que as tarefas 2 e 1 são finalizadas nos tempos $C_{12} = 130$ e $C_{21} = 135.5$. Observe, também, que as máquinas começam o processamento das tarefas simultaneamente.

Neste trabalho, para representar uma solução do problema, utilizaremos um arranjo de tamanho $n+m-1$ contendo as n tarefas e $m-1$ elementos (-1) utilizados para dividir as tarefas em m grupos, um para cada máquina. Dessa forma, a solução do exemplo demonstrado seria representada pelo arranjo [4, 2, -1, 3, 1]. Note que os grupos de tarefas [4, 2] e [3,1] são processados respectivamente nas máquinas 1 e 2.

Se assumirmos, por exemplo, que $\lambda = 30$ e $\delta = 1$, então é possível calcularmos o valor da função objetivo *Z*. Note que o total de recursos usados neste sequenciamento é dado pela soma da quantidade de recursos utilizados no *setup time* entre as tarefas 4 e 2 ($R_{142} = 4$) com a quantidade de recursos utilizados no *setup time* entre as tarefas 3 e 1 ($R_{231} = 3$), ou seja, o total de recursos neste sequenciamento é 7 (4+3). O tempo de conclusão total é obtido pela soma dos tempos de conclusão das *n* tarefas. Portanto, neste exemplo, esse valor é obtido pela soma: 27+43+130+135.5 = 335.5. Dessa forma, podemos calcular o valor de *Z* por: 30×(7) + 1×(335,5) = 545,5. Neste caso, por utilizar uma heurística construtiva, não se pode garantir que a solução encontrada é a solução ótima.

Após a definição do PPTMPSR, iremos apresentar nos próximos capítulos as heurísticas construtivas, já propostas na literatura, e os métodos heurísticos, baseados em metaheurísticas, que foram propostos e aplicados ao problema.

3 MÉTODOS HEURÍSTICOS CONSTRUTIVOS

Nas últimas décadas, um conjunto de métodos tem sido desenvolvido para resolver problemas de programação de produção. Técnicas de programação matemática tais como técnicas de enumeração do tipo *Branch-and-Bound* e Programação Linear Inteira, têm sido empregadas para a solução ótima do problema. Contudo, tais técnicas não são eficientes em termos computacionais, para problemas de médio e grande porte. Deste modo, muitos métodos heurísticos têm sido propostos para a solução de tais problemas.

Segundo GOMES e NETO (2003), métodos heurísticos, em sua grande maioria, são utilizados para resolver problemas que não podem ser tratados por métodos exatos, tentando reproduzir, muitas vezes, o que é realizado manualmente. Desta forma o objetivo principal de uma heurística é encontrar soluções de boa qualidade, mas com um baixo custo computacional, ou seja, de forma rápida.

É comum que uma solução de boa qualidade seja encontrada, e algumas vezes uma solução ótima é obtida. Para isso, observa-se que quanto maior for o número de informações (restrições) fornecidas às heurísticas, maiores são as chances delas obterem uma solução que satisfaça o problema, logo no início da sua execução.

As heurísticas, em geral, não garantem que a solução encontrada seja a solução ótima, por isso podemos limitar sua busca. Essa limitação pode ser realizada através de um número fixo de iterações, retirada ou relaxamento de restrições, entre outras técnicas.

Portanto, pode-se resumir que um método heurístico é um processo de solução de problemas apoiado em critérios racionais para escolher um bom caminho entre vários possíveis, sem a preocupação de percorrer todas as possibilidades ou atingir a solução ótima. Além disso, os métodos heurísticos são procedimentos mais flexíveis e simples que os métodos exatos de solução, permitindo abordar problemas mais complexos. Os métodos exatos geralmente exigem simplificações nos modelos, tornando-se menos representativos do problema real.

Como já mencionado, os métodos heurísticos não garantem uma solução ótima, sendo qualificados pelo tempo computacional envolvido no seu processamento e na proximidade da solução ótima do problema, já que estes métodos possuem uma relação estrita às características do problema e dos dados fornecidos. Em muitos casos, a pequena diferença entre a solução heurística e a ótima não justifica o grande esforço computacional requerido para obter esta última.

De acordo com SOUZA e MOCCELLIN (2000), os métodos heurísticos podem ser classificados de diversas formas, e uma delas classifica-os em construtivos ou melhorativos.

- **Métodos Construtivos** – Estes métodos constroem uma solução progressivamente. O algoritmo sempre começa com uma solução parcial nula e, a cada passo, um elemento é adicionado à solução parcial. No final do algoritmo, tem-se uma solução para o problema que, para os problemas de programação de tarefas, é obtida:

  i) diretamente a partir da ordenação das tarefas segundo índices de prioridade calculados em função dos tempos de processamento das tarefas;

  ii) escolhendo-se o melhor sequenciamento das tarefas a partir de um conjunto de soluções também obtidas utilizando-se índices de prioridade associados às tarefas; ou

  iii) a partir da geração sucessiva de sequenciamentos parciais das tarefas até a obtenção de uma programação completa através de algum critério de inserção de tarefas.

- **Métodos Melhorativos** – Estes métodos são procedimentos que tentam iterativamente melhorar uma solução, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. Para os problemas de programação de tarefas em máquinas, estes métodos obtém uma solução inicial e, posteriormente, através de algum procedimento iterativo, geralmente envolvendo trocas de posições das tarefas no sequenciamento inicial, busca-se conseguir uma solução melhor que a atual quanto à medida de desempenho escolhida.

Na categoria dos métodos melhorativos destacam-se os procedimentos de busca em vizinhança, tais como *Greedy Randomized Adaptive Search Procedures*

(GRASP) e *Iterated Local Search* (ILS), que têm sido alvo de grande interesse na comunidade científica em função de aplicações bem sucedidas reportadas na literatura. Outra técnica que pode ser considerada do tipo melhorativo, denominada Algoritmo Genético, desperta interesse pela sua capacidade de solução de problemas de natureza combinatória. Essa técnica faz parte da classe dos algoritmos evolucionários.

Os métodos heurísticos GRASP e ILS, conhecidos pelo termo metaheurísticas, consistem de procedimentos de busca em subespaços do espaço de soluções, definidos por estratégias que exploram apropriadamente a topologia de tal subespaço. Assim, as metaheurísticas GRASP e ILS foram aplicadas ao PPTMSPR e serão detalhadas no próximo capítulo.

Nesse capítulo, principalmente nas seções seguintes, serão apresentados os 14 métodos heurísticos construtivos propostos na literatura por RUIZ e ANDRÉS (2007) para o PPTMPSR.

Conforme dito anteriormente, ao final do algoritmo de qualquer método heurístico construtivo tem-se uma solução. Esta solução é obtida iterativamente, de tal forma que em cada iteração é utilizado algum critério, ou regra, para definir a parte da solução que seria construída, ou inserida. Para os problemas de programação de tarefas em máquinas é muito comum que em cada iteração seja inserida uma tarefa. As principais regras, que definem qual será a tarefa a ser inserida, são baseadas na *Shortest Processing Time* (SPT). As regras baseadas na SPT utilizam o tempo de processamento para definir, em cada iteração, qual tarefa (não sequenciada) será adicionada à solução parcial.

Portanto, conforme pode ser visto em PINEDO (2002), as regras baseadas na SPT são eficientes para encontrar soluções de boa qualidade, principalmente em problemas de programação simples, que utilizam o total do tempo de conclusão como critério a ser minimizado. WENG *et al*. (2001) propuseram sete heurísticas, baseadas na SPT, para o problema de programação em tarefas em máquinas paralelas com *setup time*, cujas tarefas possuíam pesos ou prioridades associadas. Por isso, neste caso específico, foi utilizada uma variação da SPT denominada *Weighted Shortest Processing Time* (WSPT).

O fundamento principal das regras baseadas na SPT é que, para o critério do tempo total de conclusão, o tempo de conclusão da primeira tarefa associada a uma máquina pode contribuir, ou interferir, no tempo de conclusão das demais tarefas

associadas a essa mesma máquina. Por exemplo, imagine três tarefas associadas a uma mesma máquina, com tempos de processamento $p_1 = 1$, $p_2 = 9$, $p_3 = 10$. Descartando os *setup times* e recursos entre as tarefas, tem-se que a sequência {1, 2, 3} possui os tempos de conclusão de cada tarefa em 1, 10 e 20, respectivamente, com um tempo de conclusão total de 31. Entretanto, a sequência {3, 2, 1} resulta que os tempos de conclusão de cada tarefa serão 10, 19 e 20, respectivamente, com um tempo de conclusão total de 49. Assim sendo, fica evidente que se sequenciarmos as tarefas em ordem crescente pelo tempo de processamento, teremos o total do tempo de conclusão menor. Portanto, como um dos critérios a ser minimizado no PPTMPSR é o total do tempo de conclusão, RUIZ e ANDRÉS (2007) também propuseram seus métodos heurísticos construtivos baseados na regra SPT.

3.1 *Shortest Processing Time with Setups Resource Assignment* (SPTSA)

O método SPTSA utiliza a regra SPT de forma mais simples possível para construir uma solução. Ao final do procedimento, será obtida uma solução com a sequência e com os recursos associados a cada *setup time*. Abaixo são descritas as principais etapas do método.

1. Para cada tarefa *j* deve-se definir a máquina *i* em que *j* possua o menor tempo de processamento. Armazena-se a tarefa *j*, a máquina i e o valor do tempo de processamento mínimo p_{ij} em um vetor *W* de tamanho *n*.
2. Ordena-se o vetor *W* em ordem crescente do tempo de processamento mínimo de cada tarefa em todas as máquinas.
3. Para $k = 1$ até *n* faça:
   (a) Obtenha de *W*[*k*] a tarefa *j* e a máquina *i* e insira *j* na sequência de *i*.
   (b) Associe recursos para o *setup time* entre a última tarefa sequenciada na máquina *i* e a tarefa *j*.

Após a descrição deste procedimento, uma dúvida final que persiste é sobre a quantidade de recursos que devemos associar em cada *setup time* (passo 3(b)). RUIZ e ANDRÉS (2007) testaram três diferentes abordagens do SPTSA em que sempre atribuía o mínimo, máximo ou o número médio de recursos possíveis para aquele *setup time*. Estas três abordagens são denotadas por $SPTSA^-$, $SPTSA^+$ e $SPTSA^{av}$, respectivamente. O número médio de recursos é calculado pela expressão: $(R_{ijk}^+ + R_{ijk}^-)/2$. Caso esse valor não seja inteiro, será atribuído o maior valor inteiro

correspondente. Observe no procedimento descrito anteriormente que ao associarmos a primeira tarefa a uma máquina, também associamos um número de recursos para o *setup time* inicial. Entretanto, sem perda de generalidade, nestes casos se considera o *setup time* inicial (e os correspondentes recursos) sendo de valor nulo.

Note mais uma vez que no método SPTSA é utilizado apenas o tempo de processamento como critério para definir qual será a próxima tarefa a ser sequenciada.

3.2 *Shortest Processing and Setup Time with Setups Resource Assignment* (SPSTSA)

O método SPSTSA é muito similar ao método SPTSA, a única diferença é que as médias dos possíveis *setup times* são consideradas juntamente com os tempos de processamentos mínimos, para assim definir a ordem de sequenciamento das tarefas nas máquinas. Com isso, os dois primeiros passos do algoritmo SPTSA foram modificados, conforme descrito abaixo.

1. Para cada tarefa j deve-se calcular, em cada máquina, o valor da soma do tempo de processamento com a média de todos os possíveis setup times. O menor valor será considerado. Pode-se escrever esse cálculo através da expressão I como segue:

$$I_j = \min_{i \in M} \left(p_{ij} + \frac{1}{n-1} \sum_{k=1, k \neq j}^{n} \frac{S_{ijk}^{+} + S_{ijk}^{-}}{2} \right)$$

   Seja i a máquina em que o menor valor de I_j foi obtido. Armazena-se a tarefa j, a máquina i e o valor de I_j em um vetor W de tamanho n.

2. Ordena-se o vetor W em ordem crescente pelo valor I_j.

Os demais passos do procedimento SPSTSA são idênticos ao do SPTSA. Observe que ao contrário do SPTSA, o método SPSTSA também considera o *setup time* esperado entre uma tarefa associada a uma determinada máquina e todas as demais tarefas. Fornece-se assim, para cada tarefa, uma estimativa do tempo de processamento juntamente com o *setup time* esperado após o seu processamento, que por sua vez, irá influenciar o tempo de conclusão das tarefas sucessoras.

Da mesma forma que o método SPTSA, o método SPSTSA possui três abordagens diferentes quanto à quantidade de recursos (mínimo, máximo ou valor

médio) a ser realmente associada em cada *setup time*. As três abordagens são denotadas, respectivamente, por SPSTSA^{-}, SPSTSA^{+} e SPSTSA^{av}.

3.3 *Dynamic Job Assignment with Setups Resource Assignment* (DJASA)

Os dois métodos apresentados anteriormente possuem um inconveniente em comum. Esse inconveniente é que num problema de programação de tarefas em máquinas paralelas não-relacionadas, o tempo de processamento mínimo (SPTSA), ou o valor do índice I (SPSTSA), podem ser iguais para duas tarefas na mesma máquina i. Portanto, se adicionarmos essas tarefas na máquina i, numa determinada ordem, pode-se não encontrar uma solução de boa qualidade. Assim sendo, talvez seja mais vantajoso associarmos uma das tarefas à máquina i e a outra a uma máquina em que o valor do tempo de processamento mínimo, ou do índice I, seja um pouco maior. Dessa forma, tenta-se que o total do tempo de conclusão seja menor. O método proposto nesta seção, denominado DJASA, resolve este problema de forma dinâmica, considerando em cada iteração todas as tarefas não sequenciadas em todas as possíveis máquinas. O algoritmo do método DJASA é descrito abaixo:

1. Adicione todas as tarefas numa lista de tarefas não sequenciadas ρ.
2. Enquanto $\rho \neq \varnothing$ faça:
   (a) Para cada tarefa k em ρ e para cada máquina $i \in M$ faça:
      i. Temporariamente atribua recursos ao *setup time* entre a tarefa k e a tarefa anteriormente associada à máquina i.
      ii. Sequencie temporariamente a tarefa k na máquina i.
      iii. Calcule o valor da função objetivo Z após a associação dos recursos e da tarefa.
   (b) Seja j e l a tarefa e a máquina, respectivamente, que resultaram no menor incremento na função objetivo.
   (c) Atribua os recursos ao *setup time* entre a tarefa j e a tarefa anteriormente sequenciada na máquina l.
   (d) Sequencie a tarefa j na máquina l.
   (e) Remova a tarefa j da lista ρ.

Da mesma forma que nos métodos SPTSA e SPSTSA, deve-se decidir sobre a quantidade de recursos que serão atribuídos. Assim sendo, três métodos são derivados em função do número de recursos atribuídos (mínimo, máximo ou o valor

médio) nos passos 2(a)i e 2(c). Os métodos derivados, são denotados por DJASA^{-}, DJASA^{+} e DJASA^{av}, respectivamente.

Para os métodos SPTSA e SPSTSA, a complexidade no pior dos casos será $O(nlog\ n)$ já que a operação mais custosa nestes métodos é a ordenação dos tempos de processamento e do índice I, respectivamente. Para o método DJASA, um total de $[n(n+1)/2]m$ inserções de tarefas são realizadas, o que resulta numa complexidade $O(n^2m)$.

3.4 Atribuição otimizada de recursos

A partir do momento que as tarefas foram sequenciadas nas máquinas paralelas e não relacionadas, e desde que essa sequência seja conhecida, é possível realizar algumas observações sobre a etapa de atribuição de recursos aos *setup times* utilizada nesses métodos.

- Encurtando os *setup times* utilizados entre as primeiras tarefas sequenciadas de cada máquina, é possível que se tenha um grande efeito na redução do total do tempo de conclusão.
- Poupando recursos (expandindo os *setup times*) entre as tarefas sequenciadas no final de cada máquina, tem-se um efeito menor sobre o total do tempo de conclusão.
- Os *setup times* mais interessantes são aqueles que possuem um elevado declive K_2, conforme descrito na expressão (2).

Esta última observação é particularmente interessante, já que um declive K_2 elevado indica que grandes reduções no setup time são obtidas através da atribuição de recursos adicionais. Dada a linearidade da relação entre S_{ijk} e R_{ijk}, a mesma redução é obtida para cada recurso adicionado no intervalo $[R_{ijk}^{-};\ R_{ijk}^{+}]$. Como resultado, o processo de atribuição de recursos pode ser otimizado se seguir as seguintes etapas:

1. Para cada máquina i e para cada tarefa k sequenciada em i, com exceção da última, faça
   (a) Seja j a tarefa antecessora da tarefa k na máquina i ou 0 se a tarefa k é a primeira tarefa sequenciada na máquina i.
   (b) Seja h o número de tarefas sucessoras de j sequenciadas na máquina i.
   (c) A quantidade de recursos R_{ijk} atribuída ao *setup time* entre as tarefas j e k na máquina i é re-atribuída de acordo com a expressão abaixo:

$$R_{ijk} = \begin{cases} R_{ijk}^{+}, \text{ se } K_2 \times h \times \delta > \lambda \\ R_{ijk}^{-}, \text{ caso contrário} \end{cases} \quad (17)$$

A expressão acima pode ser explicada da seguinte forma: se é adicionado um recurso, a um determinado *setup time* do sequenciamento realizado, tem-se a redução de K_2 unidades de tempo do *setup time*, conforme descrito no capítulo anterior. Consequentemente, o tempo de conclusão de cada uma das h tarefas (sequenciadas naquela máquina após esse *setup time*) terá uma redução de K_2 unidades de tempo, proporcionando assim uma diminuição do total dos tempos de conclusão em $K_2 \times h$ unidades de tempo. Com relação ao valor de Z, cada unidade de tempo possui um custo δ, o que nos leva a afirmar que a adição de um recurso ao *setup time* gera uma economia em Z de $K_2 \times h \times \delta$ unidades. Se essa economia for maior que o custo do recurso adicionado (λ), tem-se que a adição de mais um recurso a esse *setup time* sempre proporciona uma redução no valor da função objetivo Z e, dessa forma, o número máximo de recursos permitido para esse *setup time* é o valor que proporciona a maior redução no valor de Z. No caso contrário, ou seja, quando o custo de adição de um recurso for maior que a redução que essa adição gera no custo do total do tempo de conclusão, então quanto menor for o número de recursos atribuídos, maior será a redução no valor de Z. Assim, fica evidente que essa re-atribuição de recursos aos *setup times* é a ideal, ou em outras palavras, é a atribuição ótima.

O procedimento de atribuição ótima de recursos, detalhada nesta seção, é muito simples. Se forem conhecidos os dados de entrada do problema, juntamente com o sequenciamento realizado em cada máquina, é possível realizar a re-atribuição, ou atribuição otimizada, utilizando apenas $O(n)$ etapas. Este procedimento pode ser aplicado também em sequenciamentos realizados pelos métodos SPTSA, SPSTSA e DJASA. No caso do SPTSA e SPSTSA, onde cada um possui três abordagens diferentes quanto à atribuição de recursos, mas o mesmo sequenciamento é obtido nas três abordagens, tem-se que apenas a atribuição original é modificada e o sequenciamento permanece inalterado. Portanto, estes métodos, ao utilizarem o procedimento de atribuição otimizada de recursos, não importando qual abordagem foi utilizada no sequenciamento, serão denotados por SPTSA* e SPSTSA*, respectivamente. Note que para o método DJASA as tarefas são sequenciadas de forma dinâmica e a forma como os recursos são atribuídos durante o sequenciamento pode influenciar na definição da sequência final, ou seja, essa atribuição de recursos

durante o sequenciamento é independente da re-atribuição otimizada realizada ao final do sequenciamento. Portanto, além das três abordagens DJASA (DJASA$^-$, DJASA$^+$ e DJASA^{av}), temos mais três abordagens resultantes da aplicação do procedimento de re-atribuição otimizada de recursos a cada um deles. Estas abordagens são denotadas por: DJASA$^-$*, DJASA$^+$* e DJASA^{av}*.

3.5 Exemplo de Aplicação

Ilustra-se, agora, um exemplo de aplicação dos métodos heurísticos DJASA^{av} e DJASA^{av}*, para um pequeno problema que também foi utilizado na seção 2.5, e que consiste em sequenciar 4 tarefas (n = 4) em 2 máquinas (m = 2). Os tempos de processamento e os mínimos e máximos recursos e *setup times* já foram exibidos na Tabela 2.1 e na Tabela 2.2. Para facilitar a compreensão, abaixo apresentamos novamente essas tabelas, respectivamente nas tabelas 3.1 e 3.2.

Tabela 3.1. Tempos de processamento (p_{ij}) para o exemplo do problema

	i	
j	1	2
1	79	45
2	51	78
3	32	27
4	43	90

Fonte: (RUIZ e ANDRÉS, 2007)

A partir desse instante, será aplicada a regra dinâmica DJASA^{av}. Para este exemplo, considera-se λ = 30 e δ = 1, ou seja, o custo (ou importância) de cada unidade de recurso utilizado é 30 vezes maior que o custo de uma unidade de tempo gasto. Além disso, vale observar que a quantidade de recursos atribuído deve ser um valor inteiro. Inicialmente, nenhuma tarefa está sequenciada, ou seja, ρ = {1, 2, 3, 4}. Se assumirmos que os *setup times* iniciais são nulos, a tarefa que possuir o menor tempo de processamento em todas as máquinas será sequenciada. Da Tabela 2.1, nota-se que o menor tempo de processamento corresponde à tarefa 3 na máquina 2 (p_{23} = 27). Portanto, a tarefa 3 será a primeira tarefa sequenciada na máquina 2. Assim sendo, remove-se a tarefa 3 de ρ.

Neste momento serão consideradas todas as tarefas não sequenciadas, ρ = {1, 2, 4}, e as máquinas 1 e 2. Para isso, será necessário a utilização dos valores fornecidos como dados de entrada na tabela 3.2.

Tabela 3.2. Mínimo e máximo recursos e *setup times* (R_{ijk}^-, R_{ijk}^+, S_{ijk}^-, S_{ijk}^+) para o exemplo do problema

Recursos, (R_{ijk}^- ; R_{ijk}^+)

	$i = 1$				$i = 2$			
	k				k			
j	1	2	3	4	1	2	3	4
1	0;0	3;4	1;4	3;5	0;0	3;4	3;4	2;4
2	1;5	0;0	1;5	1;3	2;2	0;0	3;4	2;4
3	3;3	1;3	0;0	2;5	2;4	1;5	0;0	2;5
4	1;3	3;4	3;5	0;0	2;4	2;5	2;3	0;0

Setup Times, (S_{ijk}^- ; S_{ijk}^+)

	$i = 1$				$i = 2$			
	k				k			
j	1	2	3	4	1	2	3	4
1	0;0	47;95	28;51	17;63	0;0	34;58	41;52	29;53
2	10;51	0;0	35;58	50;57	30;30	0;0	44;55	31;60
3	28;28	5;57	0;0	18;50	28;99	44;75	0;0	19;71
4	20;89	36;61	22;93	0;0	40;97	23;76	21;83	0;0

Fonte: (RUIZ e ANDRÉS, 2007)

Abaixo todas as possibilidades de sequenciamento:

- Tarefa 1, Máquina 1: $C_{11} = C_{10} + S_{101} + p_{11} = 0 + 0 + 79 = 79$. O valor da função objetivo Z será incrementado em 79 unidades.
- Tarefa 1, Máquina 2: Recursos atribuídos: $(R_{231}^+ + R_{231}^-)/2 = 3$ que corresponde a $S_{231} = 63.5$. $C_{21} = C_{23} + S_{231} + p_{21} = 27 + 63.5 + 45 = 135.5$. O valor da função objetivo Z será incrementado em $30 \times 3 + 135.5 = 225.5$ unidades.
- Tarefa 2, Máquina 1: $C_{12} = C_{10} + S_{102} + p_{12} = 0 + 0 + 51 = 51$. O valor da função objetivo Z será incrementado em 51 unidades.
- Tarefa 2, Máquina 2: Recursos atribuídos: $(R_{232}^+ + R_{232}^-)/2 = 3$ que corresponde a $S_{232} = 59.5$. $C_{22} = C_{23} + S_{232} + p_{22} = 27 + 59.5 + 78 = 164.5$. O valor da função objetivo Z será incrementado em $30 \times 3 + 164.5 = 254.5$ unidades.
- Tarefa 4, Máquina 1: $C_{14} = C_{10} + S_{104} + p_{14} = 0 + 0 + 43 = 43$. O valor da função objetivo Z será incrementado em **43** unidades.

- Tarefa 4, Máquina 2: Recursos atribuídos: $(R_{234}^{+} + R_{234}^{-})/2 = 3.5 \rightarrow 4$ que corresponde a $S_{234} = 36.3$. $C_{24} = C_{23} + S_{234} + p_{24} = 27 + 36.3 + 90 = 153.3$. O valor da função objetivo Z será incrementado em $30 \times 4 + 153.3 = 273.3$ unidades.

O menor acréscimo na função objetivo Z é obtido se sequenciarmos a tarefa 4 na máquina 1. Com isso, sequenciamos a tarefa 4 na máquina 1, que será a primeira tarefa sequenciada na máquina 1. Após isso, a tarefa 4 é removida da lista ρ.

Na próxima etapa, considera-se todas as tarefas não sequenciadas, $\rho = \{1, 2\}$, e novamente as máquinas 1 e 2.

- Tarefa 1, Máquina 1: Recursos atribuídos: $(R_{141}^{+} + R_{141}^{-})/2 = 2$ que corresponde a $S_{141} = 54.5$. $C_{11} = C_{14} + S_{141} + p_{11} = 43 + 54.5 + 79 = 176.5$. O valor da função objetivo Z será incrementado em $30 \times 2 + 176.5 = 236.5$ unidades.
- Tarefa 1, Máquina 2: Recursos atribuídos: $(R_{231}^{+} + R_{231}^{-})/2 = 3$ que corresponde a $S_{231} = 63.5$. $C_{21} = C_{23} + S_{231} + p_{21} = 27 + 63.5 + 45 = 135.5$. O valor da função objetivo Z será incrementado em $30 \times 3 + 135.5 =$ **225.5** unidades.
- Tarefa 2, Máquina 1: Recursos atribuídos: $(R_{142}^{+} + R_{142}^{-})/2 = 3.5 \rightarrow 4$ que corresponde a $S_{142} = 36$. $C_{12} = C_{14} + S_{142} + p_{12} = 43 + 36 + 51 = 130$. O valor da função objetivo Z será incrementado em $30 \times 4 + 130 = 250$ unidades.
- Tarefa 2, Máquina 2: Recursos atribuídos: $(R_{232}^{+} + R_{232}^{-})/2 = 3$ que corresponde a $S_{232} = 59.5$. $C_{22} = C_{23} + S_{232} + p_{22} = 27 + 59.5 + 78 = 164.5$. O valor da função objetivo Z será incrementado em $30 \times 3 + 164.5 = 254.5$ unidades.

Portanto, conforme se pode observar, o menor acréscimo na função objetivo Z é obtido se sequenciarmos a tarefa 1 na máquina 2. A tarefa 1 será a segunda tarefa sequenciada na máquina 2, sendo que foram atribuídas 3 unidades de recursos ao *setup time* S_{231}. Após o sequenciamento, a tarefa 1 é removida da lista ρ.

Por fim, considera se a última tarefa não sequenciada, $\rho = \{2\}$, nas máquinas 1 e 2.

- Tarefa 2, Máquina 1: Recursos atribuídos: $(R_{142}^{+} + R_{142}^{-})/2 = 3.5 \rightarrow 4$ que corresponde a $S_{142} = 36$. $C_{12} = C_{14} + S_{142} + p_{12} = 43 + 36 + 51 = 130$. O

valor da função objetivo Z será incrementado em 30×4 + 130 = **250** unidades.

- Tarefa 2, Máquina 2: Recursos atribuídos: $(R_{212}^{+} + R_{212}^{-})/2 = 3.5 \rightarrow 4$ que corresponde a $S_{212} = 34$. $C_{22} = C_{21} + S_{212} + p_{22} = 135.5 + 34 + 78 = 247.5$. O valor da função objetivo Z será incrementado em 30×4 + 247.5 = 367.5 unidades.

Dessa forma, na última etapa do procedimento DJASA^{av}, a tarefa 2 é a segunda tarefa sequenciada na máquina 1, sendo que foram atribuídas 4 unidades de recursos ao *setup time* S_{142}. Após a remoção da tarefa 2 de ρ, tem-se que $\rho = \varnothing$, que por sua vez é o critério de parada do procedimento. No total foram atribuídos 4 + 3 = 7 recursos. O tempo de conclusão total é 43 + 130 + 27 +135.5 = 335.5. Assim, o valor da função objetivo será: 30×7 + 335.5 = 545.5, da mesma forma que foi apresentado na Figura 2.3.

Neste momento será aplicado, no sequenciamento obtido pelo método DJASA^{av}, o procedimento de re-atribuição ótima de recursos (DJASA^{av}*). No sequenciamento deste exemplo há apenas dois *setup times*. O $S_{142} = 36$ com $R_{142} = 4$ recursos e $S_{231} = 63.5$ com $R_{231} = 3$ recursos. Lembra-se que $\lambda = 30$, $\delta = 1$ e que cada um destes *setup times* possuem apenas uma tarefa ($h = 1$), a ser processada, após a sua execução. Portanto:

- *Setup time* na máquina 1 entre as tarefas 4 e 2 ($K_2 = 25$): Assim sendo, $K_2 \times 1 \times 1 < 30$. Portanto, R_{142} é re-atribuido para $R_{142}^{-} = 3$. O valor da função objetivo Z sofre uma redução de 5 unidades ($\lambda - K_2$).
- *Setup time* na máquina 2 entre as tarefas 3 e 1 ($K_2 = 35.5$): Ou seja, $K_2 \times 1 \times 1 > 30$. Portanto, R_{231} é re-atribuido para $R_{231}^{+} = 4$. O valor da função objetivo Z sofre uma redução de 5.5 unidades ($K_2 - \lambda$).

Com o resultado final da re-atribuição ótima de recursos, o novo valor da função objetivo é 545.5 - 5 - 5.5 = 535. O sequenciamento final pode ser representado graficamente como na Figura 3.1.

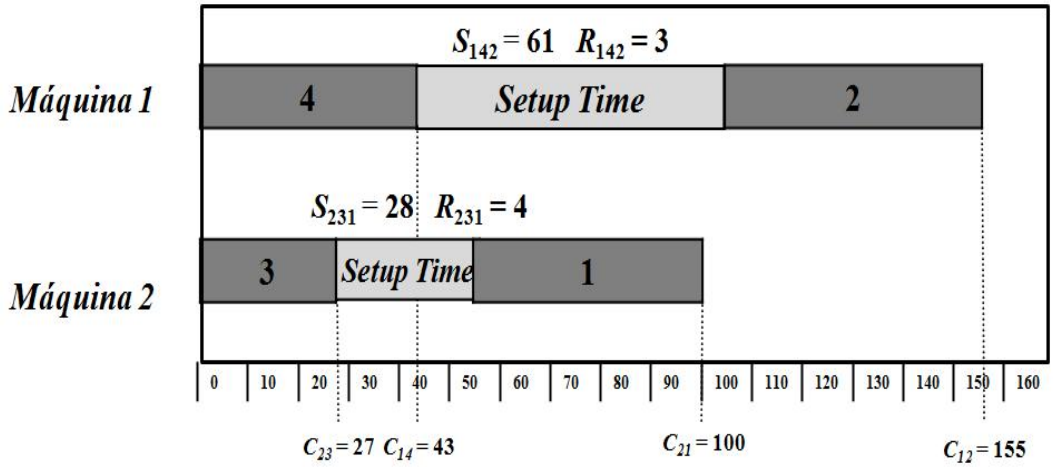


Figura 3.1. Solução do exemplo do problema utilizando o método heurístico construtivo DJASA^{av} e o procedimento de re-atribuição ótima de recursos (DJASA^{av}*)

Após a definição das regras utilizadas nos métodos heurísticos construtivos, propostos por RUIZ e ANDRÉS (2007), e do procedimento de atribuição ótima dos recursos, é possível afirmar que foram apresentados 14 métodos, sendo que quatro deles utilizam a regra SPTSA (SPTSA^{-}, SPTSA^{+}, SPTSA^{av} e SPTSA*), outros quatro métodos utilizam a regra SPSTSA (SPSTSA^{-}, SPSTSA^{+}, SPSTSA^{av} e SPSTSA*) e, por fim, os seis métodos restantes utilizam a regra DJASA (DJASA^{-}, DJASA^{+}, DJASA^{av}, DJASA^{-}*, DJASA^{+}* e DJASA^{av}*). RUIZ e ANDRÉS (2007) realizaram diversos testes nos 14 métodos e os resultados mostram que o método DJASA^{+}* se destacou dos demais, por fornecer os melhores resultados.

Dessa forma, no desenvolvimento dos métodos melhorativos, que serão detalhados no próximo capítulo, sempre que for necessária a construção de uma solução, o método DJASA^{+}* será utilizado.

4 MÉTODOS HEURÍSTICOS PROPOSTOS

No capítulo anterior foram detalhados os métodos heurísticos construtivos propostos na literatura. Dois desses métodos utilizam a regra SPT como critério na escolha da tarefa que será adicionada na solução, enquanto que o último método evita alguns inconvenientes que podem ocorrer com a utilização da regra SPT, e por isso faz a associação de recursos e de *setup time* dinamicamente. Na ocasião foi dito que os métodos heurísticos podem ser divididos em duas categorias: Construtivos e Melhorativos. Neste capítulo se dará uma especial atenção em detalhar os métodos heurísticos melhorativos, propostos neste trabalho, para o PPTMPSR.

Dentre os métodos melhorativos destacam-se os procedimentos também conhecidos como metaheurísticas. Estes métodos emergiram na década de 1970, como sendo um novo tipo de algoritmo de aproximação. Basicamente, as metaheurísticas são uma tentativa de combinar métodos heurísticos básicos em um *framework* de alto nível, explorando desta forma um espaço de busca de forma eficiente e efetiva. O termo metaheurística foi introduzido por GLOVER (1986) e vem da composição de duas palavras gregas: Heurística deriva do verbo grego *heuriskein* que significa "procurar", enquanto meta significa "em um nível superior".

Embora não haja um consenso exato sobre a definição de metaheurística, o conceito mais aceito nos meios acadêmicos foi desenvolvido por GLOVER e KOCHENBERGER (2002), que afirmam que uma metaheurística é um conjunto de conceitos que podem ser utilizados para definir métodos heurísticos aplicáveis a um extenso conjunto de diferentes problemas. Em outras palavras, uma metaheurística pode ser vista como uma estrutura algorítmica geral que pode ser aplicada a diferentes problemas de otimização, de forma tal, que com poucas modificações, possa ser adaptada a um problema específico.

As metaheurísticas dividem-se em duas categorias: metaheurísticas de busca local (do inglês, *Local Search Metaheuristics*-LSMs) e algoritmos evolucionários (do inglês, *Evolutionary Algorithms*-EAs). Uma busca local inicia com uma solução inicial e, a cada etapa de busca, a solução atual é substituída por outra (normalmente,

a melhor) solução melhorada e identificada na vizinhança. Normalmente, LSMs dispõem de mecanismo para diversificar a busca, fugindo de ótimos locais e explorando todo o espaço de soluções. Já os EAs fazem uso de uma população de soluções previamente geradas. A cada iteração do processo de busca, a população inteira ou uma parte da população são substituídas por indivíduos recentemente gerados e melhores (ALBA *et al.*, 2005). Nesta categoria, as metaheurísticas também utilizam mecanismos de diversificação, que garantem uma melhor exploração do espaço de soluções.

Alguns dos principais métodos de solução conhecidos na categoria LSMs são: *Simulated Annealing* (KIRKPATRICK *et al.*, 1983), *Tabu Search* (GLOVER, 1989), *Greedy Randomized Adaptive Search Procedures* (FEO e RESENDE, 1995), *Variable Neighborhood Search* (MLADENOVIC e HANSEN, 1997), *Iterated Local Search* (LOURENÇO *et al.*, 2003); enquanto na categoria EAs: Algoritmos Genéticos (BÄCK *et al.*, 1997), Estratégias Revolucionárias (BÄCK *et al.*, 1997), *Genetic Programming* (BÄCK *et al.*, 1997), *Ant Colonies* (DORIGO, 1992), *Estimation of Distribution Algorithms* (MUHLENBEIN *et al.*, 1999), *Scatter Search* (LÓPEZ *et al.*, 2006). Outros métodos são também esboçados sob as denominações de metaheurísticas heterogêneas e otimização multiobjetivo paralela.

Neste trabalho, principalmente nas subseções seguintes, serão detalhadas as metaheurísticas *Greedy Randomized Adaptive Search Procedures* (GRASP) e *Iterated Local Search* (ILS), juntamente com o procedimento de intensificação na busca de soluções conhecido como *Path Relinking*. Estes procedimentos foram aplicados ao PPTMSPR e os resultados são apresentados no capítulo 5. Nas seções 4.3 e 4.5 são descritos, respectivamente, o procedimento de busca local e o critério de parada utilizados em ambas metaheurísticas.

4.1 Metaheurística GRASP

A metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) foi proposta por FEO e RESENDE (1995), e consiste num método iterativo, em que a cada iteração há um reinício. Desta forma, se diz que o GRASP é um método iterativo probabilístico, onde a cada iteração é obtida uma solução para o problema, e por isso possui múltiplas partidas. Cada iteração do GRASP consiste de duas fases: uma fase de construção de uma solução viável e uma fase de busca local, na qual se procura melhorar a qualidade da solução construída na fase anterior (FEO e

RESENDE, 1995; RESENDE e RIBEIRO, 2003). Na Figura 4.1 são apresentadas as etapas do método GRASP para um problema de minimização da função objetivo *Z*.

```
GRASP (α)
1    Z_min ← +∞;
2    enquanto não CritérioParada faça
3       s1 ← Construção_Solução(α);
4       s2 ← Busca_Local(s1);
5       se Z(s2) < Z_min então
6          s ← s2;
7          Z_min ← Z(s2);
8       fim-se
9    fim-enquanto;
10   retorne s;
fim;
```

Fonte: (FEO e RESENDE, 1995)

Figura 4.1. Pseudocódigo genérico da metaheurística GRASP

O GRASP pode ser visto como um método heurístico que utiliza de boas características dos algoritmos puramente gulosos e dos procedimentos aleatórios na fase de construção de soluções viáveis.

A melhor solução, a que possui o menor valor da função objetivo *Z* calculada dentre todas as iterações do método é retornada como resultado. Observe na Figura 4.1 que os únicos parâmetros a serem definidos no método GRASP são o parâmetro de aleatoriedade α e o critério de parada.

Na maioria das aplicações, o critério de parada é baseado no número máximo de iterações. Podem-se definir outros critérios, como exemplo, estabelecer um número limite de iterações (ou um limite de tempo) em que o procedimento poderá ficar sem encontrar melhores soluções ou ainda estabelecer um tempo máximo de execução (RANGEL *et al.*, 2000).

Note, na Figura 4.1, que o método GRASP, enquanto não satisfizer o critério de parada, repetidamente constrói uma solução *s*1 (passo 3) e esta solução é melhorada por um procedimento de busca local (passo 4). Sempre é armazenada a melhor solução encontrada até o momento (passos 5 a 7), que ao final do procedimento será definida como resultado do método (passo 10).

O parâmetro de entrada do método GRASP, denotado como parâmetro de aleatoriedade α, é utilizado na fase de construção de soluções viáveis. Nesta fase uma solução viável para o problema será construída e em seguida aplicado o procedimento de busca local a esta solução. A eficiência do procedimento de busca

local será melhor se a solução construída for uma solução de boa qualidade. Portanto, na fase de construção é interessante a utilização de métodos gulosos que favoreçam a construção de soluções de boa qualidade. No entanto, como o GRASP é um método iterativo, um fator de aleatoriedade precisa estar presente para garantir a variedade de soluções construídas a cada iteração.

Este componente probabilístico do GRASP é definido pelo parâmetro de aleatoriedade α, de tal forma que $0 \leq \alpha \leq 1$. As soluções mais gulosas são construídas com valores pequenos de α, enquanto que para valores maiores de α, as soluções são construídas de forma mais aleatória.

Nas subseções seguintes serão apresentados a fase de construção de uma solução viável implementada neste trabalho e o método GRASP reativo utilizado para o auto-ajuste do parâmetro de aleatoriedade α.

4.1.1 Construção de soluções viáveis

Segundo MELO e MARTINHON (2004), na fase construtiva do GRASP a solução é construída iterativamente, um elemento por vez. A cada iteração, o próximo elemento a ser adicionado é determinado pela ordenação de todos os elementos numa lista de candidatos LC, isto é, todos que podem ser adicionados à solução, respeitando uma função gulosa qualquer. Esta função estima os benefícios na escolha de cada elemento. O procedimento é adaptativo, pois os benefícios associados a cada elemento são atualizados a cada iteração da fase de construção e refletem as mudanças trazidas pela seleção do elemento anterior.

O componente probabilístico do GRASP é caracterizado pela escolha aleatória de um dos melhores candidatos de LC, não necessariamente do melhor. A lista dos melhores candidatos é denotada por Lista Restrita de Candidatos (LRC). O GRASP possui esse componente probabilístico justamente em vista da escolha aleatória em LRC, uma vez que num método completamente guloso ($\alpha = 0$) sempre seria selecionado o elemento que proporcionasse o maior benefício. Esta técnica de escolha permite que diferentes soluções sejam geradas a cada iteração do GRASP.

O parâmetro de aleatoriedade α é usado para determinar o tamanho de LRC, de forma que quanto maior o valor de α, maior será o tamanho de LRC, e consequentemente mais elementos farão parte desta lista, tornando assim a escolha do próximo elemento a ser adicionado na solução mais aleatória. Contrariamente, quanto menor for o valor de α, menor será o tamanho de LRC, limitando a escolha do

próximo elemento a ser adicionado entre aqueles que fornecem os melhores benefícios, da mesma forma que os métodos gulosos.

Alguns métodos utilizam a LRC de tamanho fixo, porém, segundo SERRA e MOURA (2006), estes métodos apresentam bons resultados apenas quando o tamanho da LRC representa algum percentual significativo do total de elementos, e mesmo assim com alguma degeneração ao final do processo iterativo. Este fato justifica o uso, neste trabalho, do parâmetro α ao invés das LRCs de tamanho fixo.

Para o PPTMPSR, na fase de construção do GRASP, uma solução (sequência de tarefas) é construída iterativamente, iniciando-se com uma sequência vazia, e a cada iteração é adicionada uma única tarefa na sequência parcial. Na escolha da tarefa a ser adicionada, é definida a LC com todas as tarefas ainda não sequenciadas. Para cada tarefa dessa lista são feitas simulações de inclusão da tarefa nas máquinas da sequência parcial, da mesma forma que são realizadas no método heurístico construtivo DJASA$^+$. As tarefas de LC são então ordenadas de forma crescente pelo incremento que elas proporcionarão ao valor da função objetivo *Z*. No método DJASA$^+$ sempre é incluída a tarefa que apresentará o menor incremento ao valor da função objetivo.

Dessa forma, no algoritmo de construção implementado neste trabalho, ao invés de escolher a primeira tarefa, é escolhida aleatoriamente uma tarefa, dentre as η primeiras tarefas de LC, que por sua vez formam a LRC, para ser adicionada na sequência parcial. O valor de η depende do parâmetro α e é definido pela expressão: MAX(1, $\alpha \times |LC|$). Na Figura 4.2 é exibido o algoritmo da fase de construção do GRASP.

```
Construção_Solucão(α)
1   LC ← {t1,...,tn};    // Lista com todas as tarefas
2   Sequência ← { };
3   enquanto ( | Sequência | < n ) faça
4      Ordena_LC( ); // Ordena pelo incremento mínimo que cada tarefa dará em Z
5      η ← MAX(1, ⌊α× | LC |⌋);
6      i ← Random(1, η);
7      Sequência ← Sequência ∪ {ti};
8      LC ← LC - {ti};
9   fim-enquanto;
10  retorne Sequência;
fim;
```

Figura 4.2. Algoritmo da fase de construção do GRASP

Após a construção da solução, é aplicado na solução (sequência de tarefas) construída o método de re-atribuição ótima de recursos, explicado na seção 3.4 do capítulo anterior, já que a comparação entre as soluções será baseada na função objetivo e é interessante que este valor seja o menor possível para aquela sequência. Além disso, o procedimento de busca local aplicado à solução construída pode exigir um tempo maior se a busca partir de uma solução inicial qualquer. Conforme constatado empiricamente (RUIZ *et. al.*, 2006), o desempenho da busca local é dependente da qualidade da solução inicial, de forma tal, que o tempo gasto pela busca local pode ser diminuído através do uso de uma fase de construção que gere uma boa solução inicial. Obviamente, uma estrutura de dados eficiente e uma implementação cuidadosa são importantes.

A dificuldade desta fase consiste em definir o valor ideal para o parâmetro de aleatoriedade α, já que para o PPTMPSR o tamanho de LC está relacionado com o número de tarefas n, e cada instância do problema possui diferente número de tarefas. Em outras palavras, para instâncias do problema com poucas tarefas não é interessante que o valor de α seja pequeno, pois poderia tornar o procedimento guloso demais. Ao contrário, para instâncias do problema com muitas tarefas, se o valor de α for grande poderá tornar o procedimento muito aleatório.

Na tentativa de ajudar na definição do valor de α para o PPTMPSR, a solução encontrada foi a utilização do método GRASP reativo, que possui um mecanismo de auto-ajuste dos valores de α, conforme demonstrado na subseção seguinte.

4.1.2 Método GRASP reativo

O método GRASP apresentado no começo desta seção, também denominado GRASP básico, pode se tornar reativo. No GRASP básico, o parâmetro de aleatoriedade α é constante, e com isso o tamanho de LRC terá sempre a mesma proporção com relação ao tamanho de LC, que por sua vez varia em cada iteração da fase de construção. No entanto, conforme dito anteriormente é difícil encontrar um tamanho de LCR que dê em média os melhores resultados. Com $\alpha = 0$ o GRASP se torna uma heurística determinística e em todas as iterações a solução construída será a mesma. Se α for muito grande, as soluções serão mais diversificadas e soluções de baixa qualidade serão encontradas.

O princípio do GRASP reativo, proposto por PRAIS e RIBEIRO (2000), é que o próprio algoritmo encontre o melhor valor de α em um pequeno conjunto de

valores permitidos. Em outras palavras, o GRASP reativo faz o auto-ajuste do parâmetro de aleatoriedade α, utilizado na fase de construção, de acordo com a qualidade das soluções encontradas nas iterações anteriores.

O método GRASP reativo funciona da seguinte maneira: Seja v um número fixo de valores de α permitidos e seja $A = \{\alpha_1, \alpha_2, ..., \alpha_v\}$ a lista que contém esses valores. Ambos são determinados durante testes preliminares. Dessa forma, considera-se que α_k é o k-ésimo elemento de A, $k = 1, 2,...,v$. O método GRASP reativo é muito similar ao GRASP básico, exceto que o valor de α é escolhido aleatoriamente em A, antes de se construir uma solução.

A escolha do valor α_k a ser utilizado é baseado apenas na probabilidade (ou chance) que cada valor de $\alpha_k \in A$ possui. Inicialmente, todos os valores $\alpha_k \in A$ têm a mesma probabilidade de serem escolhidos, ou seja, $P_k = 1/v$. A cada iteração escolhe-se um valor $\alpha_k \in A$, com uma probabilidade P_k de ser escolhido, para o parâmetro α. As probabilidades P_k são recalculadas a cada γ iterações, de forma tal que os valores de α_k que produzem melhores soluções terão maior probabilidade P_k, consequentemente, nas próximas iterações terão maiores chances de serem escolhidos.

O método GRASP reativo foi implementado da seguinte maneira: a quantidade de soluções construídas utilizando cada valor α_k é armazenado num arranjo de tamanho v, denominado *count*. Da mesma forma, a soma dos valores da função objetivo dessas soluções é armazenada num arranjo, também de tamanho v, denotado por *score*. Considera-se Z_{min} o menor valor da função objetivo encontrada até o momento. Os valores de P_k são atualizados a cada γ iterações, utilizando *count*, *score*, Z_{min} e um parâmetro de amplificação θ, que justamente amplifica a diferença entre os valores Q_k, que são utilizados diretamente para definir os novos valores de P_k.

A Figura 4.3 apresenta a estrutura genérica do método GRASP reativo. Nos passos 2 a 6 são inicializadas as variáveis e estruturas usadas no ajuste do parâmetro α. No passo 8 seleciona-se aleatoriamente, com probabilidade P_k, um $\alpha_k \in A$ que será utilizado na iteração atual. No passo 15 é incrementado o número de soluções construídas com o α_k escolhido e também é acumulado o valor da função objetivo encontrada nessa iteração. Os passos 17 a 20, correspondentes à atualização das

probabilidades P_k, são executados a cada γ iterações e funcionam da seguinte maneira:

- Para cada valor $\alpha_k \in A$, calcula-se $avg[k] = score[k]/count[k]$ e $Q_k = (Z_{min}/avg[k])^{\theta}$;
- Calcula-se a soma σ de todos Q_k;
- Substitua cada P_k por Q_k/σ;

```
GRASP_reativo
1   Z_min ← +∞;
2   iter ← 1;
3   Defina A = {α1, α2,..., αv}
4   para k ← 1 até v faça
5     count[k] ← 0; score[k] ← 0; p_k ← 1/v;
6   fim-para;
7   enquanto não CritérioParada faça
8     α ← Selecione α_k ∈ A com probabilidade de escolha P_k;
9     s1 ← Construção_Solução(α);
10    s2 ← Busca_Local(s1);
11    se Z(s2) < Z_min então
12      s ← s2;
13      Z_min ← Z(s2);
14    fim-se
15    count[k] ← count[k] + 1; score[k] ← score[k] + Z(s2);
16    se iter mod γ = 0 então
17      avg[k] ← score[k]/count[k] para todo k ∈{1,2,...,v};
18      Q_k ← (Z_min/avg[k])^Θ        para todo k ∈{1,2,...,v};
19      σ ← Σ Q_k
20      P_k ← Q_k /σ                  para todo k ∈{1,2,...,v};
21    fim-se
22    iter ← iter + 1;
23  fim-enquanto;
24  retorne s;
fim;
```

Figura 4.3. Algoritmo do mecanismo reativo do GRASP

Note que quanto maior for a média ($avg[k]$) das soluções obtidas por um valor $\alpha_k \in A$, menor será o valor de Q_k, e consequentemente da probabilidade recalculada P_k. Após alguns blocos de iterações, o GRASP reativo tende a usar o valor de α que encontrou, em média, as melhores soluções. Observando a figura 4.3, pode-se observar que o mecanismo reativo do GRASP adiciona apenas uma pequena sobrecarga ao algoritmo básico do GRASP. Particularmente, a complexidade das fases de construção e da busca local são preservados, o que ajuda a comprovar a vantagem de utilização do mecanismo.

4.2 Metaheurística ILS

A metaheurística ILS (*Iterated Local Search*) foi proposta recentemente na literatura, por LOURENÇO *et al.* (2002), e está sendo aplicada satisfatoriamente em diversos problemas de otimização combinatória. O método ILS foi escolhido para ser aplicado ao PPTMPSR em análise dentro da arquitetura computacional proposta. Considerada, segundo BLUM e ROLLI (2003), como uma das metaheurísticas mais generalizadas, em comparação a todas as demais, baseia-se na ideia de aplicacação de uma busca local dentro de uma solução inicial gerada anteriomente, até que um ótimo local seja encontrado e, a partir desse momento, iterativamente, novas soluções de partida são obtidas por meio de perturbações nesta solução ótima local e submetidas a novo processo de busca local.

O procedimento de perturbação da solução ótima local deve permitir que a busca local explore diferentes espaços de vizinhança. Além disso, deve evitar também um reinício aleatório. Em outras palavras, o procedimento de perturbação não pode ser fraco o suficiente para não explorar outros espaços de busca e nem forte o suficiente para que uma solução totalmente diferente da solução ótima local seja encontrada.

Portanto, o ILS é um método de busca local que permite a exploração de pequenos espaços de vizinhança, sem a necessidade de se explorar todo o espectro de soluções existentes para o problema. A Figura 4.4 ilustra o procedimento descrito até o momento. A partir de uma solução ótima local *s**, é realizada uma perturbação e obtida uma solução *s*', na qual é realizada uma busca local na tentativa de se encontrar soluções *s**'melhores que a atual.

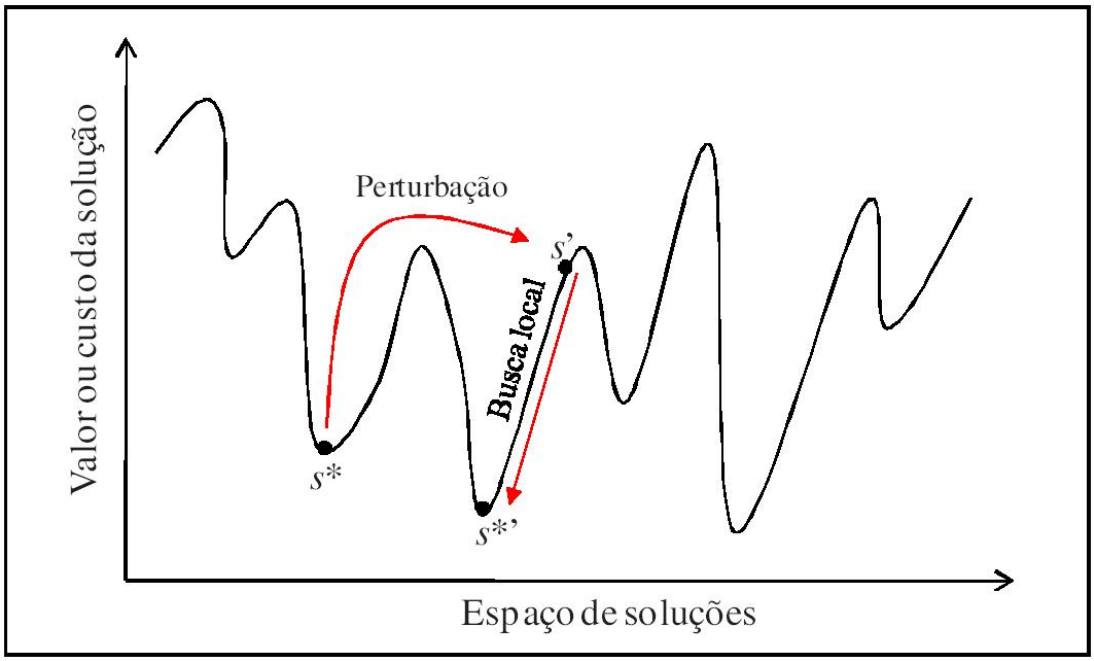


Figura 4.4. Procedimento realizado pela metaheurística ILS

Na Figura 4.5, apresenta-se um pseudocódigo genérico do método ILS, que geralmente possui quatro etapas:

- Obtenção de uma solução ótima local inicial s (passo 1);
- Perturbação da solução obtendo uma solução s_1 (passo 4);
- Melhoria da solução s_1 (passo 5);
- Critério de aceitação da solução atual (passo 9);

As três últimas etapas são executadas iterativamente enquanto o critério de parada não seja atendido. O algoritmo retorna a melhor solução obtida s^* durante toda a sua execução.

```
ILS(d)
1     s ← Obtenção_Solução_Ótima_Local_Inicial;
2     s* ← s;
3     enquanto não CritérioParada faça
4        s1 ← Perturbação(s, d);
5        s2 ← Busca_Local(s1);
6        se Z(s2) < Z(s*) então
7           s* ← s2;
8        fim-se
9        s ← Critério_Aceitação(s, s2);
10    fim-enquanto;
11    retorne s*;
fim;
```

Figura 4.5. Pseudocódigo da metaheurística ILS

Conforme se pode notar no pseudocódigo apresentado anteriormente, o método ILS depende da solução inicial para encontrar soluções de boa qualidade. Observe que o único parâmetro do método é utilizado no procedimento de perturbação, pois define justamente a intensidade da perturbação. Neste trabalho, o parâmetro *d* representa a quantidade de elementos perturbados.

Nas subseções seguintes serão descritas as etapas do método ILS, de acordo com o que foi implementado neste trabalho.

4.2.1 Obtenção da solução ótima local inicial

Nesta etapa uma solução (sequência de tarefas) para o problema PPTMPSR é gerada. Para isso, utiliza-se o método heurístico construtivo guloso DJASA$^+$*, proposto por RUIZ e ANDRÈS (2007) e detalhado no capítulo 3.

O método DJASA$^+$ inicia com uma sequência vazia e, iterativamente, adiciona-se uma tarefa à sequência parcial. Para escolher a tarefa a ser adicionada, são feitas simulações de inclusão, de cada tarefa não sequenciada, em todas as máquinas da sequência parcial. A tarefa da lista, que proporcionar o menor incremento no valor da função objetivo é adicionada à sequência parcial na respectiva máquina que garante este incremento mínimo. O método DJASA$^+$ finaliza quando a sequência contiver todas as *n* tarefas.

Após o sequenciamento, é aplicado o método de re-atribuição ótima de recursos para, finalmente, a solução construída pelo método DJASA$^+$* ser melhorada utilizando um procedimento de busca local. A busca local neste caso conduzirá a solução encontrada pelo método heurístico construtivo até uma solução, garantidamente, ótimo local.

Na Figura 4.6 são exibidas as etapas do procedimento de obtenção da solução ótimo local inicial a ser utilizada no procedimento de perturbação do método ILS.

Obtenção_Solução_Ótima_Local_Inicial	
1	$s \leftarrow$ DJASA$^+$;
2	$s_1 \leftarrow$ Atribuição_Ótima_Recursos(s); // DJASA$^+$*
3	$s^* \leftarrow$ Busca_Local(s_1);
4	retorne s^*;
fim;	

Figura 4.6. Procedimento para obter uma solução ótima local inicial para a metaheurística ILS

A utilização do método DJASA^{+}* é justificada por este método construir boas soluções iniciais, o que geralmente é essencial para a busca local encontrar uma solução ótima local de boa qualidade e, conforme já dito, para que as demais etapas também façam o mesmo.

4.2.2 Procedimento de perturbação

O procedimento de perturbação modifica a solução corrente, que é uma solução ótima local, para encontrar outra solução e permitir, assim, que a busca local explore diferentes locais do espaço de soluções.

Neste trabalho, a modificação é feita por movimentos aleatórios de remoção de elementos da solução corrente. Esses movimentos são responsáveis por alterar a solução corrente, guiando-a para uma solução intermediária. Dessa forma, o mecanismo de perturbação deve ser forte o suficiente para permitir escapar do ótimo local corrente e permitir também a exploração de diferentes regiões do espaço de soluções. Ao mesmo tempo, a perturbação precisa ser fraca o suficiente para guardar características do ótimo local corrente, evitando o reinício aleatório.

Para o PPTMPSR, o procedimento de perturbação é composto de 2 etapas: destruição e reconstrução. Na primeira etapa são escolhidos aleatoriamente e sem repetição, d elementos que serão removidos da solução corrente s. O movimento de remoção dos d elementos cria uma subsequência, de tamanho d, denotada por s_R, que contém os elementos removidos. Neste instante, a solução corrente terá tamanho k-d, onde $k = n+m$-1 (tamanho da sequência), e conterá a sequência original sem os elementos removidos.

Após a remoção dos elementos, realizada na etapa de destruição, é aplicada a segunda etapa, de reconstrução da solução. Nesta etapa, iterativamente, todas os elementos de s_R, na mesma ordem que foram removidos de s, são reinseridos na solução original s. A cada iteração, o elemento é removido de s_R e inserido em cada possível posição de s, gerando, assim, um conjunto temporário de sequências, onde cada uma delas terá o elemento em uma posição. Por fim, a sequência pertencente a esse conjunto que tiver o menor valor da função objetivo é considerada para a próxima iteração. O procedimento encerra-se quando todos os elementos de s_R forem reinseridos em s.

Na Figura 4.7 apresenta-se um pseudocódigo genérico do procedimento de perturbação. No passo 1, a sequência s_R é inicializada vazia. A etapa iterativa de

destruição é apresentada nos passos 3 a 5. Os passos 8 e 9 mostram a etapa iterativa de reconstrução, onde em cada iteração um elemento de s_R é removido e inserido em s, na posição em que apresentará o menor valor da função objetivo.

```
Perturbação(s, d)
1     s_R ← ∅
2     enquanto ( |s_R| < d ) faça     //Etapa de Destruição
3        t_i ← Escolhido aleatoriamente um elemento de s;
4        s_R ← s_R + {t_i};            //Insere o elemento selecionado em s_R
5        s ← s - {t_i};                //Remove o elemento selecionado de s
6     fim-enquanto;
7     enquanto ( |s_R| > 0) faça      //Etapa de Reconstrução
8        s ← s ∪ s_R[1]; //Insere o primeiro elemento de s_R na melhor posição de s
9        s_R ← s_R \ s_R[1];   //Remove o primeiro elemento de s_R
10    fim-enquanto;
11    retorne s;
fim;
```

Figura 4.7. Pseudocódigo genérico do procedimento de perturbação utilizado na metaheurística ILS

A solução retornada pelo procedimento de perturbação pode não ser um ótimo local, por isso é aplicado um procedimento de busca local em seguida.

4.2.3 Critério de aceitação das soluções

O critério de aceitação consiste num procedimento que define se a solução ótima local retornada pela busca local será aceita ou descartada. Uma solução sempre é aceita se for melhor que a melhor solução encontrada até o momento.

No método ILS, as soluções que são piores que a melhor solução atual podem também ser aceitas com uma pequena probabilidade. Dessa forma, evita-se a utilização constante da melhor solução encontrada até o momento e, consequentemente, uma rápida estagnação das soluções avaliadas. Por isso se diz que o procedimento possui um aspecto importante de polarizar a intensificação e diversificação na busca de soluções.

Neste trabalho utiliza-se um critério, que define se uma solução será aceita ou descartada, muito similar ao usado pela metaheurística *Simulated Annealing*. No entanto, o valor da temperatura utilizado no método *Simulated Annealing* é variável e neste trabalho o valor da temperatura é constante e foi definido de forma similar ao trabalho de RUIZ e STÜTZLE (2007). O valor da temperatura é calculado pela expressão a seguir:

$$Temperatura = 0{,}05 \times \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \times m}$$

Observe que o valor da temperatura depende do somatório dos tempos de processamento (p_{ij}) de todas as tarefas em todas as máquinas, além do número de tarefas n e do número de máquinas m.

Após definir o valor da temperatura, pode-se aplicar o procedimento do critério de aceitação, conforme apresentado no pseudocódigo da Figura 4.8. Note que solução s_2 obtida pela busca local é comparada com a solução s que atualmente está sendo submetida ao procedimento de perturbação. Caso s_2 seja melhor que s, a solução s_2 é aceita ($s \leftarrow s_2$), conforme apresentado no passo 2. Caso contrário, ou seja, se s_2 for pior ou igual a s, então é definida uma condição (passo 4), que se for atendida pela solução s_2, ela será aceita. A condição também foi apresentada por RUIZ e STÜTZLE (2007) e foi definida por:

$$r < e^{\frac{-(Z(s_2)-Z(s))}{temperatura}}$$

em que e representa o número neperiano e r representa um número aleatório, escolhido a cada iteração, numa distribuição uniforme no intervalo [0, 1]. Caso o critério seja atendido, a solução s_2 será aceita, mesmo sendo pior que a solução s, e será perturbada na próxima iteração, pois $s \leftarrow s_2$ (passo 5).

```
Critério_Aceitação(s, s2)
1     se Z(s2) < Z(s) então
2        s ← s2;
3     senão
4        se CondiçãoAceitação então
5           s ← s2;
6        fim-se;
7     fim-se;
fim;
```

Figura 4.8. Pseudocódigo do procedimento utilizado como critério de aceitação de soluções na metaheurística ILS

Este critério de aceitação já foi utilizado em diversos trabalhos na literatura científica. Por exemplo, para problemas de programação de tarefas em máquinas paralelas, já foi utilizado por OSMAN e POTTS (1989); RUIZ e STÜTZLE (2007).

4.3 Busca Local

A busca local é um procedimento iterativo que consiste em melhorar uma solução *s* procurando novas soluções, denominadas vizinhas. Estas soluções vizinhas são obtidas realizando alterações, ou movimentos, na estrutura da solução atual *s*.

Neste trabalho, para explorar o espaço de soluções são utilizados dois tipos de movimentos: troca da ordem de dois elementos da sequência e realocação de um elemento em outra posição da sequência. Estes movimentos definem, respectivamente, as estruturas de vizinhança N^T e N^R.

Dado uma instância do PPTMPSR com *n* tarefas e *m* máquinas e, relembrando que a sequência possui tamanho $k = n + m - 1$, na primeira estrutura de vizinhança, N^T, há $k(k-1)/2$ movimentos possíveis ou, equivalentemente, esse mesmo número de vizinhos. Por exemplo, a solução *s*' = {4, 3, -1, 2, 1} é uma solução vizinha da solução *s* = {4, 2, -1, 3, 1}, pois é obtida desta pela troca do segundo com o quarto elemento. Já na segunda estrutura de vizinhança, N^R, há $(k-1)^2$ movimentos possíveis. Por exemplo, a solução *s*' = {2, -1, 3, 4, 1} é obtida de *s* = {4, 2, -1, 3, 1} pela realocação da tarefa da primeira posição na quarta posição da sequência. Nessa estrutura são permitidos movimentos para posições sucessoras ou antecessoras à posição em que o elemento se encontra na sequência. Porém, observe que uma solução vizinha *s*' é gerada, no problema abordado neste trabalho, realocando um elemento que está na posição *x* da sequência em outra posição *y*, de forma que $1 \leq x$, $y \leq k$ e $x \neq y$.

Além das estruturas N^T e N^R de vizinhança, foram utilizados dois critérios de retorno do procedimento. Ambos os critérios foram utilizados nas duas estruturas de vizinhança, tendo como resultado quatro métodos de busca local.

O primeiro critério, denotado por "melhor de todos", consiste em analisar, a partir de uma solução *s* todas as soluções pertencentes a sua estrutura de vizinhança *N*. Seja *s*'a melhor solução dentre as soluções vizinhas de *s*. Se *s*' for melhor que *s*, então a busca local é aplicada em *s*'. Caso contrário, o procedimento é encerrado e a solução *s* é retornada. Relembre que, para definir se uma solução é melhor do que outra solução, é utilizada a função objetivo *Z*.

Na Figura 4.9 é apresentado o algoritmo que usa este critério para uma estrutura de vizinhança *N* qualquer. Neste algoritmo, *nv* representa o número de possíveis vizinhos de *s* em *N*.

```
Busca_Local(s)
1       Z_min ← +∞;
2       Para i := 1 até nv faça
3           s_1 ← Uma solução vizinha de s em N não analisada;
4          se Z(s_1) < Z_min então
5              s' ← s_1;
6              Z_min ← Z(s_1);
7          fim-se;
8       fim-para;
9       se Z(s') < Z(s) então
10         s ← Busca_Local(s');
11      fim-se;
12      retorne s;
fim;
```

Figura 4.9. Busca local genérica realizada em *s* na vizinhança *N*, baseado no critério "melhor de todos"

O segundo critério, denotado por "primeiro melhor", consiste em analisar cada solução vizinha de s em N. No entanto, a primeira solução $s' \in N$ que for melhor que s, será considerada e nela aplicada a busca local recursivamente, sem verificar as demais soluções vizinhas de s em N.

A Figura 4.10, da mesma forma que o critério "melhor de todos", apresenta o algoritmo básico do procedimento de busca local que utiliza este critério. Note que, em ambos os critérios, se não for encontrada nenhuma solução melhor que s, a própria solução s é retornada.

```
Busca_Local(s)
1       Para i := 1 até nv faça
2          s' ← Uma solução vizinha de s em N não analisada;
3          se Z(s') < Z(s) então
4              s ← Busca_Local(s');
5              retorne s;              // Retorna e sai do laço iterativo
6          fim-se;
7       fim-para;
8       retorne s;
fim;
```

Figura 4.10. Busca local genérica realizada em *s* na vizinhança *N*, baseado no critério "primeiro melhor"

Nos testes realizados verificou-se que o procedimento de busca local que apresenta os melhores resultados baseia-se no critério "melhor de todos", quando aplicado sobre a estrutura de vizinhança de realocação N^R. Os resultados deste teste são detalhados no capítulo 5 deste trabalho.

Portanto, foi utilizado em todos os métodos descritos neste trabalho, o procedimento de busca local baseado na estrutura de vizinhança N^R e com critério de retorno definido por "melhor de todos".

Conforme destacado por RUIZ *et al*. (2006), a estrutura de vizinhança N^R tem sido considerada superior a outras estruturas de vizinhança em diversos problemas de programação de tarefas em máquinas paralelas, o que ajuda a confirmar os resultados obtidos pelos testes.

4.4 *Path relinking*

A técnica reconexão de caminhos, do inglês *path relinking*, é uma estratégia que integra estratégias de intensificação e diversificação. Originalmente, o *path relinking* foi proposto por GLOVER (1996) no contexto das metaheurísticas *tabu search* (GLOVER e LAGUNA, 1997) e *scatter search* (GLOVER, 1998; GLOVER *et al*., 2000; YAMASHITA *et al*., 2004).

A utilização do *path relinking* proporciona a geração de novas soluções através da exploração de trajetórias que conectam soluções de alta qualidade (intensificação), e tenta encontrar soluções que estão em diferentes regiões da vizinhança (diversificação), que por sua vez não foram analisadas no procedimento de busca local.

Iniciando com uma solução ótima local, denotada solução origem s_s, um caminho é gerado, num espaço de vizinhança diferente do usado no procedimento de busca local, conectando s_s a uma outra solução, denotada solução guia s_g. Em outras palavras, o objetivo desta estratégia é construir iterativamente um caminho de soluções entre a solução origem e a solução guia. Este caminho será gerado por meio de movimentos, ou modificações, realizados na solução origem que incluam atributos da solução guia, a cada iteração.

RESENDE e RIBEIRO (2005) descreveram alternativas que são consideradas em recentes implementações do *path relinking*. Considerando s_1 e s_2 duas soluções do problema que será aplicado o *path relinking*, listam-se abaixo algumas dessas alternativas.

- *forward relinking*: A solução origem é a pior das soluções s_1 e s_2.
- *backward relinking*: A solução origem é a melhor das soluções s_1 e s_2.
- *back* e *forward relinking*: Dois caminhos são explorados. Sendo que em um caminho a solução origem é s_1 e no outro é s_2.

Neste trabalho, utiliza-se a última alternativa citada, ou seja, o *path relinking* que explora dois caminhos, um conectando a solução origem à solução guia, e o outro na direção inversa, já que esses caminhos podem ser diferentes. Além disso, neste trabalho também foram testadas duas estratégias que integram as metaheurísticas GRASP e ILS com o *path relinking*. A primeira estratégia conecta uma solução ótima local obtida pela metaheurística com uma solução de boa-qualidade escolhida aleatoriamente, denotada solução elite. A segunda estratégia cria caminhos que conectam cada par do conjunto de soluções elites, sendo assim um procedimento de pós-otimização. A estratégia de pós-otimização nos testes realizados não proporcionou vantagens a nenhuma das metaheurísticas e por isso foi descartada.

O *path relinking*, neste trabalho, foi integrado aos métodos GRASP reativo e ILS, de forma que em ambos os métodos o *path relinking* mantém um conjunto de soluções elite E, similar a AIEX *et al*. (2005). Da mesma forma, em ambos os métodos, a solução origem s_s é obtida pelo procedimento de busca local que foi aplicado à solução construída (GRASP reativo) ou perturbada (ILS). O *path relinking* inicia com a escolha aleatória da solução guia $s_g \in E$, desde que $s_s \neq s_g$. Utilizando movimentos de trocas de tarefas em s_s, de tal forma que estes movimentos definem a estrutura de vizinhança N^T, o *path relinking* tenta encontrar soluções melhores que s_s e s_g, que ainda não foram avaliadas pela busca local, já que esta utiliza outra estrutura de vizinhança.

Em cada etapa do procedimento, analisam-se todas as possíveis trocas na ordem das tarefas, de modo que a troca incorpore atributos da solução guia s_g na solução origem s_s. É escolhida a troca que melhora mais (ou deteriora menos) a solução origem. A solução s_s obtida após a movimentação é armazenada se for melhor que a melhor solução encontrada durante a execução do *path relinking*. O procedimento continua até que s_s e s_g se tornem iguais.

Na Figura 4.11 é exibido um pequeno exemplo em que a solução guia é obtida, através da solução origem, utilizando apenas duas iterações.

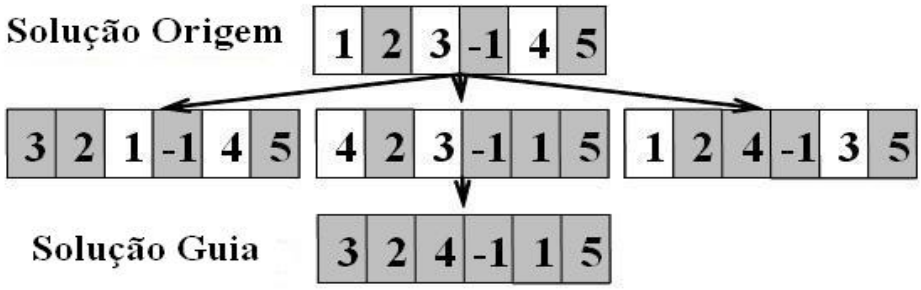


Figura 4.11. Exemplo de utilização da técnica *path relinking*

Observe no exemplo da Figura 4.11, que a partir da solução origem é possível realizar três movimentações de troca, de forma que a solução origem se torne mais próxima da solução guia. Dentre as três possíveis trocas, é escolhida aquela que melhora mais (ou deteriora menos) a solução origem. A partir desta solução realiza-se mais uma troca, para que as soluções origem e guia se tornem idênticas.

Para uma solução origem que possui k elementos em posições diferentes da solução guia, são necessárias k-1 movimentações de troca até que as soluções sejam idênticas. Na exploração desse caminho, k-2 soluções vizinhas, diferentes da solução origem e da solução guia, são avaliadas.

Nos algoritmos propostos neste trabalho, o *path relinking* está acoplado aos métodos GRASP reativo e ILS. Nestes algoritmos o *path relinking* é aplicado como refinamento de ótimos locais e as soluções de alta qualidade são armazenadas no conjunto E. Nestes casos, os algoritmos têm como entrada o parâmetro E_{tam}, que representa o tamanho máximo do conjunto E.

O conjunto E não permite a inclusão de uma solução repetida. Na execução dos métodos GRASP reativo com *path relinking* e ILS com *path relinking*, enquanto $|E|$ for menor que E_{tam}, a solução obtida pela busca local é inserida em E, desde que seja diferente de das soluções contidas em E. Caso $|E| = E_{tam}$, então é feita a comparação entre a solução obtida após a aplicação do *path relinking* com a pior solução do conjunto elite E. Se a solução obtida pela busca local for melhor que a pior solução do conjunto elite E e não estiver contida em E, então a pior solução de E é removida do conjunto elite e a solução obtida pela busca local é inserida no seu lugar. Note que o conjunto elite E sempre conterá as melhores soluções encontradas.

Nas Figuras 4.12 e 4.13 são exibidos os algoritmos obtidos na integração da técnica *path relinking* com os métodos GRASP reativo e ILS, respectivamente.

```
GRASP_reativo_path_relinking(E_tam)
1       Z_min ← +∞;
2       E = ∅;
3       iter ← 1;
4       Defina A = {α_1, α_2,..., α_v}
5       para k ← 1 até v faça
6         count[k] ← 0; score[k] ← 0; p_k ← 1/v;
7       fim-para;
8       enquanto não CritérioParada faça
9          α ← Selecione α_k ∈ A com probabilidade de escolha p_k;
10         s ← Construção_Solução (α);
11         s_1 ← Busca_Local(s);
12         se |E| = E_tam então
13            s_2 ← Uma solução elite de E escolhida aleatoriamente;
14            s_3 ← Path_Relinking(s_1, s_2);
15            s_4 ← Path_Relinking(s_2, s_1);
16            s ← Melhor_Solucao(s_3, s_4);
17            se s ∉ E e s ≠ s_1 então
18               s ← Busca_Local(s);
19            fim-se
20            ATUALIZA_E(s);    // Atualiza também Z_min (se for o caso)
21         senão
22            E ← E ∪ {s_1};
23         fim-se
24         count[k] ← count[k] + 1; score[k] ← score[k] + Z(s_2);
25         se iter mod γ = 0 então
26            avg[k] ← score[k]/count[k] para todo k ∈{1,2,...,v};
27            Q_k ← (Z_min/avg[k])^Θ          para todo k ∈{1,2,...,v};
28            σ ← Σ Q_k
29            P_k ← Q_k /σ                    para todo k ∈{1,2,...,v};
30         fim-se
31         iter ← iter + 1;
32      fim-enquanto
33      retorne Melhor_Solucao(E);
fim;
```

Figura 4.12. Pseudocódigo do algoritmo GRASP reativo com *path relinking*

```
ILS_path_relinking(E_tam, d)
1   E = ∅;
2   s ← Obtenção_Solução_Ótima_Local_Inicial;
3   enquanto não CritérioParada faça
4       s2 ← Perturbação(s, d);
5       s2 ← Busca_Local(s2);
6       se |E| = E_tam então
7           s3 ← Uma solução elite de E escolhida aleatoriamente;
8           s4 ← Path_Relinking(s2, s3);
9           s5 ← Path_Relinking(s3, s2);
10          s6 ← Melhor_Solucao(s4, s5);
11          se s6 ∉ E e s6 ≠ s2 então
12              s2 ← Busca_Local(s6);
13          senão
14              s2 ← s6;
15          fim-se
16          ATUALIZA_E (s2);
17      senão
18          E ← E ∪ {s2};
19      fim-se
20      s ← Criterio_Aceitação(s, s2);
21  fim-enquanto;
22  retorne Melhor_Solucao(E);
fim;
```

Figura 4.13. Pseudocódigo do algoritmo ILS com *path relinking*

Dos pseudocódigos apresentados nas Figuras 4.12 e 4.13, nota-se que o conjunto de soluções elite é inicializado vazio em ambos os procedimentos. Da mesma forma, ambos os procedimentos verificam se o conjunto elite já foi totalmente preenchido antes de escolher aleatoriamente a solução guia no conjunto elite. Após a aplicação do *path relinking* nos dois sentidos, a melhor solução encontrada é armazenada em E se for melhor que a pior solução presente em E naquele instante. Ao final dos procedimentos, a melhor de todas as soluções contidas no conjunto elite é retornada como solução do método.

Neste instante, convém ressaltar que foram apresentados até o momento, cinco métodos que serão aplicados ao PTMPSR: GRASP, GRASP reativo, GRASP reativo com *path relinking*, ILS e ILS com *path relinking*. Desses, apenas o terceiro e quinto método utilizam a estratégia de intensificação *path relinking*. Enquanto os demais são baseados nos métodos básicos das metaheurísticas GRASP e ILS, descrito nas seções anteriores.

4.5 Critério de parada dos métodos iterativos

Os métodos heurísticos melhorativos que foram apresentados nas seções anteriores possuem a característica comum de serem procedimentos iterativos, ou seja, são executados várias vezes até que um critério, ou condição, seja atendido.

Conforme dito anteriormente, na maioria das aplicações utiliza-se o critério de parada mais simples, que é o baseado no número de iterações. Neste critério, é definido de antemão o número n_i de vezes (iterações) que o procedimento será executado. No entanto, outros critérios podem ser usados, como por exemplo, definir um número limite de iterações l_i em que o procedimento poderá ficar sem encontrar melhores soluções, além do critério baseado num tempo limite de execução.

Estes critérios possuem algumas desvantagens quando analisados mais profundamente. Por exemplo, para problemas que possuem instâncias com grande variação na definição do seu tamanho, pode ocorrer que para uma instância pequena do problema, o número de iterações definido seja elevado demais, consumindo tempo desnecessariamente, enquanto que para problemas de grandes instâncias o mesmo número de iterações pode ser insuficiente. O mesmo problema ocorre para casos em que é utilizado o critério baseado num tempo limite de execução. Os métodos, que têm o critério de parada baseado no limite de iterações, possuem a desvantagem relacionada com a dificuldade de definir um número limite de iterações, uma vez que sempre ficará a dúvida se na próxima iteração, após alcançar o limite, não se encontraria uma solução melhor.

Por isso, nos métodos iterativos apresentados neste trabalho, é utilizado um critério de parada que tem como objetivo fugir das consequências negativas citadas acima. Nesse caso, o critério de parada baseou-se num tempo limite de execução, e esse tempo limite é definido de acordo com o tamanho da instância do problema.

No caso do PPTMPSR, definiu-se que as medidas que determinam o tamanho de uma instância do problema são o número de tarefas n e o número de máquinas m. A partir disso, testes, cujos resultados serão apresentados no próximo capítulo, foram realizados para se definir um tempo limite justo. O tempo computacional analisado e identificado, pelos resultados dos testes, como sendo ideal para ser utilizado como critério de parada, foi de $(n \times m)/2$ segundos. Dessa forma, por exemplo, em cada método apresentado anteriormente, um problema com n = 100 tarefas e m = 20 máquinas é executado durante 1000 segundos.

No próximo capítulo são descritos os resultados obtidos na aplicação dos métodos apresentados ao PPTMPSR. Estes resultados são comparados com outros da literatura e entre si, mostrando a eficiência na utilização das metaheurísticas propostas.

5 EXPERIMENTAÇÃO COMPUTACIONAL

Nesse capítulo são apresentados os resultados da aplicação das metaheurísticas propostas e da técnica de intensificação *path relinking* ao conjunto de problemas disponibilizados na internet para testes. Além disso, são detalhados também os parâmetros e justificativas para utilização dos métodos.

A seção 5.1 apresenta os detalhes do ambiente em que os testes foram realizados e a descrição básica do conjunto de problemas que foram utilizados para testes, enquanto que a seção 5.2 apresenta os parâmetros envolvidos na aplicação dos métodos, além da própria justificativa para sua utilização. Por fim, a seção 5.3 apresenta os resultados e as comparações de desempenho entre os métodos heurísticos construtivos e os resultados das metaheurísticas, e entre as próprias metaheurísticas.

5.1 Ambiente de testes e instâncias utilizadas

Para validar o modelo e a metodologia descrita no capítulo anterior foram aplicados 5 métodos a um conjunto de instâncias-teste do PPTMPSR, gerados e disponibilizados na internet, em *http://www.upv.es/gio/rruiz*, por RUIZ e ANDRÉS (2007). Os métodos são baseados nas metaheurísticas GRASP e ILS, e na técnica de intensificação e diversificação *path relinking*, sendo denotados por: GRASP Básico (GB), GRASP Reativo (GR), GRASP Reativo com *path relinking* (GRPR), ILS Básico (ILS) e ILS com *path relinking* (ILSPR).

Os algoritmos dos métodos desenvolvidos foram implementados na linguagem de programação Java, versão 1.6. Os testes foram executados usando o compilador JDK 6.0. Já o computador utilizado para execução dos testes possui processador Intel® Core™ Quad com 2.4GHz e 3GB de memória RAM, sem a utilização de multiprocessamento.

Os resultados disponibilizados por RUIZ e ANDRÈS (2007) para as heurísticas construtivas foram obtidos quando executadas em um Pentium IV com 3.2GHz e 1 GB de memória RAM. Na seção 5.4, ao realizarmos as comparações entre os

resultados, é necessário observar a diferença das configurações das máquinas em que os testes foram realizados.

Conforme citado anteriormente, RUIZ e ANDRÈS (2007) geraram um conjunto de 720 instâncias-teste para o PPTMPSR e disponibilizaram esse conjunto, com os seus respectivos resultados, na internet. Neste trabalho, utilizamos este mesmo conjunto de instâncias-teste para avaliarmos o desempenho dos métodos propostos.

Nos exemplos do PPTMPSR apresentados anteriormente, nota-se que, além do número de tarefas n e do número de máquina m, vários outros conjuntos de dados devem ser gerados. Por exemplo, os tempos de processamento (p_{ij}), os valores mínimos e máximos de recursos e os *setup times* (R_{ijk}^{-}, R_{ijk}^{+}, S_{ijk}^{-} e S_{ijk}^{+}), respectivamente.

As 720 instâncias-teste foram divididas em 2 grupos: os de pequeno porte (do inglês: *small*) e os de grande porte (do inglês: *large*). Essa divisão foi feita para que RUIZ e ANDRÉS (2007) pudessem testar a execução das instâncias do grupo *small* num software matemático. Nas instâncias do grupo *large* isso não seria possível devido ao grande número de variáveis e restrições exigidas.

No grupo de instâncias *small*, todas as seguintes possíveis combinações de n e m foram utilizadas: $n \in \{6, 8, 10\}$ e $m \in \{3, 4, 5\}$. Para o grupo de instâncias *large* as combinações são: $n \in \{50, 75, 100\}$ e $m \in \{10, 15, 20\}$. Em todos esses casos, os tempos de processamento foram gerados de acordo com a distribuição uniforme no intervalo [1, 99] que, segundo RUIZ e ANDRÉS (2007), é o mais utilizado em problemas de programação de tarefas em máquinas paralelas.

Para os valores mínimos e máximos de recursos são consideradas duas combinações. Na primeira, as quantidades mínima e máxima são uniformemente distribuídas nos intervalos [1, 3] e [3, 5], respectivamente. Na segunda combinação, é utilizada a distribuição uniforme nos intervalos [1, 5] e [5, 10], respectivamente. Isso significa que R_{ijk}^{-} é uniformemente distribuído em [1, 3] e [1, 5], enquanto que R_{ijk}^{+} é uniformemente distribuído em [3, 5] e [5, 10]. Nota-se que apenas duas combinações são consideradas para (R_{ijk}^{-}; R_{ijk}^{+}): ([1, 3]; [3, 5]) ou ([1, 5]; [5, 10]).

Os valores mínimos e máximos dos *setup times* também são considerados em duas combinações. Na primeira, a quantidade mínima e máxima de *setup time* é uniformemente distribuída nos intervalos [1, 50] e [50, 100], respectivamente. Na segunda, a distribuição uniforme ocorre, respectivamente, nos intervalos [50, 100] e

[100, 150]. Da mesma forma que a distribuição dos recursos, estas duas combinações representam a distribuição dos valores mínimos e máximos dos *setup times*.

A tabela 5.1 apresenta quais são as possíveis combinações nos dois grupos (*small* e *large*) de instâncias-teste.

Tabela 5.1. Possíveis combinações nos valores de *n*, *m*, recursos e *setup times* utilizados na geração das instâncias-teste do PPTMPSR por RUIZ e ANDRÈS (2007)

	Small	***Large***
n	{6, 8, 10}	{50, 75, 100}
m	{3, 4, 5}	{10, 15, 20}
$(S_{ijk}^{-}; S_{ijk}^{+})$	([1, 50];[50, 100]) ou ([50, 100];[100, 150])	
$(R_{ijk}^{-}; R_{ijk}^{+})$	([1, 3];[3, 5]) ou ([1, 5];[5, 10])	

Note que há a possibilidade de R_{ijk}^{-} ser igual à R_{ijk}^{+}. Essa idéia tenta descrever a realidade de algum *setup time* que não pode variar, ou seja, nesse caso tem-se que S_{ijk}^{-} é igual a S_{ijk}^{+}.

Como conclusão, na geração das instâncias-teste têm-se três valores para *n*, três para *m*, dois intervalos de recursos e dois intervalos para *setup times*, em cada grupo. No total há 36 possíveis combinações de valores, sendo que para cada uma dessas combinações foram geradas 10 instâncias, ou seja, cada grupo possui 360 instâncias-teste. Consequentemente, conforme dito anteriormente, o total de instâncias-teste utilizadas é 720.

5.2 Parâmetros dos métodos propostos e sua utilização nos testes

Os métodos heurísticos implementados neste trabalho utilizam parâmetros que definem a sua melhor forma de utilização. Alguns desses parâmetros tiveram que ser calibrados para se definir o valor ideal. Nestas calibrações, foram realizados testes que nos permitiram avaliar e chegar a algumas conclusões, antes da execução das instâncias-teste.

Nesta seção iremos demonstrar a utilização destes testes e como utilizamos os resultados para definir os respectivos parâmetros. Em alguns casos, os parâmetros foram definidos basicamente de acordo com indicações na literatura científica.

5.2.1 Custos de unidade de recurso e unidade de tempo

No capítulo 2 deste trabalho foi apresentada a função objetivo *Z* do PPTMPSR. A função objetivo busca minimizar dois critérios: o tempo total de conclusão das

tarefas e o número de recursos utilizados na preparação das máquinas. Portanto, para modularizar o valor da função objetivo são utilizados os parâmetros λ e δ, que representam os custos associados a cada unidade de recurso e a cada unidade de tempo, respectivamente.

Nossa intenção é comparar os resultados obtidos pelos métodos propostos neste trabalho com os obtidos por RUIZ e ANDRÉS (2007). Portanto, é necessário que esses parâmetros, na execução dos métodos propostos, possuam os mesmos valores utilizados por tais autores.

RUIZ e ANDRÉS (2007) afirmam que geralmente nos casos reais do PPTMPSR os recursos são muito escassos e, consequentemente, o seu custo é muito maior que a unidade de tempo. Seguindo essa lógica eles definiram estes custos da seguinte forma: $\lambda = 50$ e $\delta = 1$. Isso indica que cada unidade de recurso é 50 vezes mais caro que uma unidade de tempo.

5.2.2 Parâmetros do GRASP Básico

O método GRASP Básico possui apenas dois parâmetros a serem definidos: O critério de parada e o valor da taxa de aleatoriedade (α). Na tentativa de definir um valor para α que proporcione bons resultados, estudou-se a estrutura do algoritmo e foi notado que seria interessante que esse valor fosse diferente para os grupos *small* e *large*.

A partir disso, foram escolhidas 10 instâncias-teste de cada um dos grupos. No grupo *small* instâncias com $n = 10$ tarefas e $m = 5$ máquinas foram escolhidas, enquanto que no grupo *large* instâncias com $n = 100$ tarefas e $m = 20$ máquinas foram escolhidas. Observe que essas instâncias são consideradas as maiores em cada um dos grupos, por ter o maior número de tarefas (n) e o maior número de máquinas (m) permitidas.

Cada uma destas instâncias foi executada 4 vezes. Em cada execução um valor diferente de α foi utilizado. Os valores de α testados pertencem ao seguinte conjunto: {0,1; 0,3; 0,5; 0,7}. Neste teste específico, o critério de parada foi baseado no número de iterações. Dessa forma, foi definido o critério de parada em 2000 iterações.

Na tabela 5.2 apresenta-se a média dos valores finais da função objetivo Z, para cada grupo e para cada valor de α.

Tabela 5.2. Médias do valor de *Z* para os dez maiores problemas de cada grupo quando aplicado o método GRASP Básico com quatro diferentes valores da taxa de aleatoriedade *α*

	Small	***Large***
α = 0,1	1.604	**33.750**
α = 0,3	1.592	34.101
α = 0,5	**1.571**	34.796
α = 0,7	1.582	35.693

Com os dados da tabela 5.2, nota-se que para as instâncias do grupo *large*, a média das soluções aumenta à medida que o valor de α também aumenta. Essa situação já era esperada, uma vez que, aumentando o valor de α, a solução será construída de forma mais aleatória, e por se tratar de instâncias de grande porte, as chances da solução construída ser completamente diferente de uma solução gulosa é maior.

Nas instâncias do grupo *small* a tendência é justamente o contrário. No entanto, observa-se que o valor médio das soluções tende a ser menor entre os valores de α entre 0,3 e 0,7, sendo que a menor média foi obtida com α = 0,5. A justificativa para isso é que, com valores menores de α, a solução era construída praticamente de forma gulosa e, nesse conjunto de instâncias, isso afetava a diversificação na construção das soluções pelo método GRASP Básico.

Dessa forma, o parâmetro α da taxa de aleatoriedade, quando utilizado no método GRASP Básico, foi calibrado e definido em α = 0,5 para instâncias de pequeno porte e α = 0,1 para instâncias de grande porte.

Em outra análise realizada com esse mesmo teste, verifica-se a dificuldade de se definir um valor fixo de α que seja unânime em encontrar os melhores valores. No teste citado acima, exibimos a média dos valores de *Z* para cada conjunto de 10 instâncias de cada um dos grupos: *small* e *large*. No entanto, na tabela 5.3, exibe-se a quantidade de instâncias em que cada um dos valores de α encontrou a melhor solução.

Tabela 5.3. Quantidade de instâncias em que cada valor de *α* encontrou a melhor solução

	Small	***Large***
$\alpha = 0{,}1$	1	**8**
$\alpha = 0{,}3$	1	2
$\alpha = 0{,}5$	**6**	0
$\alpha = 0{,}7$	2	0

Pela análise da tabela 5.3, observe que no conjunto das 10 instâncias do grupo *small*, em 4 instâncias a utilização de outro valor de α, diferente do que foi definido, apresentou resultados melhores. Este fato também ocorreu com as 10 instâncias do grupo *large*, porém em menor escala.

Essa análise comprova a dificuldade de se estabelecer um valor fixo de α que sempre encontrasse as melhores soluções. A partir disso, essa análise do teste foi a justificativa para a implementação e utilização do mecanismo reativo no GRASP.

5.2.3 Parâmetros do GRASP Reativo

Na implementação do mecanismo reativo no GRASP, e da mesma forma que o GRASP Básico, é necessário que os parâmetros sejam calibrados e testados para fornecerem o melhor desempenho possível.

Para o parâmetro v, que define o número de elementos no conjunto A, definiu-se que seriam utilizados valores de α uniformemente distribuídos no intervalo entre [0, 1], excluindo-se as extremidades do intervalo. Na seção anterior foram definidos os valores de $\alpha = 0{,}5$ (grupo *small*) e $\alpha = 0{,}1$ (grupo *large*). Dessa forma, decidiu-se incluir os valores testados anteriormente no conjunto A, e de tal forma que outras possibilidades também fossem testadas.

Por fim, alguns testes foram executados com $v = 9$, ou seja, $A = \{0{,}1;, 0{,}2; ...; 0{,}9\}$. Estes testes mostraram que após a execução de 5000 iterações, quando aplicados a 10 instâncias do grupo *small* com $n = 10$ tarefas, os valores de $\alpha_k \in A$ possuíam probabilidades finais diferentes. Isso indica que valores de $\alpha_k \in A$ que não haviam sido testados anteriormente, foram utilizados neste teste. Em alguns casos esses valores de α apresentaram bom desempenho.

Os gráficos apresentados na figura 5.1 exibem a diferença da probabilidade final para uma mesma instância, quando executada duas vezes consecutivamente.

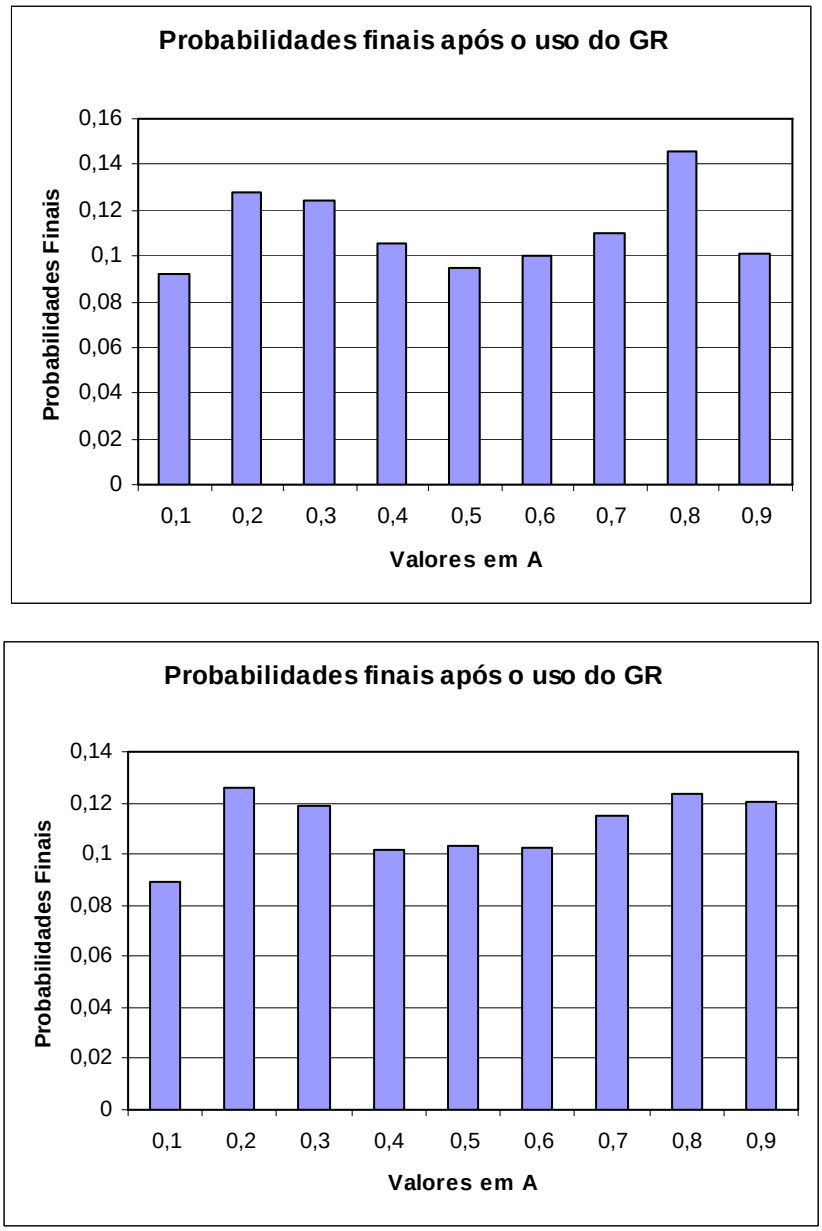


Figura 5.1. Variação das probabilidades finais após execução consecutiva do GRASP Reativo na mesma instância do grupo *small*

Observe que na primeira execução dessa instância o valor $\alpha = 0.8$, que não havia sido testado anteriormente, apresentou probabilidade final superior aos demais valores de α, ou seja, encontrou em média soluções melhores que as demais. Na segunda execução, o valor $\alpha = 0.2$, que também não havia sido testado anteriormente, apresentou probabilidade final maior que os demais. Dessa forma, se manteve a utilização do parâmetro $v = 9$ e do conjunto $A = \{0,1; 0,2; ... ; 0,9\}$.

Além disso, os gráficos da figura 5.1, comprovam que a utilização do GRASP Reativo se mostrou eficiente ao evitar a definição de um valor fixo para α, já que a

cada execução um valor diferente de α obteve a maior probabilidade final, e assim encontrou, em média, as melhores soluções.

O parâmetro γ que define em quantas iterações o valor das probabilidades vai ser atualizado foi determinado a partir de testes com $\gamma = 5$, $\gamma = 10$ e $\gamma = 20$. Os resultados mostraram que a diferença em termos do valor da função objetivo Z foi quase nula, o que não permitiu chegarmos a uma conclusão baseado neste fato. No entanto, analisando a estrutura do algoritmo, nota-se que um valor de $\alpha_k \in A$ pode ter sua probabilidade reduzida logo nas primeiras iterações, caso o valor de γ seja pequeno. E, assim, esse valor de α, que poderia ser mais utilizado e gerar boas soluções, não apresentará desempenho satisfatório. Com isso, o maior valor testado ($\gamma = 20$) foi usado no método GRASP Reativo.

O fator de amplificação θ foi testado com três valores, $\theta = 10$, $\theta = 50$ e $\theta = 100$. Da mesma forma que o parâmetro γ, os resultados não apresentaram diferença significativa no valor da função objetivo Z. Por isso, foi escolhido o valor $\theta = 10$, que é o valor utilizado em todos os trabalhos sobre GRASP Reativo pesquisados na literatura científica, como, por exemplo, em PRAIS e RIBEIRO (2000).

5.2.4 Parâmetros do ILS

Nos métodos ILS implementados neste trabalho, o valor do parâmetro d é praticamente o único a ser definido. Lembre que o parâmetro d representa o número de elementos da sequência que serão removidos da sequência original e reinseridos, um a cada iteração, na melhor posição possível da sequência original, durante o procedimento de *perturbação*. Neste procedimento, a solução ótima local perturbada não pode sofrer uma perturbação muito forte, para gerar um reinício aleatório, e nem muito fraca, que não possibilite escapar da solução ótima local.

Dessa forma, neste trabalho definiu-se que a solução não pode sofrer perturbação em mais de 50% dos seus elementos. As menores instâncias-teste do grupo *small* possuem $n = 6$ e $m = 3$, logo o tamanho $k = n+m-1$ da sequência para essas instâncias será $k = 8$, ou seja, o valor do parâmetro d não deveria ser maior que 4.

Todas as instâncias-teste (grupos *small* e *large*) foram executadas com $d = 4$. No entanto, para as instâncias do grupo *large* que possuíam valores mais elevados de k (tamanho da sequência), foram testados outros valores de d. Neste teste foram utilizadas duas instâncias, sendo uma com $n = 50$ e $m = 10$ (menor instância do grupo

large) e a outra com $n = 100$ e $m = 20$ (maior instância do grupo *large*). A tabela 5.4 exibe a média dos resultados encontrados usando cada valor de d testado.

Tabela 5.4. Médias do valor de *Z* encontradas usando diferentes valores de *d*

	Média no valor de Z
***d* = 4**	18.692
***d* = 10**	**18.563**
***d* = 30**	19.321

Note que o maior valor de d testado é 30, pois praticamente representa 50% do valor de k para as menores instâncias ($n = 50$, $m = 10$ e consequentemente $k = 59$). Além disso, note na tabela 5.4 que os melhores valores foram encontrados para $d = 10$, que dessa forma foi utilizado em todas as instâncias do grupo *large*.

A equação que determina o valor do parâmetro *temperatura*, conforme citado anteriormente, foi definido de acordo com trabalhos disponíveis na literatura, principalmente o de RUIZ e STÜTZLE (2007), que usaram, nesse cálculo, o total dos tempos de processamento, o número de máquinas m, o número de tarefas n e um fator T ajustável a cada problema.

Na definição do valor de T no PPTMPSR, foram usados valores próximos ao que foi usado por RUIZ e STÜTZLE (2007), que foi $T = 0{,}05$. Por isso, foram usados nos testes, além do valor $T = 0{,}05$, os valores $T = 0{,}04$ e $T = 0{,}06$. Por ser utilizado no cálculo do parâmetro *temperatura*, que por sua vez é utilizado na condição de aceitação dos métodos ILS, não foi possível chegar a uma conclusão sobre qual valor de T influenciou a condição de aceitação de forma mais vantajosa. Além disso, em termos de Z não houve diferença nos valores encontrados, quando os testes foram aplicados em 10 instâncias-teste escolhidas aleatoriamente no grupo *large*. Com isso, o valor de $T = 0{,}05$ foi definido para ser utilizado nos métodos ILS implementados, da mesma forma que o trabalho de RUIZ e STÜTZLE (2007).

5.2.5 Tipos de Busca Local

No capítulo 4 (seção 4.3) foi detalhado o procedimento de busca local implementado neste trabalho. Naquela seção, foi dito que foram implementadas duas formas de explorar locais do espaço de soluções, uma realizando troca de elementos na sequência e definindo assim a estrutura de vizinhança N^T e outra que realiza a

realocação de um elemento da sequência para outra posição da sequência, que define a estrutura de vizinhança N^R.

Além disso, em cada uma dessas formas de explorar locais do espaço de soluções foram desenvolvidas duas maneiras de realizar o retorno do procedimento de busca local. Essas duas maneiras de realizar o retorno do procedimento são denotadas por "primeiro melhor" e "melhor de todos".

Neste cenário, podemos afirmar que quatro formas de busca local foram implementadas. Os testes foram realizados num conjunto de 10 instâncias-teste com $n = 50$ e $m = 10$. As instâncias foram executadas com o método GRASP Básico com critério de parada definido em 5000 iterações. A cada 500 iterações o valor da função objetivo Z, da melhor solução encontrada até o momento, era impresso no arquivo de resposta.

O gráfico da figura 5.2 apresenta as médias dos valores de Z a cada 500 iterações em cada uma das 4 formas de busca local implementadas. Na legenda do gráfico da figura 5.2, tem-se que NR representa a estrutura de vizinhança N^R, NT representa a estrutura de vizinhança N^T, PM representa a forma de retorno "primeiro melhor" e MT a forma de retorno "melhor de todos". Dessa forma, NT-PM representa a estrutura de vizinhança N^T quando aplicada com retorno do procedimento "primeiro melhor".

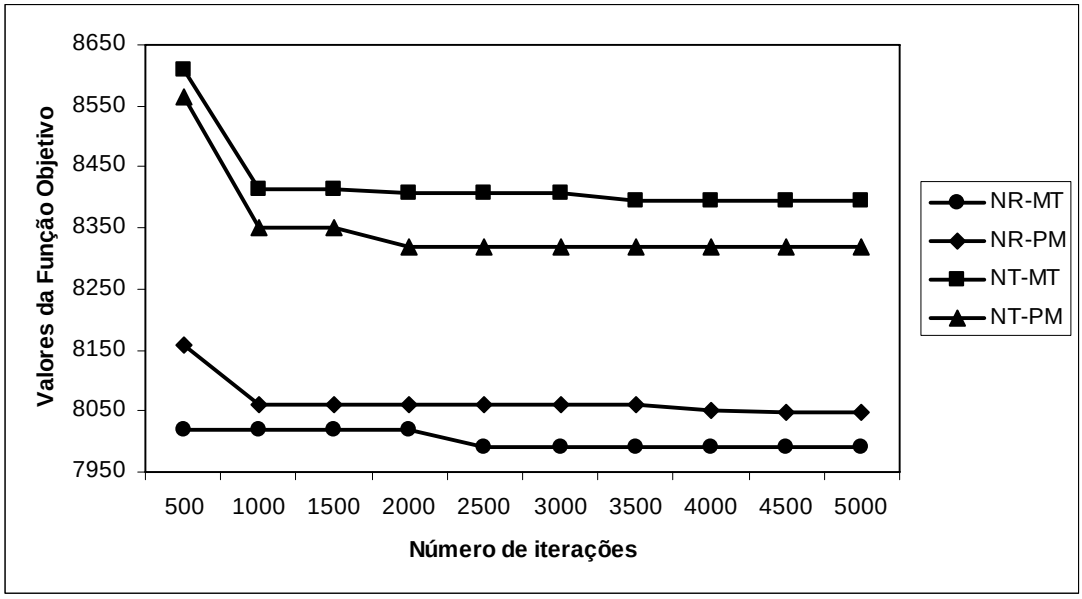


Figura 5.2. Média dos valores de *Z* a cada 500 iterações em cada forma de busca local

De acordo com o gráfico da figura 5.2, nota-se que a busca local que explora a estrutura de vizinhança N^R com procedimento de retorno "melhor de todos"

apresenta os melhores resultados durante toda a execução do teste, e por isso foi escolhida para ser utilizada na execução dos métodos propostos.

5.2.6 Parâmetros do *Path Relinking*

A partir desta seção agora iremos detalhar a escolha dos parâmetros e as justificativas na implementação da técnica de intensificação e diversificação *path relinking*. Na implementação da técnica *path relinking* o único parâmetro a ser definido é o tamanho do conjunto de soluções elite (E_{tam}).

Na definição desse parâmetro realizamos testes com os valores $E_{tam} = 5$ e $E_{tam} = 10$. Aplicamos o método GRASP Reativo com *Path Relinking* (GRPR) a um conjunto de 10 instâncias-teste escolhidas aleatoriamente no grupo *small*. Na primeira execução foi usado $E_{tam} = 5$, e em seguida, na segunda execução foi usado $E_{tam} = 10$.

Nos testes verificou-se que em todas as instâncias utilizadas, as soluções pertencentes a E quando $E_{tam} = 5$ também estavam presentes a E quando $E_{tam} = 10$. Ou seja, o conjunto interseção entre os conjuntos E da primeira e da segunda execução continha 5 elementos (soluções). No entanto, em algumas instâncias, o conjunto E não foi completamente preenchido, quando testado com $E_{tam} = 10$.

A intenção era aplicar o *path relinking* também como pós-otimização, de forma que após a execução do processo iterativo, o *path relinking* seria aplicado em cada par de soluções pertencentes a E. A melhor opção neste caso seria a utilização de $E_{tam} = 10$, pois aumentaria as chances do *path relinking* como pós-otimização encontrar soluções melhores. Por isso o tamanho máximo do conjunto elite E foi definido em 10 soluções, ou seja, $E_{tam} = 10$.

Conforme dito anteriormente, o *path relinking* quando aplicado como pós-otimização não obteve resultados satisfatórios. Essa verificação pode ser feita através da tabela 5.5, que mostra que a pós-otimização quando aplicada com cada um dos métodos: GRASP Reativo com *Path Relinking*(GRPR) e ILS com *Path Relinking* (ILSPR) não apresentou avanços significativos na média do valor de Z.

Tabela 5.5. Médias do valor de *Z* nos métodos propostos (GRPR e ILSPR) quando aplicados sem pós-otimização e com pós-otimização

	Médias do valor de *Z*			
	Small		***Large***	
	sem Pós-Otimização	**com Pós-Otimização**	**sem Pós-Otimização**	**com Pós-Otimização**
GRPR	**1.012,1**	**1.012,1**	18.556	**18.555**
ILSPR	**1.014,31**	**1.014,31**	18.376,39	**18.376,32**

Note que quando o *path relinking* foi aplicado como pós-otimização no grupo *small*, não se obteve nenhuma alteração nas médias do valor de *Z*, tanto para o GRPR quanto para o ILSPR. Já nos problemas *large*, a variação das médias de *Z* foi praticamente nula quando ambos os métodos foram aplicados com pós-otimização. A variação foi de apenas 1 unidade para o GRPR e 0,07 para o ILSPR. Dessa forma, foi decidido que nos métodos GRPR e ILSPR não seria utilizado o *path relinking* como pós-otimização.

Com a intenção de aumentar a diversificação do *path relinking*, foi definido que a técnica seria utilizada como *back* e *forward relinking*, ou seja, dois caminhos são explorados. Isto é, no segundo caminho a solução origem será a solução guia do primeiro caminho, e vice-versa. A razão para isso é que os caminhos de soluções podem ser diferentes e, consequentemente, as soluções analisadas.

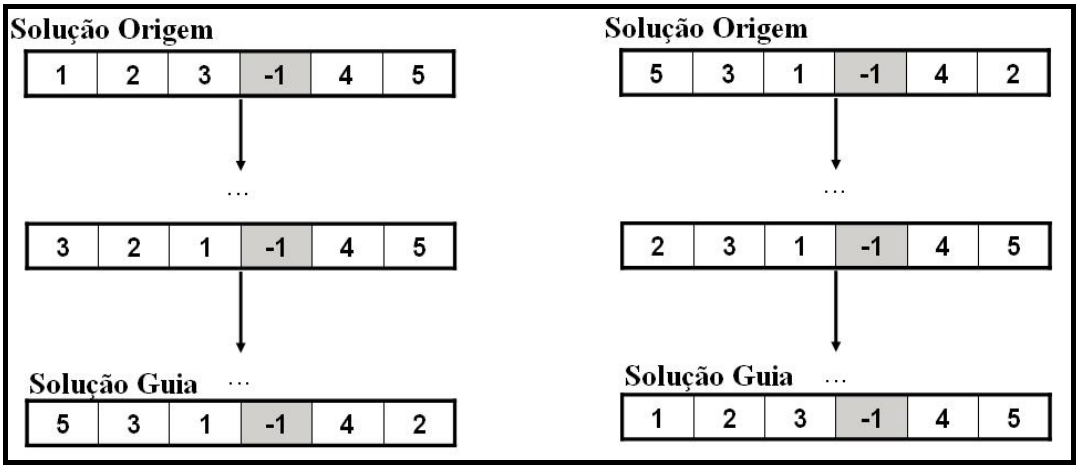


Figura 5.3. Exemplo de soluções diferentes encontradas no caminho de soluções quando aplicamos o *path relinking* em *back* e *forward relinking*

Observe na figura 5.3 que num exemplo com $n = 5$ e $m = 2$, ao construirmos o caminho de soluções entre a solução origem e a solução guia, encontramos a solução [3, 2, 1, -1, 4, 5] que não será encontrada ao realizarmos o caminho inverso. No

entanto, a solução [2, 3, 1, -1, 4, 5] é encontrada apenas no segundo caminho. Esse exemplo busca justificar a utilização do *path relinking* com *back* e *forward relinking*.

5.2.7 Definição do Critério de parada

Nesse trabalho, conforme dito anteriormente, o critério de parada dos métodos iterativos é baseado num tempo limite de execução, sendo esse tempo limite definido de acordo com o tamanho da instância-teste. Este tamanho é definido de acordo com o número de tarefas n e o número de máquinas m.

A partir de então, a dificuldade está em definir o critério de parada em termos de n e m. Com esse objetivo escolhemos 6 instâncias-teste com $n = 50$ e $m = 10$. Em seguida, executamos cada uma dessas instâncias utilizando o método GRPR com critério de parada definido em 2000 iterações e a cada 100 iterações o valor da melhor solução encontrada até o momento é impressa.

Na tabela 5.6 apresenta-se, na segunda coluna, o número mínimo de iterações que foram executadas em cada instância até se atingir uma estabilização, ou seja, a quantidade de iterações executadas até se obter a solução que seria considerada a melhor. Na terceira coluna é apresentado o tempo total gasto por cada instância durante a execução das 2000 iterações. Na análise da segunda coluna, nota-se que dentre as 6 instâncias selecionadas, o número mínimo de iterações executadas antes da estabilização foi de 800 iterações, o que corresponde a 40% do total do número de iterações. Por isso, na quarta coluna exibe-se um valor aproximado de 40% do tempo gasto, ou seja, o tempo médio que cada instância necessitava para encontrar a melhor solução.

A análise da tabela 5.6 nos permite imaginar um tempo ideal de execução próximo dos valores da quarta coluna e que, de certa forma, estejam relacionados com $n = 50$ e $m = 10$. O valor que mais se aproxima disso numa primeira análise é $(n \times m)/2$.

Com isso, realizamos mais alguns testes para definir se esse valor realmente é o ideal. Para isso, escolhemos 30 problemas do grupo *large*, sendo 10 com $n = 50$, outros 10 com $n = 75$ e os 10 restantes com $n = 100$. Aplicamos o método GRPR em cada um deles, definindo o critério de parada em $n \times m$ segundos. Após cada $(n \times m)/10$ segundos a melhor solução encontrada até o momento era impressa.

Tabela 5.6. Número mínimo de iterações e tempo médio de execução

Instâncias	Nº de Iterações até encontrar solução final	Tempo Total (s)	Tempo médio (40% do tempo total)
1	800	532	212,8
2	800	532	212,8
3	500	436	174,4
4	400	388	155,2
5	300	374	149,6
6	600	340	136,0

Ao final da execução, calculamos a média do valor de Z obtido a cada $(n \times m)/10$ segundos. Dessa forma, o gráfico da figura 5.4 apresenta o declínio da média de Z a cada $(n \times m)/10$ segundos, que neste caso está representado no gráfico pelo valor do tempo médio gasto.

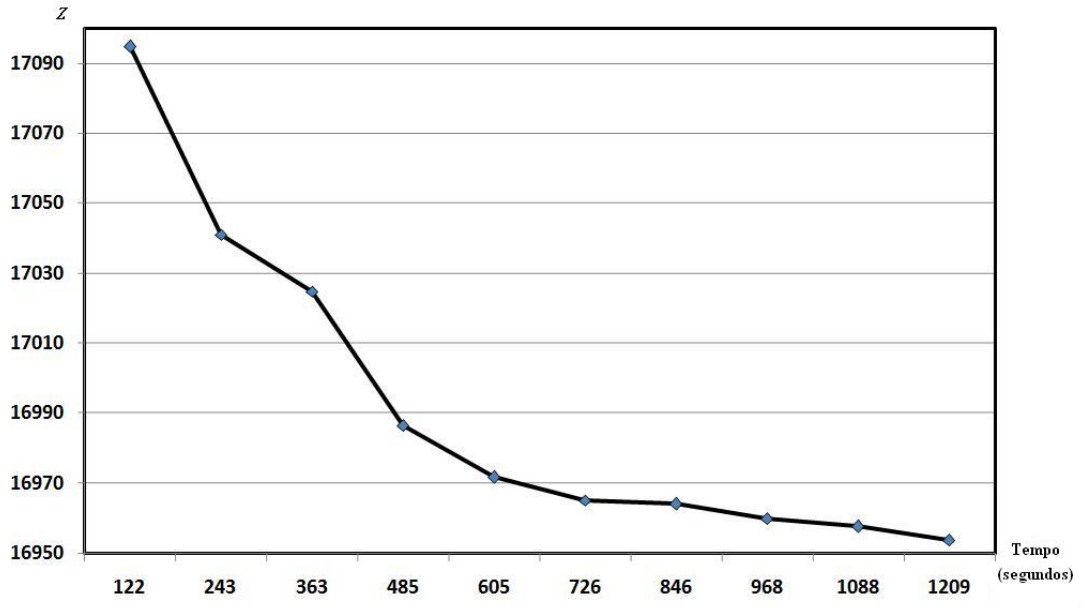


Figura 5.4. Declínio do valor médio de *Z* durante a execução de instâncias com parada em *n×m* segundos

Note que o gráfico na figura 5.4 apresenta um grande declínio na primeira metade, e em seguida mostra uma tendência de estabilização. Dos 30 problemas analisados apenas 7 exibiram uma solução melhor após a execução de $(n \times m)/2$ segundos. Assim, comprova-se que, em média, o critério de parada definido em $(n \times m)/2$ segundos é suficiente para encontrar as soluções definitivas, antes da

estabilização, em instâncias do grupo *large*. Nas instâncias do grupo *small* esse critério de parada também foi utilizado, e se mostrou mais que suficiente.

5.3 Resultados dos experimentos computacionais

Após a definição dos parâmetros utilizados nos métodos propostos e do detalhamento do ambiente de execução dos testes, o próximo passo será apresentar os resultados de aplicação de cada uma dos métodos nas 720 instâncias-teste.

Antes disso, vale lembrar que, neste trabalho, foram desenvolvidos 5 métodos, sendo 3 baseados na metaheurística GRASP: GRASP Básico (GB), GRASP Reativo (GR) e GRASP Reativo com *Path Relinking* (GRPR). Os outros 2 métodos restantes são baseados na metaheurística ILS: ILS Básico (ILS) e ILS com *Path Relinking* (ILSPR). Com a intenção de facilitar a visualização dos dados iremos denotar cada um dos métodos pela abreviatura do seu nome.

Conforme dito anteriormente, as 720 instâncias-teste do problema foram divididas em dois grupos: *small* e *large*, cada um com 360 instâncias. Além disso, em cada grupo, 36 combinações de valores de n, m, quantidade mínima e máxima de recursos e valor mínimo e máximo de *setup times* são possíveis. Para cada uma dessas combinações, em cada um dos grupos, foram geradas 10 instâncias de teste.

Dessa forma, nas tabelas abaixo, que apresentam os resultados obtidos em cada um dos métodos, exibe-se a média dos valores de Z para cada conjunto de 10 instâncias geradas na mesma combinação de valores de n, m, quantidade mínima e máxima de recursos e valor mínimo e máximo de *setup times*. Para facilitar a leitura e visualização dos resultados, dividimos os métodos em 2 grupos: GRASP e ILS, e os resultados são apresentados separadamente nos grupos *small* e *large*. Com isso temos 4 tabelas apresentando os resultados: GRASP-*small*, GRASP-*large*, ILS-*small* e ILS-*large*.

Nesta seção serão apresentados somente os resultados e, na seção seguinte, a comparação com outros resultados heurísticos, além da comparação entre os métodos propostos. Na seção seguinte, também, são apresentadas as discussões e análises dessas comparações.

Tabela 5.7. Médias do valor de *Z* nas instâncias-teste do grupo *small* obtidas pelos métodos GRASP

Instâncias				*Médias de Z*		
n	***m***	$S^-;S^+$	$R^-;R^+$	***GB***	***GR***	***GRPR***
6	3	[1,50];[50,100]	[1,3];[3;5]	6.338	**6.318**	**6.318**
			[1,5];[5;10]	7.183	**7.169**	**7.169**
		[50,100];[100,150]	[1,3];[3;5]	8.029	7.997	**7.986**
			[1,5];[5;10]	8.204	**8.057**	**8.057**
	4	[1,50];[50,100]	[1,3];[3;5]	4.405	**4.360**	**4.360**
			[1,5];[5;10]	4.291	**4.200**	**4.200**
		[50,100];[100,150]	[1,3];[3;5]	5.201	**5.168**	**5.168**
			[1,5];[5;10]	5.285	**5.238**	**5.238**
	5	[1,50];[50,100]	[1,3];[3;5]	2.806	2.745	**2.743**
			[1,5];[5;10]	2.504	**2.473**	**2.473**
		[50,100];[100,150]	[1,3];[3;5]	3.258	**3.181**	**3.181**
			[1,5];[5;10]	3.254	**3.131**	**3.131**
8	3	[1,50];[50,100]	[1,3];[3;5]	**11.029**	**11.029**	**11.029**
			[1,5];[5;10]	12.247	**12.236**	**12.236**
		[50,100];[100,150]	[1,3];[3;5]	14.812	**14.811**	**14.811**
			[1,5];[5;10]	**16.047**	**16.047**	**16.047**
	4	[1,50];[50,100]	[1,3];[3;5]	8.308	**8.287**	**8.287**
			[1,5];[5;10]	8.789	**8.772**	**8.772**
		[50,100];[100,150]	[1,3];[3;5]	10.045	**9.965**	**9.965**
			[1,5];[5;10]	10.717	**10.709**	**10.709**
	5	[1,50];[50,100]	[1,3];[3;5]	**5.748**	**5.748**	**5.748**
			[1,5];[5;10]	6.476	**6.459**	**6.459**
		[50,100];[100,150]	[1,3];[3;5]	7.196	**7.193**	**7.193**
			[1,5];[5;10]	7.955	**7.916**	**7.916**
10	3	[1,50];[50,100]	[1,3];[3;5]	**16.003**	**16.003**	**16.003**
			[1,5];[5;10]	**18.301**	**18.301**	**18.301**
		[50,100];[100,150]	[1,3];[3;5]	**22.866**	**22.866**	**22.866**
			[1,5];[5;10]	**25.870**	**25.870**	**25.870**
	4	[1,50];[50,100]	[1,3];[3;5]	**12.782**	**12.782**	**12.782**
			[1,5];[5;10]	13.292	**13.280**	**13.280**
		[50,100];[100,150]	[1,3];[3;5]	16.064	**16.051**	**16.051**
			[1,5];[5;10]	**16.950**	**16.950**	**16.950**
	5	[1,50];[50,100]	[1,3];[3;5]	9.172	9.147	**9.139**
			[1,5];[5;10]	10.005	**9.925**	**9.925**
		[50,100];[100,150]	[1,3];[3;5]	11.881	**11.848**	**11.848**
			[1,5];[5;10]	12.149	**12.144**	**12.144**

Tabela 5.8. Médias do valor de *Z* nas instâncias-teste do grupo *large* obtidas pelos métodos GRASP

Instâncias				*Médias de Z*		
n	***m***	$S^-;S^+$	$R^-;R^+$	***GB***	***GR***	***GRPR***
50	10	[1,50];[50,100]	[1,3];[3;5]	81.867	81.917	**81.821**
			[1,5];[5;10]	101.824	101.576	**101.157**
		[50,100];[100,150]	[1,3];[3;5]	132.603	132.697	**132.262**
			[1,5];[5;10]	152.616	152.668	**152.137**
	15	[1,50];[50,100]	[1,3];[3;5]	59.682	59.580	**59.522**
			[1,5];[5;10]	68.823	68.586	**68.569**
		[50,100];[100,150]	[1,3];[3;5]	91.327	91.287	**91.028**
			[1,5];[5;10]	99.717	99.829	**99.449**
	20	[1,50];[50,100]	[1,3];[3;5]	45.746	45.712	**45.631**
			[1,5];[5;10]	48.634	**48.534**	48.646
		[50,100];[100,150]	[1,3];[3;5]	66.211	**65.900**	65.935
			[1,5];[5;10]	69.197	69.500	**69.073**
75	10	[1,50];[50,100]	[1,3];[3;5]	148.315	148.069	**147.347**
			[1,5];[5;10]	198.381	198.327	**198.094**
		[50,100];[100,150]	[1,3];[3;5]	270.570	**270.416**	270.680
			[1,5];[5;10]	322.948	322.741	**322.655**
	15	[1,50];[50,100]	[1,3];[3;5]	116.423	**115.674**	115.873
			[1,5];[5;10]	145.266	144.975	**144.880**
		[50,100];[100,150]	[1,3];[3;5]	192.474	192.547	**192.242**
			[1,5];[5;10]	225.870	**224.618**	225.513
	20	[1,50];[50,100]	[1,3];[3;5]	95.033	**94.441**	94.625
			[1,5];[5;10]	111.201	111.389	**110.954**
		[50,100];[100,150]	[1,3];[3;5]	149.429	**149.086**	149.215
			[1,5];[5;10]	165.706	165.719	**165.072**
100	10	[1,50];[50,100]	[1,3];[3;5]	225.050	224.869	**224.617**
			[1,5];[5;10]	309.812	**309.431**	309.592
		[50,100];[100,150]	[1,3];[3;5]	450.458	450.091	**449.682**
			[1,5];[5;10]	534.009	534.074	**532.942**
	15	[1,50];[50,100]	[1,3];[3;5]	178.085	178.274	**178.029**
			[1,5];[5;10]	237.716	**237.265**	237.658
		[50,100];[100,150]	[1,3];[3;5]	321.988	321.392	**321.275**
			[1,5];[5;10]	383.844	383.384	**383.298**
	20	[1,50];[50,100]	[1,3];[3;5]	150.785	**150.767**	150.924
			[1,5];[5;10]	190.591	190.977	**190.284**
		[50,100];[100,150]	[1,3];[3;5]	254.326	253.632	**253.433**
			[1,5];[5;10]	297.001	296.107	**296.090**

Tabela 5.9. Médias do valor de *Z* nas instâncias-teste do grupo *small* obtidas pelos métodos ILS

Instâncias				*Médias de Z*	
n	***m***	$S^-;S^+$	$R^-;R^+$	***ILS***	***ILSPR***
6	3	[1,50];[50,100]	[1,3];[3;5]	**6.318**	**6.318**
			[1,5];[5;10]	**7.169**	**7.169**
		[50,100];[100,150]	[1,3];[3;5]	**7.986**	**7.986**
			[1,5];[5;10]	**8.057**	**8.057**
	4	[1,50];[50,100]	[1,3];[3;5]	4.365	**4.360**
			[1,5];[5;10]	**4.200**	**4.200**
		[50,100];[100,150]	[1,3];[3;5]	**5.168**	**5.168**
			[1,5];[5;10]	**5.238**	**5.238**
	5	[1,50];[50,100]	[1,3];[3;5]	**2.738**	**2.738**
			[1,5];[5;10]	**2.473**	**2.473**
		[50,100];[100,150]	[1,3];[3;5]	**3.148**	**3.148**
			[1,5];[5;10]	**3.128**	**3.128**
8	3	[1,50];[50,100]	[1,3];[3;5]	**11.029**	**11.029**
			[1,5];[5;10]	**12.262**	12.288
		[50,100];[100,150]	[1,3];[3;5]	**14.811**	14.826
			[1,5];[5;10]	**16.047**	**16.047**
	4	[1,50];[50,100]	[1,3];[3;5]	**8.287**	**8.287**
			[1,5];[5;10]	8.933	**8.896**
		[50,100];[100,150]	[1,3];[3;5]	**9.965**	**9.965**
			[1,5];[5;10]	10.781	**10.729**
	5	[1,50];[50,100]	[1,3];[3;5]	**5.748**	**5.748**
			[1,5];[5;10]	**6.462**	6.498
		[50,100];[100,150]	[1,3];[3;5]	**7.193**	**7.193**
			[1,5];[5;10]	**7.916**	**7.916**
10	3	[1,50];[50,100]	[1,3];[3;5]	**16.079**	**16.079**
			[1,5];[5;10]	**18.303**	**18.303**
		[50,100];[100,150]	[1,3];[3;5]	**22.866**	**22.866**
			[1,5];[5;10]	**25.912**	25.958
	4	[1,50];[50,100]	[1,3];[3;5]	**12.807**	**12.807**
			[1,5];[5;10]	**13.292**	13.433
		[50,100];[100,150]	[1,3];[3;5]	**16.092**	16.099
			[1,5];[5;10]	17.159	**17.056**
	5	[1,50];[50,100]	[1,3];[3;5]	**9.139**	**9.139**
			[1,5];[5;10]	9.984	**9.966**
		[50,100];[100,150]	[1,3];[3;5]	**11.848**	**11.848**
			[1,5];[5;10]	**12.177**	12.194

Tabela 5.10. Médias do valor de *Z* nas instâncias-teste do grupo *large* obtidas pelos métodos ILS

Instâncias				*Médias de Z*	
n	***m***	$S^-;S^+$	$R^-;R^+$	***ILS***	***ILSPR***
50	10	[1,50];[50,100]	[1,3];[3;5]	81.961	**81.914**
			[1,5];[5;10]	101.101	**100.895**
		[50,100];[100,150]	[1,3];[3;5]	132.090	**131.748**
			[1,5];[5;10]	**151.802**	151.956
	15	[1,50];[50,100]	[1,3];[3;5]	**59.081**	59.219
			[1,5];[5;10]	**67.973**	68.023
		[50,100];[100,150]	[1,3];[3;5]	**90.421**	90.751
			[1,5];[5;10]	**98.558**	98.706
	20	[1,50];[50,100]	[1,3];[3;5]	45.295	**45.152**
			[1,5];[5;10]	**48.110**	48.134
		[50,100];[100,150]	[1,3];[3;5]	**65.412**	65.515
			[1,5];[5;10]	**68.728**	68.734
75	10	[1,50];[50,100]	[1,3];[3;5]	**147.055**	147.450
			[1,5];[5;10]	195.235	**195.003**
		[50,100];[100,150]	[1,3];[3;5]	268.341	**268.109**
			[1,5];[5;10]	**318.812**	320.549
	15	[1,50];[50,100]	[1,3];[3;5]	**115.036**	115.394
			[1,5];[5;10]	143.760	**143.755**
		[50,100];[100,150]	[1,3];[3;5]	191.125	**190.454**
			[1,5];[5;10]	223.131	**222.499**
	20	[1,50];[50,100]	[1,3];[3;5]	**92.958**	93.279
			[1,5];[5;10]	109.588	**109.383**
		[50,100];[100,150]	[1,3];[3;5]	147.949	**147.667**
			[1,5];[5;10]	**163.691**	163.888
100	10	[1,50];[50,100]	[1,3];[3;5]	**220.202**	220.301
			[1,5];[5;10]	305.310	**304.140**
		[50,100];[100,150]	[1,3];[3;5]	**445.769**	445.974
			[1,5];[5;10]	528.246	**527.512**
	15	[1,50];[50,100]	[1,3];[3;5]	**176.067**	176.308
			[1,5];[5;10]	234.273	**233.794**
		[50,100];[100,150]	[1,3];[3;5]	**318.463**	318.618
			[1,5];[5;10]	**379.459**	380.591
	20	[1,50];[50,100]	[1,3];[3;5]	149.495	**148.795**
			[1,5];[5;10]	188.242	**187.704**
		[50,100];[100,150]	[1,3];[3;5]	251.372	**250.608**
			[1,5];[5;10]	293.397	**292.978**

As tabelas acima apresentam os resultados obtidos por cada método em cada grupo de instâncias-teste. Nestas tabelas os valores em negrito representam os melhores resultados médios encontrados dentre os valores listados na respectiva tabela.

5.4 Análise e discussão dos resultados

Na análise que se propõe nesta seção, o objetivo é mostrar a eficiência dos métodos propostos quando comparados entre si e com os melhores resultados conhecidos na literatura, que foram disponibilizados por RUIZ e ANDRÉS (2007).

As soluções disponibilizadas, por RUIZ e ANDRÉS (2007), para o grupo de instâncias *small* foram obtidas pelo software matemático CPLEX (versão 9.1) através da resolução do Modelo Matemático de Programação Inteira (MMPI), também proposto no mesmo trabalho. A execução do CPLEX foi limitada a 300 segundos para instancias com $n = 6$ tarefas e 3600 segundos para instâncias com $n = 8$ e $n = 10$ tarefas. Em todas as instâncias com $n = 6$ e $n = 8$, o CPLEX determinou a solução ótima. Já nas instâncias com $n = 10$ somente foram obtidas soluções aproximadas, pois o tempo de execução foi atingido. Na tabela 5.11 são exibidas as médias no valor de Z para instâncias do grupo *small* quando executadas pelo CPLEX.

Os resultados disponibilizados por RUIZ e ANDRÉS (2007), para as instâncias do grupo *large*, correspondem aos melhores resultados obtidos pelas heurísticas construtivas, que também foram propostas no trabalho de RUIZ e ANDRÉS (2007) e detalhadas no capítulo 3 deste trabalho. A tabela 5.12 exibe as médias no valor de Z para instâncias do grupo *large*, que foram obtidas pelos melhores resultados das heurísticas construtivas.

Nas etapas seguintes, serão realizadas comparações entre os métodos propostos e os resultados disponíveis na literatura. Para essas comparações utilizamos a medida da Porcentagem Média de Desvio (PMD), calculada pela seguinte fórmula:

$$PMD = \frac{1}{R}\sum_{r=1}^{R}\frac{(Z_{melhor} - Z_h)}{Z_{melhor}} \times 100\%$$

em que Z_{melhor} representa a solução com a qual estamos querendo comparar e Z_h representa a solução obtida no método heurístico proposto. Além disso, o valor R representa o número de instâncias avaliadas.

Dessa forma, se o valor de *PMD* for um valor x, de tal forma que x é positivo ($x > 0$), então as soluções obtidas pelos métodos heurísticos são em média x% melhores

que a solução que estamos utilizando como base para comparação. No caso contrário, ou seja, se x for negativo ($x < 0$), então as soluções obtidas pelos métodos heurísticos não conseguiram encontrar, em média, resultados melhores.

Tabela 5.11. Médias do valor de *Z* nas instâncias-teste do grupo *small* obtidas pelo software matemático CPLEX 9.1

Instâncias				*Médias de Z*
n	***m***	$S^-;S^+$	$R^-;R^+$	***CPLEX***
6	3	[1,50];[50,100]	[1,3];[3;5]	6.318*
			[1,5];[5;10]	7.169*
		[50,100];[100,150]	[1,3];[3;5]	7.986*
			[1,5];[5;10]	8.057*
	4	[1,50];[50,100]	[1,3];[3;5]	4.360*
			[1,5];[5;10]	4.200*
		[50,100];[100,150]	[1,3];[3;5]	5.168*
			[1,5];[5;10]	5.238*
	5	[1,50];[50,100]	[1,3];[3;5]	2.738*
			[1,5];[5;10]	2.473*
		[50,100];[100,150]	[1,3];[3;5]	3.148*
			[1,5];[5;10]	3.128*
8	3	[1,50];[50,100]	[1,3];[3;5]	11.029*
			[1,5];[5;10]	12.236*
		[50,100];[100,150]	[1,3];[3;5]	14.811*
			[1,5];[5;10]	16.047*
	4	[1,50];[50,100]	[1,3];[3;5]	8.287*
			[1,5];[5;10]	8.772*
		[50,100];[100,150]	[1,3];[3;5]	9.965*
			[1,5];[5;10]	10.709*
	5	[1,50];[50,100]	[1,3];[3;5]	5.748*
			[1,5];[5;10]	6.459*
		[50,100];[100,150]	[1,3];[3;5]	7.193*
			[1,5];[5;10]	7.916*
10	3	[1,50];[50,100]	[1,3];[3;5]	16.302
			[1,5];[5;10]	18.590
		[50,100];[100,150]	[1,3];[3;5]	23.441
			[1,5];[5;10]	26.326
	4	[1,50];[50,100]	[1,3];[3;5]	12.993
			[1,5];[5;10]	13.414
		[50,100];[100,150]	[1,3];[3;5]	16.462
			[1,5];[5;10]	17.245
	5	[1,50];[50,100]	[1,3];[3;5]	9.409
			[1,5];[5;10]	10.165
		[50,100];[100,150]	[1,3];[3;5]	12.314
			[1,5];[5;10]	12.785

*** Soluções Ótimas**

Fonte: RUIZ e ANDRÉS (2007)

Tabela 5.12. Médias do valor de *Z* nas instâncias-teste do grupo *large* obtidas pelos melhores resultados das heurísticas construtivas de RUIZ e ANDRÉS (2007)

Instâncias				*Médias de Z Resultados Heurísticos*
n	***m***	$S^-;S^+$	$R^-;R^+$	
50	10	[1,50];[50,100]	[1,3];[3;5]	92.615
			[1,5];[5;10]	112.832
		[50,100];[100,150]	[1,3];[3;5]	143.256
			[1,5];[5;10]	164.401
	15	[1,50];[50,100]	[1,3];[3;5]	67.979
			[1,5];[5;10]	74.990
		[50,100];[100,150]	[1,3];[3;5]	97.742
			[1,5];[5;10]	105.534
	20	[1,50];[50,100]	[1,3];[3;5]	49.928
			[1,5];[5;10]	53.859
		[50,100];[100,150]	[1,3];[3;5]	70.002
			[1,5];[5;10]	74.022
75	10	[1,50];[50,100]	[1,3];[3;5]	166.415
			[1,5];[5;10]	236.000
		[50,100];[100,150]	[1,3];[3;5]	286.680
			[1,5];[5;10]	354.873
	15	[1,50];[50,100]	[1,3];[3;5]	130.347
			[1,5];[5;10]	159.490
		[50,100];[100,150]	[1,3];[3;5]	205.400
			[1,5];[5;10]	236.376
	20	[1,50];[50,100]	[1,3];[3;5]	107.804
			[1,5];[5;10]	120.296
		[50,100];[100,150]	[1,3];[3;5]	160.067
			[1,5];[5;10]	172.409
100	10	[1,50];[50,100]	[1,3];[3;5]	246.576
			[1,5];[5;10]	359.014
		[50,100];[100,150]	[1,3];[3;5]	469.931
			[1,5];[5;10]	585.176
	15	[1,50];[50,100]	[1,3];[3;5]	195.793
			[1,5];[5;10]	270.150
		[50,100];[100,150]	[1,3];[3;5]	336.943
			[1,5];[5;10]	409.763
	20	[1,50];[50,100]	[1,3];[3;5]	167.631
			[1,5];[5;10]	206.661
		[50,100];[100,150]	[1,3];[3;5]	267.809
			[1,5];[5;10]	306.226

Fonte: RUIZ e ANDRÉS (2007)

Antes de realizarmos as comparações sugeridas, convém relembrar as diferenças na especificação e configuração das máquinas em que foi realizada as medições dos métodos propostos e a máquina utilizada por RUIZ e ANDRÉS (2007) para executar as medições das heurísticas construtivas. Apesar dessas diferenças, as

comparações realizadas, a seguir, mostram que a superioridade na qualidade das soluções obtidas pelos métodos propostos não pode ser justificada somente por este aspecto.

5.4.1 Comparação dos melhores resultados da literatura e os métodos propostos baseados na metaheurística GRASP-Grupo *small*

Na tabela 5.13 são apresentados os valores do *PMD* quando comparamos os resultados dos métodos baseados na metaheurística GRASP com os resultados disponíveis na literatura, para o grupo de instâncias *small*, obtidos pelo software CPLEX com limitação de tempo.

Na tabela 5.13, os valores negativos de *PMD* são destacados em itálico e os melhores valores em cada linha são destacados em negrito. Note que para as instâncias de $n = 6$, os três métodos GRASP não encontraram a solução ótima em todas as instâncias. Porém, observe que o método GR consegue melhorar (ou manter) o desempenho médio do método GB, e o mesmo ocorre entre GRPR e GR, respectivamente.

Nas instâncias com $n = 8$, apesar do método GB não encontrar a solução ótima em todas as instâncias, os métodos GR e GRPR assim o fizeram, demonstrando que o método GR já foi suficiente para encontrar tais soluções, mesmo sem ter a técnica *path relinking*. Por fim, nos problemas com $n = 10$, todos os métodos GRASP encontraram resultados médios superiores aos resultados aproximados encontrados pelo CPLEX. Além disso, nesse caso (instâncias com $n = 10$) os métodos GR e GRPR apresentaram resultados médios praticamente iguais. A exceção ficou por conta de algumas poucas instâncias, em que a utilização da técnica *path relinking* colaborou melhorando ainda mais a qualidade das soluções.

Os valores médios de *PMD* para os métodos GRASP no grupo *small* mostram uma grande diferença entre o método GB e os demais. Já os valores para os métodos GR e GRPR possuem pouca diferença, sendo que proporcionaram em média uma melhoria de 0,716% e 0,725%, respectivamente, quando comparados com os resultados obtidos pelo software CPLEX e disponibilizados por RUIZ e ANDRÉS (2007).

Tabela 5.13. PMD- Comparação dos resultados do CPLEX com os métodos baseados na metaheurística GRASP (grupo *small*)

Instâncias				*PMD (%)*		
n	***m***	$S^-;S^+$	$R^-;R^+$	***GB***	***GR***	***GRPR***
6	3	[1,50];[50,100]	[1,3];[3;5]	*-0,28*	**0,00**	**0,00**
			[1,5];[5;10]	*-0,19*	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	*-0,52*	*-0,13*	**0,00**
			[1,5];[5;10]	*-1,84*	**0,00**	**0,00**
	4	[1,50];[50,100]	[1,3];[3;5]	*-1,10*	**0,00**	**0,00**
			[1,5];[5;10]	*-2,20*	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	*-0,67*	**0,00**	**0,00**
			[1,5];[5;10]	*-0,80*	**0,00**	**0,00**
	5	[1,50];[50,100]	[1,3];[3;5]	*-2,80*	*-0,25*	***-0,17***
			[1,5];[5;10]	*-1,00*	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	*-3,54*	***-1,26***	***-1,26***
			[1,5];[5;10]	*-4,61*	***-0,07***	***-0,07***
8	3	[1,50];[50,100]	[1,3];[3;5]	**0,00**	**0,00**	**0,00**
			[1,5];[5;10]	*-0,08*	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	*-0,01*	**0,00**	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**	**0,00**
	4	[1,50];[50,100]	[1,3];[3;5]	*-0,24*	**0,00**	**0,00**
			[1,5];[5;10]	*-0,20*	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	*-0,80*	**0,00**	**0,00**
			[1,5];[5;10]	*-0,07*	**0,00**	**0,00**
	5	[1,50];[50,100]	[1,3];[3;5]	**0,00**	**0,00**	**0,00**
			[1,5];[5;10]	*-0,25*	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	*-0,04*	**0,00**	**0,00**
			[1,5];[5;10]	*-0,47*	**0,00**	**0,00**
10	3	[1,50];[50,100]	[1,3];[3;5]	**1,84**	**1,84**	**1,84**
			[1,5];[5;10]	**1,51**	**1,51**	**1,51**
		[50,100];[100,150]	[1,3];[3;5]	**2,44**	**2,44**	**2,44**
			[1,5];[5;10]	**1,74**	**1,74**	**1,74**
	4	[1,50];[50,100]	[1,3];[3;5]	**1,55**	**1,55**	**1,55**
			[1,5];[5;10]	0,96	**1,04**	**1,04**
		[50,100];[100,150]	[1,3];[3;5]	2,40	**2,49**	**2,49**
			[1,5];[5;10]	**1,62**	**1,62**	**1,62**
	5	[1,50];[50,100]	[1,3];[3;5]	2,35	2,58	**2,66**
			[1,5];[5;10]	1,39	**2,15**	**2,15**
		[50,100];[100,150]	[1,3];[3;5]	3,37	**3,67**	**3,67**
			[1,5];[5;10]	4,84	**4,88**	**4,88**
Média				0,12	0,716	**0,725**

5.4.2 Comparação dos melhores resultados da literatura e os métodos propostos baseados na metaheurística GRASP-Grupo *large*

A tabela 5.14 apresenta os valores do *PMD* quando comparamos os resultados dos métodos baseados na metaheurística GRASP com os resultados disponíveis na literatura, para o grupo de instâncias *large*. Estes resultados são os melhores resultados obtidos pelas heurísticas construtivas propostas por RUIZ e ANDRÉS (2007).

Nas instâncias do grupo *large*, todos os métodos GRASP, por utilizarem uma heurística construtiva na sua fase de construção, conseguiram melhorar, em todas as instâncias, os resultados considerados os melhores resultados heurísticos da literatura.

Observe que os valores em negrito da tabela 5.14 representam os maiores percentuais *PMD* dentre os três métodos GRASP. Note que nenhum desses valores aparece na coluna do método GB, o que nos faz concluir que o mecanismo reativo do GRASP foi eficientemente implementado e apresentou resultados satisfatórios. Além disso, observa-se que na maioria dos casos, o método GRPR conseguiu melhorar ainda mais os resultados do método GR, o que nos permite afirmar que a técnica *path relinking* é eficiente na diversificação e intensificação, ao criar um caminho entre duas soluções elite.

Da tabela 5.14, observa-se que as porcentagens médias de melhoria dos métodos GB, GR e GRPR são 8,12%, 8,23% e 8,33%, respectivamente. Em algumas instâncias, o método GRPR conseguiu melhorar os resultados heurísticos em até 16,06%. No pior dos casos, o método GB melhorou apenas em 3,01% os resultados heurísticos. Conforme dito anteriormente, os três métodos conseguiram melhorar os resultados heurísticos em todas as instâncias.

5.4.3 Comparação dos melhores resultados da literatura e os métodos propostos baseados na metaheurística ILS-Grupo *small*

Na tabela 5.15, apresentam-se os valores do *PMD* quando comparamos os resultados dos métodos baseados na metaheurística ILS com os resultados disponíveis na literatura, para o grupo de instâncias *small*, obtidos pelo software CPLEX.

Note que para as instâncias de n = 6, os dois métodos ILS não conseguiram encontrar a solução ótima em todas as instâncias. Porém, observe que o método ILSPR consegue melhorar (ou manter) o desempenho médio do método ILS Básico.

Tabela 5.14. PMD-Comparação dos melhores resultados heurísticos da literatura com os métodos baseados na metaheurística GRASP (grupo *large*)

Instâncias				***PMD (%)***		
n	***m***	$S^-;S^+$	$R^-;R^+$	***GB***	***GR***	***GRPR***
50	10	[1,50];[50,100]	[1,3];[3;5]	11,60	11,55	**11,65**
			[1,5];[5;10]	9,69	9,92	**10,28**
		[50,100];[100,150]	[1,3];[3;5]	7,41	7,35	**7,65**
			[1,5];[5;10]	7,14	7,11	**7,42**
	15	[1,50];[50,100]	[1,3];[3;5]	12,18	12,33	**12,42**
			[1,5];[5;10]	8,14	8,45	**8,50**
		[50,100];[100,150]	[1,3];[3;5]	6,55	6,59	**6,86**
			[1,5];[5;10]	5,45	5,36	**5,72**
	20	[1,50];[50,100]	[1,3];[3;5]	8,36	8,42	**8,59**
			[1,5];[5;10]	9,67	**9,85**	9,64
		[50,100];[100,150]	[1,3];[3;5]	5,40	**5,84**	5,79
			[1,5];[5;10]	6,51	6,09	**6,67**
75	10	[1,50];[50,100]	[1,3];[3;5]	10,82	10,98	**11,41**
			[1,5];[5;10]	15,94	15,96	**16,06**
		[50,100];[100,150]	[1,3];[3;5]	5,61	**5,66**	5,57
			[1,5];[5;10]	8,98	9,04	**9,06**
	15	[1,50];[50,100]	[1,3];[3;5]	10,68	**11,25**	11,10
			[1,5];[5;10]	8,90	9,08	**9,13**
		[50,100];[100,150]	[1,3];[3;5]	6,28	6,25	**6,40**
			[1,5];[5;10]	4,44	**4,97**	4,59
	20	[1,50];[50,100]	[1,3];[3;5]	11,83	**12,37**	12,21
			[1,5];[5;10]	7,53	7,37	**7,74**
		[50,100];[100,150]	[1,3];[3;5]	6,64	**6,86**	6,78
			[1,5];[5;10]	3,86	3,85	**4,23**
100	10	[1,50];[50,100]	[1,3];[3;5]	8,69	8,77	**8,87**
			[1,5];[5;10]	13,69	**13,80**	13,76
		[50,100];[100,150]	[1,3];[3;5]	4,14	4,21	**4,30**
			[1,5];[5;10]	8,73	8,73	**8,92**
	15	[1,50];[50,100]	[1,3];[3;5]	9,03	8,94	**9,06**
			[1,5];[5;10]	11,98	**12,15**	12,00
		[50,100];[100,150]	[1,3];[3;5]	4,43	4,60	**4,64**
			[1,5];[5;10]	6,32	6,43	**6,45**
	20	[1,50];[50,100]	[1,3];[3;5]	10,04	**10,05**	9,96
			[1,5];[5;10]	7,77	7,57	**7,91**
		[50,100];[100,150]	[1,3];[3;5]	5,03	5,29	**5,36**
			[1,5];[5;10]	3,01	3,30	**3,31**
Média				8,12	8,23	**8,33**

Tabela 5.15. PMD- Comparação dos resultados do CPLEX com os métodos baseados na metaheurística ILS (grupo *small*)

Instâncias				***PMD (%)***	
n	***m***	$S^-;S^+$	$R^-;R^+$	***ILS***	***ILSPR***
6	3	[1,50];[50,100]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**
	4	[1,50];[50,100]	[1,3];[3;5]	*-0,10*	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**
	5	[1,50];[50,100]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**
8	3	[1,50];[50,100]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	***-0,21***	*-0,41*
		[50,100];[100,150]	[1,3];[3;5]	**0,00**	*-0,10*
			[1,5];[5;10]	**0,00**	**0,00**
	4	[1,50];[50,100]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	*-1,84*	***-1,34***
		[50,100];[100,150]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	*-0,69*	***-0,20***
	5	[1,50];[50,100]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	***-0,04***	*-0,59*
		[50,100];[100,150]	[1,3];[3;5]	**0,00**	**0,00**
			[1,5];[5;10]	**0,00**	**0,00**
10	3	[1,50];[50,100]	[1,3];[3;5]	**1,37**	**1,37**
			[1,5];[5;10]	**1,50**	**1,50**
		[50,100];[100,150]	[1,3];[3;5]	**2,44**	**2,44**
			[1,5];[5;10]	**1,58**	1,41
	4	[1,50];[50,100]	[1,3];[3;5]	**1,35**	**1,35**
			[1,5];[5;10]	**0,96**	*-0,09*
		[50,100];[100,150]	[1,3];[3;5]	**2,24**	2,19
			[1,5];[5;10]	0,42	**0,96**
	5	[1,50];[50,100]	[1,3];[3;5]	**2,66**	**2,66**
			[1,5];[5;10]	1,57	**1,75**
		[50,100];[100,150]	[1,3];[3;5]	**3,67**	**3,67**
			[1,5];[5;10]	4,61	**4,48**
Média				**0,597**	0,585

Nas instâncias com n = 8, além dos métodos ILS e ILSPR não conseguirem encontrar a solução ótima em todas as instâncias, não foi possível estabelecer um padrão, pois conforme pode ser observado na tabela 5.15, em algumas instâncias o

ILSPR encontrou resultados mais próximos da solução ótima que o ILS Básico, mas em outras ocorreu o caso contrário. Por fim, em quase todas as instâncias com n = 10, os métodos ILS Básico e ILSPR encontraram resultados médios superiores aos resultados aproximados encontrados pelo CPLEX.

A média final do valor de *PMD* para os métodos ILS e ILSPR no grupo *small* foi de aproximadamente 0,597% e 0,585%, respectivamente. Com esses resultados, nota-se que, em média, a utilização da técnica *path relinking* não se mostrou eficiente. Esse fato pode ser facilmente justificado, pois com a inclusão do *path relinking* ao método ILS Básico, tendo o mesmo critério de parada ($n \times m/2$ segundos), o número de iterações e, consequentemente, o número de soluções perturbadas, são menores.

5.4.4 Comparação dos melhores resultados da literatura e os métodos propostos baseados na metaheurística ILS-Grupo *large*

A tabela 5.16 apresenta os valores do *PMD* quando comparamos os resultados dos métodos baseados na metaheurística ILS com os resultados disponíveis na literatura, para o grupo de instâncias *large*. Estes resultados são os melhores resultados obtidos pelas heurísticas construtivas propostas por RUIZ e ANDRÉS (2007).

Nas instâncias do grupo *large*, todos os métodos ILS, por utilizarem uma heurística construtiva na sua fase inicial de obtenção de uma solução ótima local, conseguiram melhorar, em todas as instâncias, os resultados considerados os melhores resultados heurísticos da literatura.

Note que, na maioria dos casos, o método ILSPR conseguiu melhorar ainda mais os resultados do método ILS Básico, o que pode ser observado pelo valor médio do *PMD* de 9,13% e 9,145%, respectivamente, o que nos leva a afirmar que a técnica *path relinking* se mostrou eficiente a um custo maior.

Em algumas instâncias, o método ILSPR conseguiu melhorar os resultados heurísticos em até 17,37%. No pior dos casos, o método ILS melhorou em 4,19% os resultados heurísticos. Os métodos ILS e ILSPR conseguiram melhorar os resultados heurísticos em todas as instâncias.

Tabela 5.16. PMD-Comparação dos melhores resultados heurísticos da literatura com os métodos baseados na metaheurística ILS (grupo *large*)

Instâncias				*PMD (%)*	
n	***m***	$S^-;S^+$	$R^-;R^+$	***ILS***	***ILSPR***
50	10	[1,50];[50,100]	[1,3];[3;5]	11,51	**11,56**
			[1,5];[5;10]	10,34	**10,55**
		[50,100];[100,150]	[1,3];[3;5]	7,77	**8,01**
			[1,5];[5;10]	**7,63**	7,55
	15	[1,50];[50,100]	[1,3];[3;5]	**13,07**	12,87
			[1,5];[5;10]	**9,29**	9,20
		[50,100];[100,150]	[1,3];[3;5]	**7,47**	7,14
			[1,5];[5;10]	**6,57**	6,43
	20	[1,50];[50,100]	[1,3];[3;5]	9,27	**9,55**
			[1,5];[5;10]	**10,65**	10,62
		[50,100];[100,150]	[1,3];[3;5]	**6,54**	6,39
			[1,5];[5;10]	**7,14**	7,13
75	10	[1,50];[50,100]	[1,3];[3;5]	**11,57**	11,35
			[1,5];[5;10]	17,27	**17,37**
		[50,100];[100,150]	[1,3];[3;5]	6,39	**6,47**
			[1,5];[5;10]	**10,15**	9,66
	15	[1,50];[50,100]	[1,3];[3;5]	**11,75**	11,47
			[1,5];[5;10]	**9,84**	**9,84**
		[50,100];[100,150]	[1,3];[3;5]	6,94	**7,27**
			[1,5];[5;10]	5,60	**5,86**
	20	[1,50];[50,100]	[1,3];[3;5]	**13,75**	13,46
			[1,5];[5;10]	8,86	**9,04**
		[50,100];[100,150]	[1,3];[3;5]	7,57	**7,75**
			[1,5];[5;10]	**5,04**	4,92
100	10	[1,50];[50,100]	[1,3];[3;5]	**10,67**	10,62
			[1,5];[5;10]	14,95	**15,28**
		[50,100];[100,150]	[1,3];[3;5]	**5,13**	5,09
			[1,5];[5;10]	9,72	**9,84**
	15	[1,50];[50,100]	[1,3];[3;5]	**10,06**	9,95
			[1,5];[5;10]	13,25	**13,44**
		[50,100];[100,150]	[1,3];[3;5]	**5,47**	5,43
			[1,5];[5;10]	**7,39**	7,11
	20	[1,50];[50,100]	[1,3];[3;5]	10,81	**11,23**
			[1,5];[5;10]	8,90	**9,16**
		[50,100];[100,150]	[1,3];[3;5]	6,13	**6,42**
			[1,5];[5;10]	4,19	**4,32**
Média				9,130	**9,145**

5.4.5 Comparação dos métodos propostos

Nesta seção, propõe-se uma comparação entre os cinco métodos propostos neste trabalho. Para realizar a comparação em cada um dos grupos: *small* e *large*, exibe-se nas tabelas 5.17 e 5.18, respectivamente, a média dos valores de *Z* para cada combinação possível de valores de *n* e *m*. Os valores em itálico são as maiores médias encontradas e os valores em negrito os menores. Estes valores serão o Z_{melhor} e Z_h, respectivamente, que serão usados para calcular o *PMD*.

Tabela 5.17. Médias do valor de *Z* para todos os métodos propostos quando aplicados no grupo *small*

n	***m***	***Médias de Z***				
		GB	***GR***	***GRPR***	***ILS***	***ILSPR***
6	3	*744*	739	**738**	**738**	**738**
	4	*480*	**474**	**474**	**474**	**474**
	5	*296*	288	288	**287**	**287**
8	3	**1.353**	**1.353**	**1.353**	1.354	**1.355**
	4	946	**943**	**943**	*949*	947
	5	*684*	**683**	**683**	**683**	*684*
10	3	**2.076**	**2.076**	**2.076**	2.079	*2.080*
	4	**1.477**	**1.477**	**1.477**	1.484	*1.485*
	5	*1.080*	1.077	**1.076**	1.079	1.079
Médias		*1.015,1*	1.012,2	**1.012,0**	1.014,1	1.014,3

A tabela 5.17 nos mostra que, para as instâncias do grupo *small*, em média, os métodos baseados na metaheurística GRASP: GB e GRPR apresentam os maiores e menores valores médios de *Z*, respectivamente. Dessa forma, apesar do método ILS Básico ter resultados melhores que o método ILSPR e, consequentemente, indicar que a técnica *path relinking* aplicada junto com a metaheurística ILS, não mostrou a eficiência desejada, pode se afirmar que, no caso contrário, ou seja, ao ser utilizada no método GRPR, proporciona os melhores resultados médios dentre todos os métodos propostos para o conjunto de instâncias do grupo *small*.

Note também que, em média, para os menores valores de *n* os métodos ILS apresentaram-se competitivos com os valores dos métodos GRASP, principalmente o GRPR. Já para valores maiores de *n*, o contrário pôde ser observado.

Tabela 5.18. Médias do valor de *Z* para todos os métodos propostos quando aplicados no grupo *large*

n	*m*	*Médias de Z*				
		GB	***GR***	***GRPR***	***ILS***	***ILSPR***
50	10	*11.723*	11.721	11.684	11.674	**11.663**
	15	*7.989*	7.982	7.964	**7.901**	7.917
	20	*5.745*	5.741	5.732	5.689	**5.688**
75	10	*23.505*	23.489	23.469	**23.236**	23.278
	15	*17.001*	16.945	16.963	16.826	**16.803**
	20	*13.034*	13.016	12.997	**12.855**	**12.855**
100	10	*37.983*	37.962	37.921	37.488	**37.448**
	15	*28.041*	28.008	28.007	**27.707**	27.733
	20	*22.318*	22.287	22.268	22.063	**22.002**
Médias		*18.593,2*	18.572,3	18.556,1	18.382,1	**18.376,3**

Na análise da tabela 5.18, é facilmente percebido que os melhores valores médios para o grupo *large* foram encontrados pelos métodos ILS Básico e ILSPR. Apesar da diferença da média final entre os dois métodos ser praticamente nula, o método *ILSPR* conseguiu se destacar. Observe que o método GB foi o que apresentou, novamente, os resultados de pior qualidade, em todos os casos.

Portanto, para as instâncias do grupo *small*, o método GRPR apresentou em média, os melhores resultados, enquanto que, para as instâncias do grupo *large*, o método que obteve o mesmo feito foi o ILSPR. Destaca-se que esses dois métodos possuem a técnica de intensificação e diversificação *path relinking* acoplada, o que nos garante afirmar que essa técnica permitiu aos métodos encontrar soluções de melhor qualidade e mostrando, dessa forma, sua eficiência.

5.4.6 Análise de tempo de execução

Conforme dito anteriormente, as instâncias do grupo *small* foram executadas no software CPLEX por RUIZ e ANDRÉS (2007) com um tempo limite de execução em 300 segundos para instâncias com $n = 6$ e 3600 segundos para instâncias com $n = 8$ e $n = 10$.

A partir disso, note que o tempo máximo de execução dos métodos propostos no grupo de instâncias *small* foi de 25 segundos, pois os maiores valores de n e m nesse grupo são $n = 10$ e $m = 5$. Cada uma destas instâncias foi executada pelo CPLEX durante, no mínimo, 300 segundos.

Os resultados disponibilizados na literatura para as instâncias do grupo *large* foram obtidas pelos melhores resultados das heurísticas construtivas propostas. A construção de uma solução por uma heurística construtiva é muito rápida e, por isso, o tempo utilizado por essas heurísticas é praticamente nulo e não disponível na literatura. Dessa forma, não foi possível realizar comparações com relação ao tempo gasto para esse grupo de instâncias.

6 CONCLUSÕES

A partir dos estudos e pesquisas realizadas, foram propostos métodos que se baseiam nas metaheurísticas *Greedy Randomized Adaptative Search Procedures* (GRASP) e *Iterated Local Search* (ILS) para resolução de um problema de sequenciamento de tarefas em máquinas paralelas. Nesse problema, além das máquinas serem não-relacionadas, o tempo de preparação (*setup time*) entre duas tarefas não é dependente apenas da sequência, mas também da quantidade de recursos associado, que pode variar entre um valor mínimo e máximo. Neste trabalho, o problema foi denotado pela sigla PPTMPSR.

O mecanismo reativo do GRASP foi utilizado para garantir o auto-ajuste do parâmetro de aleatoriedade (α). A técnica de intensificação e diversificação *path relinking* também foi acoplada aos métodos desenvolvidos anteriormente.

Os parâmetros dos métodos foram detalhados no capítulo 5 da mesma forma que os resultados obtidos nos testes computacionais. Por sua vez, os resultados mostram que os objetivos do trabalho foram alcançados, ou seja, a utilização de metaheurísticas, que usam heurísticas construtivas na sua estrutura, ajudam a melhorar ainda mais a qualidade das soluções. Além disso, as soluções dos métodos propostos foram encontradas com um tempo computacional relativamente curto.

Os métodos propostos encontraram resultados satisfatórios em instâncias de pequeno e grande porte do problema. Nas instâncias de pequeno porte, em que a solução ótima era conhecida, as metaheurísticas encontraram o mesmo valor em várias delas, sendo que nas demais as soluções encontradas foram muito próximas da solução ótima.

Para as instâncias do problema em que a solução ótima não era conhecida, todos os métodos encontraram soluções superiores ou iguais em relação à melhor solução conhecida. Nas instâncias de pequeno porte, os métodos baseados na metaheurística GRASP, obtiveram em média resultados superiores aos obtidos pela metaheurística ILS. O método GRASP com o mecanismo reativo e a técnica *path relinking*, em média, melhorou em 0,725% os resultados obtidos pelo software

CPLEX com limitação de tempo, sendo que em algumas instâncias a melhoria foi maior que 3%.

Nas instâncias de grande porte, os métodos baseados na metaheurística ILS, obtiveram, em média, resultados superiores aos obtidos pela metaheurística GRASP. O método ILS, que possuía a técnica *path relinking* acoplada, apresentou uma pequena vantagem sobre o método ILS, de tal forma que encontrou soluções, em média, 9,14% melhores que as soluções conhecidas. Em algumas instâncias, a melhoria foi maior que 10%.

Os resultados também demonstram que o mecanismo reativo no método GRASP foi eficiente à medida que, em média, possibilitou encontrar resultados melhores que o método GRASP sem o mecanismo. Da mesma forma, a utilização da técnica *path relinking* se mostrou eficiente ao ser aplicada juntamente com o método GRASP reativo, pois melhorou, em média, os resultados do respectivo método, tanto para instâncias de pequeno porte, quanto para as de grande porte. Porém, quando aplicada junto com o método ILS, a sua utilização se mostrou eficiente apenas nas instâncias de grande porte, uma vez que nas instâncias de pequeno porte não obteve resultados melhores.

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

6.1 Trabalhos futuros

Os métodos que podem ser aplicados ao PPTMPSR não se limitam aos que foram apresentados neste trabalho. Uma possível extensão para esse trabalho é a realização de novos estudos na modelagem do problema, com a intenção de utilizar outros critérios na função objetivo, como por exemplo, minimizar o atraso e adiantamento na entrega das tarefas que, por sua vez, possuem datas de entrega pré-estabelecidas. Além disso, o problema pode ser estudado numa versão multi-objetivo da função Z, já que os critérios utilizados neste trabalho são conflitantes.

Na implementação das metaheurísticas GRASP e ILS se propõe algumas melhorias, por exemplo, para a metaheurística ILS a sugestão é testar outras formas de perturbação e do critério de aceitação. Uma sugestão bastante pertinente é o

estudo e a implementação de outras metaheurísticas, tais como, algoritmos genéticos, *Simulated Annealing* e *Scatter Search*. Além disso, a construção e a utilização de um método híbrido, que combine as melhores técnicas das metaheurísticas GRASP e ILS, também pode ser sugerido como trabalho a ser implementado.

O método de busca local implementado neste trabalho, utilizado por todos os métodos, explora a estrutura de vizinhança N^R. Em trabalhos futuros o método de busca local pode explorar as estruturas de vizinhança N^R e N^T de forma independente, ou seja, inicialmente se explora uma dessas estruturas e, após a definição da melhor solução nessa estrutura, inicia-se a partir dela a exploração da outra estrutura. Além disso, podem ser utilizadas outras formas de movimentos na sequência, de forma a se obter outra estrutura de vizinhança, ainda não utilizada.

Na técnica *path relinking* a sugestão é construir um caminho de soluções entre uma solução de E e uma outra solução garantidamente bastante diferente, para que desta maneira tente-se escapar de soluções ótimos locais.

Por fim, o critério de parada utilizado neste trabalho pode ser alterado para que o tempo de execução dos métodos seja melhor aproveitado. Um exemplo de critério de parada que pode ser implementado é o critério baseado num tempo limite de execução, de acordo com o tamanho da instância. No entanto, se o método ficar alguma fração (definida previamente) do tempo limite sem encontrar soluções melhores, a execução é interrompida e a melhor solução encontrada até o momento é retornada. Com esse critério, a idéia é definir o tempo de execução dos métodos de acordo com a necessidade de cada instância e relacionado com o tamanho da instância.

REFERÊNCIAS BIBLIOGRÁFICAS

AIEX, R.M.; BINATO, S.; RESENDE, M.G.C. Parallel grasp with path-relinking for job shop scheduling. **Parallel Comput.,** 29(4), 393–430, 2003.

AIEX, R.M.; RESENDE, M.G.C.; PARDALOS, P.M.; TORALDO, G. Grasp with path relinking for three-index assignment. **INFORMS Journal on Computing**, 17(2), 224–247, 2005.

ALBA, E.; TALBI, E.G.; LUQUE, G.; MELAB, N. Metaheuristics and Parallelism. **John Wiley & Sons**, 2005.

ALIDAEE, B. Schedule of n jobs on two identical machines to minimize weighted mean flow time. **Computers & Indutrial Engineering**. London, v. 24, n. 1, p. 53-55, 1993.

ALLAHVERDI, A.; GUPTA, J.N.D.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. **OMEGA, The international Journal of Management Science**, 27(2):219–239, 1999.

ALLAHVERDI, A.; NG, C.T.; CHENG, T.C.E.; KOVALYOV, M.Y. A survey of scheduling problems with setup times or costs. **European Journal of Operational Research**. 187(3), 985-1032, 2008.

AMBÜHL, C; MASTROLILLI, M. On-line scheduling to minimize Max flow time: na optimal preemptive algorithm. **Operations Research Letters**. Amsterdam, v. 33, p. 597-602, 2005.

BÄCK, T.; FOGEL, D.B.; MICHALEWICZ, Z. **Handbook of Evolucionary Computation**. Oxford University Press, 1997.

BAKER, K.R. Introduction to sequencing and scheduling. New York: **John Wiley & Sons, Inc**, 1974.

BALAKRISHNAN, N.; KANET, J.J.; SHIDHARAN, S.V. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. **Computers & Operations Research**, Oxford, v. 26, p. 127-141, 1999.

BARROS, A.D.; MOCCELLIN, J.V. Análise da flutuação do gargalo em flowshop permutacional com tempos de setup assimétricos e dependentes da sequência. **Gestão & Produção**, v. 11, n. 1, p. 101-108, 2004.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimisation: Overview and conceptual comparison. **Technical Report 13**, Université Libre de Bruxelles, 2001.

BURBIDGE, J.L. The introduction of group technology. London: **Heinemann**, 1975.

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

CHENG, T.C.E.; SIN, C.C.S. A state-of-the-art review of parallel-machine scheduling research. **European Journal of Operational Research**, Amsterdan, v. 47, n. 3, p. 271-292, 1990.

CONGRAM, R.K.; POTTS, C.N.; VANE DE VELDE, S.L. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. **INFORMS Journal on Computing**, 14(1), 52–67, 2002.

DORIGO, M. Optimization, Learning and Natural Algorithms. Phd thesis, Politécnico di Milano, Italy, 1992.

FEO, T.A.; RESENDE, M.G. C. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, 6(2):109-133, 1995.

FEO, T.A.; SARATHY, K.; MCGAHAN, J. A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties. **Comput. Oper. Res.**, 23(9), 881–895, 1996.

FRENCH, S. Sequencing ans scheduling: an introduction to the mathematics of the job shop. **New York: Wiley**, 1992.

GLOVER, F. Tabu search and adaptive memory programming - advances, applications and challenges. In: BARR, R.S.; HELGASON, R.V.; KENNGINGTON, J.L. editors. **Interfaces in Computer Science and Operations Research**, 1–75, 1996.

GLOVER, F. Tabu search - part i. **ORSA Journal of Computing**, 1:190-206, 1989.

GLOVER, F. A Template for Scatter Search and Path Relinking, in Artificial Evolution. In HAO, J.-K.; LUTTON, E.; RONALD, E.; SCHOENAUER, M.; SNYERS, D. (eds.), **Lecture Notes in Computer Science**, Springer 1363, pp. 13–54, 1998.

GLOVER, F.; KOCHENBERGER, G. **Handbook of Metaheuristics**. Kluwer Academic Publishers, 2002.

GLOVER, F.; LAGUNA, M.; MARTÍ, R. Fundamentals of Scatter Search and Path Relinking. **Control and Cybernetics** 29, 653–684, 2000.

GOMES, H.A.S.; NETO, J.F.B. Utilização de metaheurística na programação de escala de pessoal em empresas de transporte coletivo por ônibus. **Anais do XXXV Simpósio Brasileiro de Pesquisa Operacional**, pp. 894-905, 2003.

GUINET, A. Textile production systems: a succession of non-identical parallel processor shops. **Journal of the Operational Research Society**, 42(8):655–671, 1991.

GUPTA, J.N.D.; RUIZ-TORRES, A.J. Generating efficient schedules for identicall parallel machines involving flow-time and tardy jobs. **European Journal of Operational Research**. Amsterdam, v. 167, p. 679-695, 2005.

KIM, D.W.; KIM, K.H.; JANG, W.; CHEN, F.F. Unrelated parallel machine scheduling with setup times using simulated annealing. **Robotics and Computer Integrated Manufacturing**, 18, 223–231, 2002.

KIRKPATRICK, S.; GELLAT, C.D.; VECCHI, M.P. Optimization by simulated annealing. **Science**, 220(4598):671–680, 1983.

LÓPEZ, F.G.; TORRES, M.G.; BATISTA, B.M.; PÉREZ, J.A.M.; MORENO-VEGA, J.M. Solving feature subset selection problem by a parallel scatter search. **European Journal of Operational Research**, 169(2):477-489, 2006.

LOURENÇO, H.R.; MARTIN, O.; STÜTZLE, T. Iterated Local Search. In F. GLOVER, F.; KOCHENBERGER, G., editors, **Handbook of Metaheuristics**, p. 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

MACCARTHY, B.L.; LIU, J.Y. Adressing a gap in scheduling research – a review of optimization and heuristic methods in production scheduling. **International Journal of Production Research**, London, v. 31, n. 1, p. 59–79, 1993.

MARSH, J.D.; MONTGOMERY, D.C. Optimal procedures for scheduling jobs with sequence-dependent changeover times on parallel processors. **AIIE Technical Papers**, p. 279–286, 1973.

MELO, V.A.; MARTINHON, C.A. Metaheurísticas híbridas para o problema do caixeiro viajante com coleta de prêmios. **Anais do XXXVI Simpósio Brasileiro de Pesquisa Operacional**, p. 1295-1306, 2004.

MLADENOVIC, N.; HANSEN, P. Variable neighborhood search. **Computers and Operations Research**, 24:1097-1100, 1997.

MÜHLENBEIN, H.; MAHNIG, T.; OCHOA, A. Schemata, distributions and graphical models in evolutionary optimization. **Journal of Heuristics**, 5(2):215-247, 1999.

NOWICKI, E.; ZDRZALKA, S. A survey of results for sequencing problems with controllable processing times. **Discrete Applied Mathematics**, 26(2-3):271–287, 1990.

OSMAN, I.; POTTS, C. Simulated annealing for permutation flow-shop scheduling. **Omega**, 17(6), 551–557, 1989.

PRAIS, M.; RIBEIRO, C.C. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. **INFORMS Journal on Computing**, 12(3), 164–176, 2000.

PINEDO, M. Scheduling: Theory, Algorithms, and Systems. **Prentice Hall, Upper Saddle, N.J, second edition**, 2002.

RANGEL, M.C.; ABREU, N.M.M.; BOAVENTURA-NETTO, P.O. Grasp para o pqa: um limite de aceitação para soluções iniciais. **Pesquisa Operacional**, 20(1):45-58, 2000.

RADHAKRISHNA, S.; VENTURA, J.A. Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. **International Journal of Production Research**, 38(10), 2233–2252, 2000.

REDDY, V.; NARENDRAN, T.T. Heuristics for scheduling sequence-dependent set-up jobs in flow line cells. **International Journal of Production Research**, Oxon, v. 41, n.1, p. 193-206, 2003.

RESENDE, M.; RIBEIRO, C. Handbook of Metaheuristics, chapter Greedy randomized adaptive search procedures. **Kluwer**, 2003.

RESENDE, M.; RIBEIRO, C. **GRASP with path-relinking: recent advances and applications**. IBARAKI, T.; NONOBE K.; YAGIURA, M. p. 29-63, 2005.

ROBINSON, A. Modern approaches to manufacturing improvement: The shingo System. Portland: **Productivity Press**, 1990.

RUIZ, R.; ANDRÉS, C. Unrelated parallel machines scheduling with resource-assignable sequence dependent setup times. Em Baptiste, P., Kendall, G., Munier-Kordon, A., Sourd, F., eds.: **Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)**, Paris, France, 439–446, 2007.

RUIZ, R.; MAROTO, Y.C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine elegibility. **European Journal of Operational Research**, 2004.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. Two new robust genetic algorithms for the flowshop scheduling problem,**Omega**, 34: 461–476, 2006.

RUIZ, R.; STÜTZLE, T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. **European Journal of Operational Research**, 187(3), 1143–1159, 2008.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, 177(3), 2033–2049, 2007.

SBC – Sociedade Brasileira de Computação. **Grandes Desafios da Pesquisa em Computação no Brasil – 2006 – 2016**, 2006. Disponível em: <http://www.sbc.org.br/index.php?language=1&content=downloads&id=272>. Acesso em: 08 Agosto 2009.

SERRA, T.; MOURA, A.V. Escalonamento integrado para o transporte coletivo utilizando grasp, path-relinking e paralelismo**. Anais do XXXVIII Simpósio Brasileiro de Pesquisa Operacional**, pp. 963-974, 2006.

SHINGO, S. Sistemas de produção com estoque zero: o Sistema Shingo para melhorias contínuas. trad Lia Weber Mendes. Porto Alegre: **Bookman**, 1996.

SIVRIKAYA-SERIFOGLU, F.; ULUSOY, G. Parallel machine scheduling with earliness and tardiness penalties. **Computers & Operations Research**, 26(8):773–787, 1999.

SOUZA, A.B.D.; MOCCELLIN, J.V. Metaheurística híbrida algoritmo genético Busca-Tabu para programação de operações flow shop. In: **Simpósio Brasileiro de Pesquisa Operacional**, XXXII SBPO, Viçosa-MG. Anais. Rio de Janeiro: SOBRAPO, 2000.

TAMINI, S.A.; RAJAN, V.N. Reduction of total weighted tardiness on uniform machines with sequence dependent setups. **Industrial Engineering Research Conference Proceedings**, 181–185, 1997.

VOLMANN, T.E.; BERRY, W.L.; WHYBARK D.C. **Manufacturing Planning and Control Systems**, Dow Jones-Irwin, 2ª edição, 1988.

WEBSTER, S.T. The complexity of scheduling job families about a common date. **Operations Research Letters**, 20(2), 65–74, 1997.

WEBSTER, S.T. Weighted flowtime bounds for scheduling identical processors. **European Journal of Operational Research**. Amsterdam, v. 80, p. 103-111, 1995.

WENG, M.X.; LU, J.; REN, H. Unrelated parallel machines scheduling with setup consideration and a total weighted completion time objective**. International Journal of Production Economics**, 70(3):215–226, 2001.

ZHANG, F.; TANG, G.C.; CHEN, Z.L. A 3/2-approximation algorithm for parallel machine scheduling with controllable processing times. **Operations Research Letters**, 29(1):41–47, 2001.

ZHU, Z.; HEADY, R. Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. **Computers & Industrial Engineering**, 38(2):297–305, 2000.

YAMASHITA, D.S.; ARMENTANO, V.A.; LAGUNA, M. Scatter Search for Project Scheduling with Resource Availability Cost. **European Journal of Operational Research**, 169, 623–637, 2006.