MARTÍN GÓMEZ RAVETTI

ALGORITMOS PARA O PROBLEMA DE SEQÜENCIAMENTO COM MÁQUINAS PARALELAS E TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

Belo Horizonte
Maio de 2007

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

ALGORITMOS PARA O PROBLEMA DE SEQÜENCIAMENTO COM MÁQUINAS PARALELAS E TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQÜÊNCIA

Tese apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

MARTÍN GÓMEZ RAVETTI

Belo Horizonte
Maio de 2007

Federal University of Minas Gerais
Instituto de Ciências Exatas
Graduate Program in Computer Science

ALGORITHMS FOR A SCHEDULING PROBLEM WITH PARALLEL MACHINES AND SEQUENCE DEPENDENT SETUPS

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

MARTÍN GÓMEZ RAVETTI

Belo Horizonte
May 2007

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Algoritmos para o problema de seqüenciamento com máquinas paralelas e tempos de preparação dependentes da seqüência

MARTÍN GÓMEZ RAVETTI

Tese defendida e aprovada pela banca examinadora constituída por:

Dr. Geraldo Robson Mateus – Orientador
Universidade Federal de Minas Gerais

Ph. D. Panos M. Pardalos – Co-orientador
University of Florida

Dr. Reinaldo Morabito Neto
Universidade Federal de São Carlos

Dr. Maurício Cardoso de Souza
Universidade Federal de Minas Gerais

Dr. Luiz Satoru Ochi
Universidade Federal Fluminense

Dr. Sebastián Alberto Urrutia
Universidade Federal de Minas Gerais

Belo Horizonte, Maio de 2007

Resumo

Problemas de seqüenciamento podem ser encontrados nas mais diversas áreas da ciência. Se considerarmos aplicações industriais, existe um grande número de problemas que podem ser modelados através de problemas de seqüenciamento, em especial aqueles relacionados a problemas de planejamento e programação da produção.

O planejamento da produção de uma empresa é usualmente realizado considerando dois ou três níveis. Cada nível com diferentes objetivos, mas com uma forte correlação entre eles. Na escola clássica, num primeiro estágio a empresa trabalha com produtos agregados ou famílias de produtos, onde o foco do planejamento está na otimização dos recursos utilizados e na capacidade das plantas. Os problemas de seqüenciamento são encontrados no nível imediato do planejamento da produção.

O objetivo desta tese é apresentar, discutir e resolver dois problemas de seqüenciamento, ambos os casos estão baseados num caso real de uma empresa Brasileira. O primeiro problema consiste num caso de máquinas paralelas considerando restrições realistas, como tempos de preparação de máquinas dependentes da seqüência e datas de entregas. Para resolver este problema, em forma exata, são propostos três modelos matemáticos e um algoritmo *Branch and Bound*. Um algoritmo baseado na técnica de relaxação Lagragiana é apresentado, seu principal objetivo é melhorar o limite inferior do problema e dessa forma melhorar o desempenho do algoritmo B&B. O segundo problema considerado é o problema de flow shop permutacional. Nesse caso, são propostos e testados dois algoritmos híbridos.

Resumo Estendido

Introdução

Os problemas de seqüenciamento podem ser encontrados nas mais diversas áreas da ciência, tanto em problemas médicos ou biológicos quanto em problemas de administração. Os problemas de seqüenciamento surgiram na literatura nos anos 50, e desde então muitos grupos de pesquisa com diferentes formações e objetivos vêm trabalhando nesta área. O grande interesse por este tipo de problemas revela por um lado sua importância para aplicações reais e, por outro, a complexidade e dificuldade do seu tratamento. O cenário industrial foi o escolhido para o desenvolvimento do nosso trabalho. Os problemas aqui tratados foram baseados em casos reais acontecidos numa empresa na cidade de Belo Horizonte. O primeiro problema considera um ambiente de máquinas paralelas e o segundo problema é o bem conhecido flow shop permutacional.

Trabalhos relacionados e métodos

Para resolver os problemas considerados neste trabalho, foram utilizados métodos heurísticos e exatos. Para resolver os problemas otimamente, utilizaram-se modelos de programação matemática junto com o solver CPLEX, assim como também foi desenvolvido um algoritmo do tipo *branch and bound*. Quando consideradas abordagens heurísticas, os algoritmos utilizados foram baseados nas metaheurísticas GRASP, VNS, Path Relinking e algoritmos Meméticos. Todos esses métodos serão posteriormente detalhados.

Problemas Reais

Como dito previamente, os problemas considerados foram baseados num caso real. A empresa Z é líder em seu mercado, possui mais de 3000 empregados e este trabalho foi desenvolvido em uma de suas plantas. Nesta planta em particular são produzidos cerca de 8000 tipos diferentes de produtos, todos eles seguindo a mesma seqüência produtiva. A seqüência de produção consiste em 5 etapas: dosificação, mistura, conformação (prensado), tratamento térmico e embalagem. Dentre estas etapas, a de conformação é a mais crítica, tanto do ponto de vista da qualidade do produto quanto do ponto de vista econômico.

Objetivos e contribuições

O principal objetivo deste trabalho é apresentar várias abordagens para dois grupos e problemas de seqüenciamento: o problema de máquinas paralelas e o problema de flow shop. Para modelar e resolver estes problemas utilizamos diferentes algoritmos baseados em métodos exatos e heurísticos, sempre tentando manter uma ligação com a aplicação real, discutindo vantagens e desvantagens dos métodos e de sua utilização no chão de fábrica.

No primeiro problema considerado, cada tarefa a ser seqüenciada possui uma data de entrega, e existe a necessidade de realizar uma preparação de máquina entre o processamento consecutivo de duas tarefas. O objetivo do problema é determinar a alocação e seqüência de produção de forma a minimizar a soma entre o tempo de conclusão de todas as tarefas e a soma dos atrasos ponderados. Para resolver o problema foram propostos três modelos, um algoritmo *branch and bound* e duas heurísticas. Como resultado da pesquisa realizada, resultaram três artigos em revistas internacionais (de Paula et al., 2007; Gómez Ravetti, M. et al., 2007; Rocha et al., 2007) e várias publicações em conferências nacionais e internacionais.

No caso do problema de flow shop, tratamos com o caso especial do problema permutacional, que foi resolvido com dois algoritmos híbridos. Como resultados da pesquisa realizada foram apresentados os resultados parciais do trabalho numa conferência em Julho de 2006, (Gómez Ravetti, M. et al., 2006).

Primeira Parte - Problema de Máquinas paralelas

Abordagens heurísticas

Neste capitulo é introduzido o primeiro problema e dois algoritmos heurísticos são propostos, implementados e testados. Os algoritmos estão baseados nas metaheuríticas GRASP e VNS.

Descrição do problema

O problema, basicamente, consiste na determinação da seqüência ótima de produção de um conjunto de tarefas. Cada tarefa pode ser processada por qualquer uma das máquinas da planta. As máquinas são não relacionadas, e isto implica que os tempos de processamento necessários para produzir uma tarefa podem variar entre as máquinas. Entre a produção de duas tarefas é necessário preparar adequadamente a máquina, e o tempo dessa preparação depende da seqüência das tarefas a serem produzidas e da máquina onde elas estão alocadas. Cada tarefa possui uma data de entrega, permitem-se atrasos, mas serão penalizados na função objetivo. O objetivo será a minimização do tempo de conclusão de todas as tarefas mais a soma dos seus atrasos ponderados.

Greedy Randomized Adaptive Search Procedure

A heurística desenvolvida neste trabalho foi baseada na estrutura do GRASP básico (Feo and Resende, 1995). GRASP é uma heurística *multi-start* para problemas combinatórios (Feo and Resende, 1995; Feo et al., 1996). Basicamente, a metaheurística consiste em duas fases: a construção de uma solução viável e uma busca local. Estas fases são repetidas em todas as iterações.

Várias versões da heurística foram implementadas e testadas, foram considerados diferentes algoritmos para a geração de soluções viáveis, assim como diferentes buscas locais. Com o objetivo de intensificar as buscas locais, foram testadas diferentes versões da heurística *path-relinking*.

O algoritmo heurístico mostrou-se muito eficiente, encontrando as soluções ótimas em instâncias pequenas e bons resultados para instâncias maiores. Também se mostrou muito flexível, já que conseguimos testar diferentes funções objetivo realizando poucas modificações. Também foi possível incorporar com certa facilidade restrições encontradas na aplicação real.

Variable Neighborhood Search

VNS é uma metaheurística proposta por (Hansen and Mladenovic, 1999), que sistematicamente cambina as estruturas de vizinhança na busca de boas soluções. VNS não segue uma trajetória específica, mas pesquisa vizinhanças progressivamente mais distantes da melhor solução encontrada até o momento. No nosso caso, diferentes buscas locais foram utilizadas, de forma a obter um algoritmo mais poderoso, especialmente para instâncias de grande porte. Uma completa análise do VNS pode ser encontrada em (Hansen and Mladenovic, 1999, 2003) e (Hansen et al., 2002).

Esta heurística apresentou um excelente desempenho, em especial para instâncias acima das 60 tarefas. A combinação das diferentes buscas locais permitiu não somente obter soluções de qualidade, mas também num tempo de execução baixo. Sendo uma excelente opção a ser utilizada na solução do caso real.

Modelos e o algoritmo de *branch and bound*

Neste capítulo dois modelos são comparados e testados, assim como um algoritmo *branch and bound*.

O primeiro modelo está baseado num grupo de restrições propostas por Manne (1960), para o problema de seqüenciamento do tipo job shop. Estas restrições são aquelas que permitem modelar corretamente a dependência da seqüência nos tempos de preparação de máquinas. Basicamente, são analisadas todas as combinações de seqüência de tarefas para cada uma das máquinas. Este tipo de modelo permite uma fácil leitura do problema, mas seu desempenho não é bom resolvendo problemas com até 15 tarefas. Os limites obtidos mediante a relaxação linear do modelo também são de baixa qualidade.

O segundo modelo está baseado numa series de restrições propostas por Wagner (1959). O modelo associa cada uma das tarefas a uma posição de processamento numa das máquinas, ao invés de considerar a posição relativa de cada par de tarefas, como no modelo anterior.

O *branch and bound* é essencialmente um algoritmo de enumeração em árvore, que de forma inteligente elimina soluções que garantidamente não são soluções ótimas. O algoritmo tem três procedimentos básicos, *inicialização*, *branching* e *bounding*. Durante a fase de inicialização, uma rápida heurística é utilizada para encontrar uma solução inicial, que servirá como limitante superior *LS*.

Branching divide o problema em subproblemas menores. Cada subproblema é representado por um nó da árvore. A estratégia de busca deve estar relacionada com o *branching* para decidir qual nó deverá ser selecionado para subdividir novamente o problema. O *LS* permite descartar os nós da árvore onde a solução representada obterá resultados melhores que a melhor solução obtida até o momento. Para determinar isto calcula-se um limite inferior *LI*. O procedimento *bounding* calcula o *LI* para cada nó para decidir qual deles pode ser eliminado e qual será escolhido para a próxima divisão.

Com relação ao desempenho dos algoritmos e modelos, foram resolvidas instâncias de até 25 tarefas. Foi demonstrado que o *B&B* possui um desempenho muito melhor que ambos os modelos resolvidos utilizando o solver CPLEX 9.0. Um aumento na variação dos tempos de preparação de máquinas parece incrementar o número de subproblemas ou divisões que o algoritmo precisa realizar, mas o incremento nas datas de entrega ou nos tempos de processamento produz o efeito contrário, diminuindo o número de subproblemas analisados e, consequentemente, o tempo de execução.

Limites Inferiores

Na literatura a minimização do makespan (tempo de conclusão das tarefas) e a soma dos tempos ponderados são problemas geralmente analisados separadamente. Neste trabalho são considerados como um único problema, devido a que é inspirado num problema real e a empresa requer a sua consideração em conjunto.

Neste capítulo, propomos um novo modelo onde, através da discretização do tempo e da metodologia conhecida como relaxação lagrangena, podemos melhorar os limites inferiores do problema. A relaxação apresentada resulta muito atrativa porque podemos decompor o problema em outro de emparelhamento que pode ser resolvido de forma eficiente. Vale advertir que somente foram testadas instâncias de pequeno porte e novos testes considerando instâncias maiores são necessários, já que o desempenho pode mudar consideravelmente.

Segunda Parte - Flow Shop

Flow shop permutacional

Nesta segunda parte do trabalho, consideramos o bem conhecido problema de flow shop permutacional. O problema encontrado na empresa pode ser representado como um caso especial de flow shop, motivo pelo qual inicialmente consideramos o problema permutacional.

O problema consiste em encontrar a seqüência de processamento ótima de forma a minimizar o tempo necessário para processar todas as tarefas. Neste problema, a seqüência produtiva consiste em M estágios, de forma que um produto é completado depois de passar por todos os estágios. Todos os produtos possuem a mesma seqüência produtiva e, ainda no caso permutacional, a seqüência de processamento das tarefas utilizada na primeira máquina deve ser mantida em todas as máquinas dos diferentes estágios.

Para resolver este problema foram propostas duas heurísticas híbridas, ambas heurísticas mantêm as características das metaheurísticas GRASP e Iterated local Search (ILS), ou seja, possuem uma fase de construção de solução viável e um estágio de busca local. Novamente, foi mantido um conjunto de boas soluções para que através da heurística *path-relinking* seja intensificado o poder das buscas locais.

As heurísticas possuem um comportamento de algoritmo Memético, utilizando o conjunto de boas soluções como população inicial. As implementações realizadas se diferenciam no momento no qual o algoritmo Memético é ativado.

Num caso, o algoritmo chama o Memético depois que, num certo número de iterações, a melhor solução não é melhorada. Desta forma espera-se que o conjunto de soluções possa ser melhorado ou evolua de forma que a heurística possa investigar novas regiões do espaço de soluções.

No segundo caso, o algoritmo chama o Memético somente no final da busca, assim, a heurística tem como objetivo básico alimentar o algoritmo Memético com uma população diversificada e de "boa qualidade".

A segunda versão mostrou um desempenho ligeiramente melhor. Numa etapa posterior da pesquisa serão incorporados estágios com máquinas paralelas e as restrições relativas aos tempos de preparação de máquinas.

Conclusões e trabalhos futuros

Na primeira parte da tese, a pesquisa foca-se num problema de seqüenciamento de tarefas com restrições que não são comumente encontradas na literatura. O problema considera tempos de preparação de máquinas dependentes da seqüência, datas de entrega e máquinas não relacionadas. O objetivo é encontrar a seqüência que minimize a soma do makespan e a soma dos atrasos ponderados.

No capitulo 2, duas heurísticas foram propostas e testadas. A primeira foi baseada no GRASP. Esta abordagem provou ser muito flexível já que resulta relativamente simples modificar o algoritmo para outros propósitos e situações. Outra vantagem é que possui poucos parâmetros, os quais podem ser facilmente ajustados. Neste caso particular, o tempo gasto pelo algoritmo é muito razoável se consideradas as restrições consideradas.

O segundo algoritmo foi baseado na metaheurística VNS. Foi realizada uma serie de experimentos comparando o VNS com três diferentes versões do GRASP, cada uma com diferentes buscas locais. Analisando os resultados computacionais, foi observado que VNS obteve bons resultados médios para instâncias com 60 tarefas ou mais, especialmente para aquelas com datas de entrega apertadas. A combinação de três buscas locais provou ser muito eficiente, de simples implementação e sem a necessidade de ajustar muitos parâmetros.

No capítulo 3, um algoritmo *B&B* utilizando GRASP como heurística de inicialização, e dois modelos de programação inteira e mista foram propostos e testados. Também como contribuição podemos citar a geração de um conjunto de instâncias com diferentes valores de datas de entrega, tempos de preparação de máquinas e tempos de processamento.

Foi provado que o algoritmo *B&B* possui um desempenho muito superior que os modelos de MIP resolvidos com o CPLEX 9.0. Um aumento na variação no tempo de preparação de máquinas incrementa o número de nós que o algoritmo deve pesquisar, mas o incremento na variação das datas de entrega ou nos tempos de processamento parece favorecer o algoritmo reduzindo o número de nós pesquisados.

No capítulo 4, é introduzida a formulação que utiliza o tempo em forma discreta e propõe-se um algoritmo baseado na relaxação Lagrangeana do modelo para melhorar os limitantes inferiores do problema. A relaxação do modelo parece ser prometedora porque foi possível decompor o problema original num problema de emparelhamento que pode ser resolvido de forma eficiente. Mesmo assim, é importante destacar que somente foram testados problemas de pequeno porte e que resulta necessário testar problemas maiores já que o desempenho do algoritmo poderia variar.

Na segunda parte da tese trabalhamos com o problema do tipo flow shop. Foram propostos dois algoritmos híbridos baseados no GRASP, ILS e em algoritmos Meméticos para o caso permutacional. Numa primeira abordagem utilizamos um algoritmo do tipo GRASP-ILS que, mantendo um conjunto de boas e diversificadas soluções, permitisse o algoritmo Memético começar com uma população de boa qualidade. Na segunda abordagem, o algoritmo Memético é chamado cada vez que o algoritmo GRASP-ILS atinge um estágio de estagnação por certo número de iterações.

O melhor desempenho foi obtido pela primeira abordagem, isto é, utilizando inicialmente o algoritmo GRASP-ILS para depois chamar o algoritmo Memético. Os resultados são bem competitivos já que nas instâncias mais difíceis do conjunto proposto por Taillard, 100 tarefas e 20 estágios, o algoritmo pode encontrar soluções a 2.5% da melhor solução conhecida em

menos de 80 segundos em 80% dos casos.

Trabalhos futuros

Considerando o problema de máquinas paralelas, atualmente estamos trabalhando em três diferentes projetos. O primeiro considera a utilização de um algoritmo do tipo *Local Branching* que será comparado com os modelos propostos e com o algoritmo *B&B*.

Em muitos cenários, especialmente no planejamento da produção, problemas de grande porte devem ser resolvidos. Por este motivo, o segundo projeto analisa a implementação de uma heurística que considera uma arquitetura distribuída.

O terceiro projeto contempla a integração de dois níveis do planejamento da produção, propondo um esquema iterativo para gerar um plano de produção que considere restrições do seqüenciamento.

Considerando o ambiente flow shop, estamos trabalhando no problema conhecido como flow shop flexível. Neste caso, cada estágio pode estar composto por mais de uma máquina, onde cada uma pode realizar qualquer uma das tarefas. Para este problema, estamos trabalhando em limites inferiores, modelos e heurísticas.

Abstract

Scheduling problems can be found in the most diverse areas of science. If we consider industrial applications, there are a big number of problems that can be modeled as a scheduling problem, especially those related to production planning.

The production planning of a company is usually done considering two or three planning levels. Pursuing different objectives, but with a strong correlation between them. In the classic school, in a first stage the company works with families of products where the main focus is the optimization of resources and the capacity of the plants. The scheduling problems are found in the level of the immediate production planning.

The aim of this dissertation is to present, discuss and solved two scheduling problems, both cases are based on a real problem from a Brazilian company. The first problem consists in a parallel machine case in which realistic constraints, as sequence dependent setups and due dates, are considered. To solve this problem, three mathematical models and a branch and bound algorithm are proposed to find the optimal solution. Two heuristics, based on the metaheuristics GRASP and VNS, are also implemented and tested. An algorithm based on a Lagrangian relaxation is also proposed, the main objective of this algorithm is to improve the problem's lower bounds and consequently the performance of the branch and bound algorithm. The second problem is the well known permutation flow shop problem. In this case, two hybrid algorithms are proposed and tested.

To Julia and Laura

Acknowledgments

There are several people who deserve my sincere acknowledgments. First, I would like to thank my advisor, Professor Geraldo Robson Mateus, for his inestimable support, knowledge, guidance and encouragement. Without his common-sense this dissertation would be an extremely difficult task.

I have the great opportunity of studying at the University of Florida, under the supervision of Dr. Pardalos. His commitment, dedication and willingness were inspiring and provide me many opportunities. I want also to thank Dr. Mauricio G. C. Resende, with whom I have the privilege to work with. His friendship, advises and comments were priceless.

I am very grateful with my committee members Professor Reinaldo Morabito Neto, Professor Luiz Satoru Ochi, Professor Mauricio C. de Souza and Professor Sebastián A. Urrutia, for the time spent in the evaluation of this work and valuable feedback.

I have the great pleasure and honor to work with Fabiola G. Nakamura, Pedro Leite Rocha, Mateus Rocha de Paula, Frederico P. Quintão, Claudio N. Meneses, Michael J. Hirsch, Taís Valeriano, Marcos Magalhães, Adriana A. de Oliveira and Luciana Assis.

All my colleagues of the Computer Science Department at the Federal University of Minas Gerais, and colleagues of the Laboratory of Operational Research (LaPO): David(s), André(s), Raquel, Pedro, Luciana, Taís, Marcos, Braulio, Flávio, Gleice, Fernanda, Gurvan, Cristiana, Naka, Mauricio, Vilar, Ruiter, Fabiano, Luiz, Wagner, João, Ricardo, and many others.

Thanks go to all the faculty and staff of the Computer Science Department at the Federal University of Minas Gerais. Their support and help allow me to focus on my work and have a good time during my years at BH.

I want to give big thank to my family and friends, for their trust and encouragement. And, finally, I owe my eternal gratitude to my hidden co-author and advisor, Laura. Her love and support make this dissertation worthy and possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Scheduling problems can be found in the most diverse areas of science, from biological and medicine to administration problems. In computer science, different scheduling problems are found in areas like computer architecture, information retrieval and computer networks. If we consider industrial applications, there are a big number of problems that can be modeled as a scheduling problem, especially those related to production planning.

Scheduling problems first appeared in the literature in the 1950's. Several research groups with different backgrounds and different approaches have been working in this area. However there is still space for new research, indicating not only the complexity and difficulty of these problems, but also their importance for real applications. It is possible to find journals dedicated only to these problems.

This research was initially based on a real case study of a company located in Belo Horizonte. From the analysis of the real case, two problems were successfully considered and solved by using different approaches (Gómez Ravetti, M., 2003). The scheduling problem family is impressive, and for that we focus in two cases found in this real case: the first one considers a parallel machine environment and the second one a flow shop environment.

1.1 Scheduling Problems

Scheduling problems consist in allocating limited resources to a set of activities that have to be done, in such a way that certain criteria are optimized.

If scheduling problems are considered in a general way, it is easy to visualize them in almost every real-world situation. However, when dealing with real cases, their own specificity make them unique, and this specificity should be taken into account for modeling and solving them (Błażewicz et al., 1996).

1.1.1 Parallel machines

In this environment we have a set of independent tasks or jobs to be scheduled on parallel machines. The machine types, the criterion to be optimized and different features considered during the optimization process define the complexity of the problem. As well as, allow us to make a classification of different problems.

The machines can be identical, uniform or unrelated:

Identical machines The processing times of each job or task are independent of the machine which processes it. The most common notation to this environment is P.

Uniform machines Each machine has a different speed, and the processing time needed to get a job done remains proportional for all the jobs. The notation is U.

Unrelated machines There is no particular relation between the machines. The processing time for each job has no relation to the processing time of the same job in any other machine. The notation used in this case is R.

Mokotoff (2001) presents a survey on parallel machines, with an emphasis on identical machines. Other interesting surveys are (Cheng and Sin, 1990) and (Lawler et al., 1993). In Chapters 2, 3 and 4, we deal with this environment, considering realistic constraints.

1.1.2 Flow shop

This is a specific case in which dedicated machines are considered, that is, they are specialized for the execution of certain tasks. A flow shop consists of a set of different dedicated machines that performs tasks of jobs. It is assumed that each job needs the same set of tasks to be done, and each machine is dedicated to one of these tasks, also called stages.

The most studied case in the flow shop scenery is the Permutation Flow Shop Problem (PSFP). This is the case in which each machine processes the jobs in the same order. Once the production sequence is set, it will remain the same for all machines (stages), this problem is known to be NP-Hard when minimizing the conclusion time and the number of machines is greater than 3, see (Coffman, 1976; Kan, 1976). In our dissertation we tackle this problem in Chapter 5.

1.1.3 Problem definition and classification

There is a vast number of features and variations that can be considered in scheduling problems and they have an important role in determining the complexity of the problems.

In general, the activities or tasks can be characterized by the following data:

- Processing times, p_{im}, time needed by machine m to process task i.

- Arrival time, r_i, time at which task i is ready for processing.
- Due date, d_i, time at which task i should be completed.
- Deadline, $\tilde{d}_i$, this is a "hard" deadline; the task i must be completed before $\tilde{d}_i$.
- weight, ν_i, degree of importance or priority of task i.
- Resources needed for each task.

In this work it is assumed that the number o jobs are finite and all the mentioned data are integers, and whenever the schedule starts all the resources or setups are ready to the first job. Preemption is not allowed, that is, once the machine begins to process any task, it must finished before processing another task.

Another important definition is related to the criterion chosen to evaluate the schedules. The most common criteria are:

Completion time or makespan C_{max} or Z, time needed to complete all the jobs. It is the maximum completion time C_j for all tasks j.

Mean Flow time $\bar{F} = \sum_{j=1}^{n} F_j$, where $F_j = C_j - r_j$, F_j represents the difference between the completion time of a task j and the date when the task is ready to be processed.

Maximum lateness $L_{max} = max\{L_j\}$, where $L_j = C_j - d_j$ represents the delay of task j.

There are other related criteria that can be used, like minimizing the earliness or tardiness, the number of weighted delays or the number of delayed jobs. For more details on definition and other criteria and features, see (Baker, 1974), (Błażewicz et al., 1996), (Pinedo, 1995) or (Brucker, 2004).

1.2 Related works and methods

In this section we present some works related to the research presented in this dissertation. As the particular problems are introduced and defined, specialized references will be incorporated and analyzed for each case.

The problems analyzed in this work were based on a real case and to solve them we used heuristic approaches and exact methods. For solving the problems optimally we use mixed integer lineal programming models with the CPLEX 9.0 [1] solver and a proposed Branch-and-Bound algorithm. When considering heuristics approaches we use heuristics based on GRASP, VNS, ILS, and hybrid approaches using Memetics Algorithms, Path Relinking and polynomial time heuristics as NEH. Below we introduce some of the methods used in this dissertation.

[1] *CPLEX Software: an optimizer for linear problems developed by ILOG. More information at their website: http://www.ilog.com/products/cplex/*

1.2.1 Branch and Bound (B&B)

Branch and Bound is a general algorithmic method that basically enumerates all feasible solutions of various optimization problems and it is especially used with discrete and combinatorial problems. Its name implies in two main procedures: Branching is the procedure of dividing into two or more subproblems mutually excluded. Bounding calculates lower bounds on the optimal solution for each subproblem generated in the branching procedure. The algorithm can be represented as a enumeration tree, then it is possible to set as the root node the original problem. Each subproblem obtained by the branching procedure will be a tree node in the next level. Assume that in some stage of a branch and bound algorithm, a solution x' has been obtained for a minimization problem. And, let's also consider that the bounding procedure obtain a lower bound lb for one node of the tree. If $lb \geq f(x')$ then the node do not need to be considered any further because the node will never led the algorithm to the optimal solution x^*. For this reason, the branching procedure is not executed from this node. Different families of algorithms are based on how the bounding and branching strategies are designed.

1.2.2 Greedy Randomized Adaptive Search Procedure (GRASP)

Greedy Randomized Adaptive Search Procedure, GRASP is a multi-start metaheuristic for combinatorial problems (Feo and Resende, 1995; Feo et al., 1996). This metaheuristic consists of two phases: the construction of a feasible solution in a greedy-random way and a local search, to improve the constructed solution. These phases are repeatedly applied, that means, the metaheuristic uses local search in different starting solutions.

The construction phase is based on a greedy algorithm but with an association to a random procedure. This combination allows the method to combine a greedy algorithm and a random procedure, to produce starting solutions for local search.

One of the main drawbacks of this method is that the iterations are memoryless. Each iteration has no relation with the next one. Thus, the use a memory-based mechanism can improve its efficiency. Several mechanisms are discussed and analyzed in (Resende and Ribeiro, 2003a). Two mechanisms are used in this work, the first being Path Relinking, and the second being a hybrid method based on a Memetic Algorithm.

1.2.2.1 Path Relinking

The Path Relinking (PR) technique is used as search strategy in many heuristics (Glover and Laguna, 1997), (Glover, 2000). In 1999 it was used in a GRASP environment by Laguna and Martí (1999). Resende and Ribeiro (2003b) present and analyze two strategies:

- PR as a tool to improve the solution found by GRASP.
- PR as a strategy to intensify the local search procedure.

The algorithm selects two solutions and analyzes a path of solutions found between both. This path is built around some number of job movements. For each movement, the encountered solution is analyzed.

In this work, we use path relinking to intensify the local search. Even so, there are several ways to implement this technique, Resende and Ribeiro (2003b) mention a few:

- Periodic use of path relinking.
- Use of path relinking in the two possible directions, that is, from solution x_1 to solution x_2 and from x_2 to x_1.
- Use of path relinking in only one direction.
- Use of truncated path relinking. That is, by analyzing only part of the path.

We also refer to (Feo et al., 1991, 1996), where we find applications of GRASP for one-machine scheduling problems, (Binato et al., 2002) which present an interesting application of the metaheuristic for job shop problems, (Festa and Resende, 2002) where a bibliography of GRASP is presented and (Resende and Ribeiro, 2003a) where we find an interesting and useful analysis of the applications of PR in GRASP

1.2.3 Variable Neighborhood Search (VNS)

Variable Neighborhood Search (VNS) is a metaheuristic proposed in (Hansen and Mladenovic, 1999), which systematically changes neighborhood structures. VNS does not follow a trajectory but investigates progressively more distant neighborhoods of the current incumbent solution. In our case, we also use different local searches to obtain an even more powerful algorithm.

A very complete study about VNS can be found on (Hansen and Mladenovic, 1999, 2003) and (Hansen et al., 2002). This metaheuristic is successfully applied to solve various problems such as the *p-median* problem (García-López et al., 2002), the multi-depot routing problem (Polacek et al., 2004) and several other classical problems (Hansen and Mladenovic, 1999), (Hansen and Mladenovic, 2003).

1.3 Contextualization

Nowadays it is difficult to find companies that deal with these problems efficiently, that means, finding good solutions in adequate time. This is one of the main purposes for our research. For that, we tried to maintain real characteristics, in modeling and specification level, as well as in the trade off between the quality of solution and processing time.

We choose an industrial scenario for our problems, more specifically, a production planning situation. In most cases, the production planning is done in different levels and pursuing different objectives, but with a strong correlation between them. In the classic school, in a first stage the company works with families of products where the main focus is the optimization of resources and the capacity of the plants.

The scheduling problems are found in the level of the immediate production planning. That is, those responsible for these tasks usually receive the production orders or tasks that have to be processed before a certain date. It is usual that at this stage the main concern when solving these problems are: *(i)* to finish all the tasks as soon as possible, *(ii)* to respect the due dates, *(iii)* minimize delayed jobs, and other similar criteria. It is easy to see that many of these objectives can be conflicting with each other, hence there is a need of looking for flexible methods and with an easy understanding. The production planning of a company is far from being a simple and exact process; even the company's policies have a decisive roll in this process. In next section we introduce the problem from which we based our work.

1.3.1 Real case

The problems studied in this work were based on a real case, from a Brazilian company, that we will refer as company Z. Company Z is a leader in its market and has more than 3000 employees, our work was based in one of its many plants.

In this particular plant, it is produced more than 8000 products, all them with the same production sequence that consist in 5 stages: material input measurement, blending, conformation, heat treatment and packaging, see Figure 1.1. In each of these stages the company has more than one machine and most of the products can be processed by any machine at a particular stage.

The main production stage is the conformation stage, and this is because the quality of the product is extremely related to the quality of its conformation and because this is the most difficult process in this production sequence. For all these reasons, company Z has been investing a lot in this particular stage. Nowadays it has 7 machines, 4 of them are identical but they need manual work, the others are unrelated and more advanced.

The production planning of this plant is made by planning their sales in the conformation stage and once the production plan is set, all other stages are adjust. Capacity problems in other stages are not common but when happen they are easily solved. The original case is a flexible flow shop.

1.3.1.1 Assumptions and simplifications

For our work we are going to make some simplifications and assumptions to define two specific problems.

The first problem considers a parallel machine environment. This case represents the problem of scheduling all the jobs considering only the *conformation* stage. We assume that

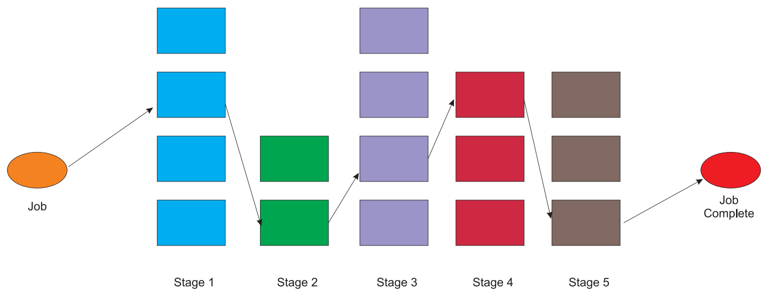


Figure 1.1: Real case exemplification.

all the machines are unrelated and they can process all the jobs. The processing times and setup times values are integers, due dates are considered and delays are allowed.

For the second problem, we suppose a flow shop environment, that is, one machine per stage is considered and all the jobs must pass for each stage in the same order to be completed. No setup times are considered neither due dates.

For both cases, preemptions are not allowed. A more detail description will be given forward.

1.4 Objectives and contributions

The main objective of this dissertation is to present several approaches for two groups of problems, the parallel machine problem and the flow shop problem. To model and solve these problems, we use several algorithms based on exact methods and heuristics, always striving to remain linked to real applications, discussing advantages and disadvantages of the different models and methods.

In the first problem, the parallel machine case, we are considering that each job has a due date, and there is a need of a setup between the process of two different jobs in the same machine. The objective is to minimize the completion time plus the sum of the weighted delays. To solve this problem we analyze three mathematical models, propose a branch and bound algorithm and two heuristics. As a result of our research three journal papers were published (de Paula et al., 2007; Gómez Ravetti, M. et al., 2007; Rocha et al., 2007) and 6 conferences papers were presented.

In the flow shop case, we deal with the permutation problem, and we solved using two hybrid algorithms. As the result of our research in flow shop problems we present our work at a conference in July of 2006, (Gómez Ravetti, M. et al., 2006).

1.5 Organization of the text

The dissertation is organized in three parts. In the first part, we deal with the parallel machine scenario.

In Chapter 2 two heuristics are proposed and analyzed. The algorithms are based in two well known metaheuristics, GRASP and VNS. In Chapter 3 we formally introduce two mathematical models and compare the solution obtained using a commercial software with a proposed Branch-and-Bound algorithm. A time-indexed model and a lagrangian relaxation is analyzed in Chapter 4.

In the second part, the flow shop environment is considered. In Chapter 5, we deal with the permutation case, considering two hybrid algorithms.

Finally in the third Part we conclude our work, analyzing results and future works.

Part I

Parallel Machines

Chapter 2

Heuristic Approaches

2.1 Introduction

In the parallel machine scenery several topics are discussed, from modeling techniques to different heuristics. In this chapter we introduced the problem and present two heuristic approaches based on the metaheuristics GRASP and VNS.

The literature on parallel machine scheduling problems (PMSP) is very extensive and this topic is studied from many points of view. Nevertheless, the consideration of an unrelated environment is not usual, especially when compared to the identical parallel machines or the single machine scenarios.

The unrelated PMSP (UPMSP) is very common in industry. In particular, the problem addressed in this work was based on a real case of a Brazilian company, where the equipment is composed of machines with different technologies. That fact implies not only different processing times per machine but also different setup times per machine.

Garey and Johnson (1979) show that minimizing the makespan considering two identical machines is a NP-hard problem. Certainly, a problem with unrelated machines and job sequence dependent setups is also NP-hard. Due to its complexity and to the necessity of almost constant reschedules, the use of exact algorithms is prohibited for a large number of jobs.

Interesting surveys on parallel machines can be found in (Lawler et al., 1993), (Cheng and Sin, 1990) and (Mokotoff, 2001). Cheng and Q. Ding (2004) also present a very useful survey on scheduling problems considering setup times.

There are few works considering UPMSP. Adamopolos and Pappis (1998) propose a polynomial heuristic to deal with a scheduling problem with common due dates. Jansen and Porkolab (2001) propose an approximation scheme for an UPMSP considering sequence dependent setup times and preemptive and non-preemptive jobs. Kim et al. (2002) use Simulated Annealing to address an unrelated PMSP, minimizing the total tardiness considering sequence-dependent setup times. Kim et al. (2003) consider a similar scenario with common due dates where four heuristics are proposed and studied.

An exact solution to an UPMSP is attempted in (Pereira-Lopes and de Carvalho, 2006). In this work the authors develop a branch-and-price approach to the problem of scheduling a set of independent jobs, with release dates and due dates, on unrelated parallel machines with availability dates and sequence-dependent setup times, to minimize the total weighted tardiness. We also refer to Baker (1974), Błażewicz et al. (1996), Pinedo (1995), Lee and Pinedo (2002) and Brucker (2004) for classic and well known approaches.

To solve our problem, we first use two heuristic approaches based on two well-known metaheuristics, GRASP and VNS. The metaheuristic GRASP (Greedy Randomized Adaptive Search Procedure) proposed by Feo and Resende (1995) is a multi-start or iterative procedure where each GRASP iteration consists of a construction phase, where a feasible solution is constructed, followed by a local search procedure that finds a locally optimal solution. We also refer to Feo et al. (1991, 1996), where we find applications of GRASP for one-machine scheduling problems, Binato et al. (2002) which present an interesting application of the metaheuristic for job shop problems, Festa and Resende (2002) where a bibliography of GRASP is presented and Resende and Ribeiro (2003a) where we find an interesting and useful analysis of the applications of path relinking in GRASP.

VNS is a modern metaheuristic that proposes systematic changes of the neighborhood structure within a search to solve optimization problems. A very complete study about it can be found in Hansen and Mladenovic's work (Hansen and Mladenovic, 1999, 2003) and (Hansen et al., 2002). This met-heuristic was successfully applied to solve various problems such as the *p-median* problem (García-López et al., 2002), the multi-depot routing problem (Polacek et al., 2004) and several other classical problems, readers can check (Hansen and Mladenovic, 1999, 2003) and (Hansen et al., 2002).

The aim of proposing a VNS algorithm is to solve large instances of the scheduling problem with identical or unrelated parallel machines and both machine and sequence-dependent setup times minimizing their makespan plus the weighted delays. The neighborhood structures are based on known relevant local searches.

The VNS performance depends on the initial solution, between other aspects, that is sequentially improved. In order to generate a fairly good initial solution we use an algorithm based on that proposed by Nawaz et al. (1983) that has shown to be one of the best polynomial heuristic for the permutation flow shop problem (NEH).

The lack of other works dealing with this problem, force us to use artificial instances to test our algorithms. All our instances and results are available at the *UFMG Scheduling Group* home page [1].

This chapter is organized as follows: Section 2.2 introduces the problem and discuses some decisions taken. In Section 2.3 the heuristic based on GRASP is introduced with some important features of our version; this section also includes some computational results. In

[1] http://www.dcc.ufmg.br/lapo/wiki/index.php/Scheduling_Team

Section 2.4 we discuss our algorithm based on VNS, introducing neighborhood structures, local searches, and the use of a polynomial heuristic based on NEH to obtained the first solution to the VNS method. Section 2.5 presents several tests comparing both approaches. Finally, Section 2.6 concludes the chapter analyzing the results and presenting incoming works in this research line.

2.2 Problem description

In this part of the dissertation we consider a problem with realistic constraints. The problem consists in an unrelated parallel machine case, in which each job has a processing time that also depends on the machine in which it is going to be processed. Between processing two different jobs, a machine needs a setup; this setup depends on both the job sequence and the machine. We also take into account that each job has a due date.

Before presenting the algorithms it is interesting to discuss some important decisions taken: First, it was decided to allow delays. When solving problems with tight due dates or with a large number of jobs, if delays are not allowed we can have no feasible solutions, loosing good possible schedules. Even in a not so extreme environment, the possibility of allowing delays can be particularly useful in the managerial decision making.

The second point to be discussed is about the objective function. As it is well known in scheduling theory, there are several criteria to use with different purposes. From a commercial point of view, one of the most important objectives in any company is do not have delays with the clients. On the other hand, from the production point of view it is interesting to finish all the jobs as soon as possible trying to use all machines in the best possible way. It is easy to see that real world scheduling problems have a well defined multi-objective nature.

Finally, we have to analyze how these different objectives are combined. This combination is not a simple task that satisfies all the decision makers. The combination of two objective functions was firstly require from the company. For all that, we decided to minimize the sum of the makespan, that is, the completion time of all jobs, plus the sum of weighted delays. These weights can be understood as a measure of the importance of the job, and this importance can be related to the client, a certain process or even a managerial decision.

The addition of the makespan plus the weighted delays is a sum of times, resulting in a coherent objective function. In (Mosheiov, 2004) we find a similar objective function but with a multi-criteria approach. Using the notation $\alpha|\beta|\gamma$ proposed by Graham et al. (1979) and Błażewicz et al. (1983), this scheduling problem can be formally defined as $R|s_{ijm}|C_{max} + \sum T_i \cdot v_i$, see (Pinedo, 1995).

2.3 Greedy Randomized Adaptive Search Procedure

The heuristic used in this dissertation has a similar structure to basic GRASP (Feo and Resende, 1995), which is a multi-start metaheuristic for combinatorial problems (Feo and Resende, 1995; Feo et al., 1996). Basically, this metaheuristic consists of two phases: the construction of a feasible solution and a local search. These two phases are repeated in every iteration. Figure 2.1 presents the pseudo-code of GRASP.

Algorithm 2.1: Pseudo-code of GRASP's main block.

```
Procedure: Rand_Construction(Job_List)
begin
    Read_Input();
    $Job\_List \leftarrow$ Sort_Jobs($rule$);
    for $i \leftarrow 1$ to iterations do
        $Local\_Solution \leftarrow$Rand_Construction($Sort\_List$);
        $Local\_Solution \leftarrow$ Local_Search($Local\_Solution$);
        if $Local\_Solution < Best\_Solution$ then
            $Best\_Solution \leftarrow Local\_Solution$;
        end
    end
end
```

In basic GRASP, for building a feasible solution, a list of candidates is compiled for random selection of the next element to be added to the solution. This list is made based on a greedy function.

However, in our implementation, the candidate's list is made only once at the beginning of the procedure. In each iteration, the list is randomized and the algorithm builds the solutions using a greedy function. The use of a greedy function to schedule the jobs, and not to build the list of candidates, proved to be remarkable. This procedure simplifies the implementation, providing good results.

2.3.1 Sorting rules

Two rules for sorting the job list are tested. The first one uses the due date as sorting criterion [2]. In this case, the jobs are ordered in a non-decreasing order. The second rule uses a tardiness lower bound (LB), suggested by Armentano and Ronconi (1999) for a Tabu-Search algorithm. In this case, the jobs are ordered in a non-increasing order. For this problem the LB is calculated as follows:

$$LB[i] = d_i - \max_m \{p_{im}\} \tag{2.1}$$

[2]This criterion is also known as Earliest Due Date (EDD)

It is important to realize that previous job-allocations do not affect the sorting criteria, this is the main motive for avoiding the use of a candidate's list. Many other sorting criteria were test but with poor results.

2.3.2 Construction phase

Two different types of construction phase are considered. In the first type, after sorting the candidate list, a greedy function schedules each job to the machine that finishes it first. In the second type, we select two machines as candidate machines and with a probability of 70% for the best candidate and 30% for the other, we randomly choose one.

Figure 2.2 shows the basic pseudo-code of the construction phase function.

Algorithm 2.2: Pseudo-code of GRASP's construction phase.

```
Procedure: Rand_Construction(Job_list)
begin
    $Solution \leftarrow \emptyset$;
    $Rnd\_list \leftarrow$ Rnd_Sort($Job\_list$);
    while $Local\_Solution$ is not complete do
        select the first element ($i$) from $Rnd\_list$;
        $Local\_Solution \leftarrow Local\_Solution \cup \{i\}$;
    end
    return $Local\_Solution$;
end
```

2.3.3 Probability distribution and local search

Once the job list is ordered, the algorithm will randomly reorder the array using a specific probability distribution. In this chapter, we present several tests with two types of probability distributions.

The first probability function is $F1(i) = 1/pos(i)$, where $pos(i)$ indicates the position of the job i on the sorted vector. The second function is created using a linear equation considering the first and last job of the sorted vector $F2(i) = -(pos(i)/n) + 1$. The second function allows the heuristic to exploit a bigger set of feasible solutions.

The local search swaps all the existing pairs of jobs. This local search is chosen arbitrarily. The algorithm used for the local search has a time complexity $O(n^2)$. Later in this chapter, the heuristic is tested with other local searches.

2.3.4 Path Relinking

The path relinking (PR) technique is used as search strategies in many heuristics (Glover and Laguna, 1997; Glover, 2000). In 1999 it was used in a GRASP environment by Laguna and Martí (1999). In (Resende and Ribeiro, 2003b) two strategies are presented:

- PR as a tool to improve the solution found by GRASP.
- PR as a strategy to intensify the local search procedure.

The algorithm selects two solutions and analyzes the path of solutions found between both. This path is built around some number of job movements. For each movement, the encountered solution is analyzed.

In this work, we use path relinking to intensify the local search. Even so, there are several ways to implement this technique. Resende and Ribeiro (2003b) mention a few:

- Periodic use of path relinking.
- Use of path relinking in the two possible directions, that is, from solution x_1 to solution x_2 and from x_2 to x_1.
- Use of path relinking in only one direction.
- Use of truncated path relinking. That is, by analyzing only part of the path.

The algorithm of PR used in this dissertatio works as follows. First, a pool of good solutions is retained. Every time path relinking is used, a solution is randomly chosen from this pool. Then, the path of solutions is analyzed in one direction, from the found solution in local search to the selected solution from the pool. Finally, the whole path between the solutions is considered.

2.3.5 Versions of GRASP

This section presents the different versions of the heuristic tested. The results of these experiments are presented in section 2.3.6.

The versions vary in the following aspects:

1. sorting rule (SR);
2. frequency of path relinking use (FPR);
3. probability distribution (PD);
4. construction phase(CP);

Table 2.1 details the parameters utilized for each of the heuristics considered. As an example, the version "GPSF" of the table, indicates a GRASP algorithm with a FPR equals 1, due date as sorting criterion, $F2$ as the probability function and a deterministic construction phase.

Table 2.1: Configurations of the heuristic for the experiments. The column SR shows the sorting rule chosen, FPR indicates the number of iterations between the use of PR, PD indicates the probability distribution $F1 = 1/pos(i)$ or $F2(i) = -(pos(i)/n) + 1$ and finally the column CP indicates if is used a deterministic or a random approach.

Name	SR	FPR	PD	CP
GP	due date	1	$F1$	deterministic
GPL	LB	1	$F1$	deterministic
GPS	due date	2000	$F1$	deterministic
GPF	due date	1	$F2$	deterministic
GPC	due date	1	$F1$	random
GPSL	LB	2000	$F1$	deterministic
GPLF	LB	1	$F2$	deterministic
GPLC	LB	1	$F1$	random
GPSF	due date	2000	$F2$	deterministic
GPSC	due date	2000	$F1$	random
GPFC	due date	1	$F2$	random
GPSLF	LB	2000	$F2$	deterministic
GPSLC	LB	2000	$F1$	random
GPLFC	LB	1	$F2$	random
GPSFC	LB	2000	$F2$	random
GPSLFC	LB	2000	$F2$	random

2.3.6 GRASP experiments

The heuristics are implemented in C, using the gcc compiler. The experiments are executed on a Pentium IV computer, 2.4GHz CPU and 1GB of RAM memory under a Debian GNU/Linux environment.

Different instances with a variety of sizes are considered. Instances for this kind of problem are not found so we have generated them randomly. Ten replications are considered for each size, N = 20, 60, 100, 150.

This problem was originally based on a real-world problem. Therefore, the configuration tested is composed by four identical machines and two unrelated machines (411).

2.3.6.1 Instances generation

Random instances are generated for these tests, based on the following rules:

- The processing times of the group of identical machines vary between 10 and 20 time units;
- The processing times of the unrelated machines vary between 5 and 8 time units;
- The setup times vary between 1 and 7 time units.

The generated instances can also be used for different machine configurations. The algorithm to generate these instances and the instances used in this dissertation are available on line at the *UFMG Scheduling Group* home page[3].

2.3.6.2 Tests planning

The tests are divided in five parts. In the first part, the seed and the number of iterations are fixed. The objective of this part is to find the best configuration of the heuristic parameters to this kind of problem. For these tests a configuration of four identical machines and two unrelated is maintained.

In the second part only instances with 150 jobs are considered. The seed is fixed but not the number of iterations. In the third part only one instance of the same size is considered. The number of iterations is fixed and the heuristic is executed ten times with different seeds.

With the second and third tests, we intend to analyze the relationship between the heuristic performance with the number of iterations, and its seed dependence.

In the fourth part of the tests, we analyze the influence of the use of PR. Finally, in the fifth part the goal is to analyze the heuristic behavior for different machine configurations.

2.3.6.3 Lower bounds

For each instance a lower bound is obtained, using three simple methods. These lower bounds will be properly presented in the next chapter.

The lower bound is used to obtain the average percentage deviation (APD) (Oğuz et al., 2003), where, the percentage deviation is defined as follows,

$$PD = \frac{X - LB}{LB} \tag{2.2}$$

where X denotes the value of the objective function obtained by the heuristic.

The percentage deviation is a measure of how far the solution is from the lower bound. Then the solution is better for small values of APD.

2.3.6.4 Results

This section presents the computational results of several tests with the different versions of the heuristic. Five different subsections are considered.

2.3.6.5 First part.

In this first part of the results the focus is on the adjustment of the parameters. The goal is to detect the best configuration for this kind of problem. The seed and the number of iterations

[3]http://www.dcc.ufmg.br/lapo/wiki/index.php/Scheduling_Team

are fixed.

The arbitrary chosen seed for these tests is 1.00 and the number of iterations is $15,000$. From Tables 2.2, 2.3, 2.4 and 2.5 we can analyze the performance of the different versions of the heuristic for different numbers of jobs.

Table 2.2: PD for each version of GRASP with the same seed and 15000 iterations, considering 20 jobs.

	Instances for 20 jobs										
Name	1	2	3	4	5	6	7	8	9	10	APD
GP	0.46	2.35	5.90	6.33	4.33	4.63	2.93	2.57	3.65	2.64	3.58
GPL	0.46	2.35	5.90	6.33	4.41	4.63	2.93	2.53	3.65	2.64	3.58
GPS	0.46	2.35	5.90	6.33	4.33	5.22	2.93	2.57	3.65	2.64	3.64
GPSL	0.46	2.35	5.90	6.37	4.33	5.30	2.97	2.53	3.65	2.64	3.65
GPF	0.46	2.35	5.90	6.33	4.33	4.93	2.93	2.57	3.65	2.64	3.61
GPLF	0.50	2.35	5.90	6.33	4.26	4.63	2.97	2.53	3.65	2.64	3.58
GPSF	0.46	2.35	5.90	6.33	4.33	5.30	2.97	2.53	3.65	2.64	3.65
GPSLF	0.50	2.35	5.90	6.33	4.33	5.26	2.93	2.53	3.65	2.64	3.64
GPC	0.50	2.35	5.90	6.33	4.22	5.30	2.97	2.57	3.65	2.64	3.64
GPLC	0.50	2.35	5.90	6.33	4.33	4.63	2.93	2.53	3.65	2.64	3.58
GPSC	0.50	2.35	5.90	6.37	4.33	5.44	2.97	2.57	3.65	2.64	3.67
GPSLC	0.50	2.35	5.90	6.37	4.44	5.30	3.00	2.57	3.65	2.64	3.67
GPSFC	0.50	2.35	5.90	6.50	4.41	5.59	2.97	2.57	3.65	2.64	3.71
GPSLFC	0.50	2.35	5.90	6.37	4.41	5.44	3.00	2.53	3.65	2.64	3.68
GPFC	0.50	2.35	5.90	6.33	4.41	4.63	3.00	2.57	3.65	2.64	3.60
GPLFC	0.50	2.35	5.90	6.43	4.44	4.93	2.97	2.53	3.65	2.64	3.63
Grand Average											3.632

Table 2.3: PD for each version of GRASP with the same seed and 15000 iterations, considering 60 jobs.

	Instances for 60 jobs										
Name	1	2	3	4	5	6	7	8	9	10	APD
GP	5.49	5.27	5.34	7.39	5.98	2.78	6.70	1.19	0.58	2.27	4.29
GPL	5.89	5.34	5.42	8.10	6.60	2.52	6.28	1.18	0.59	2.44	4.44
GPS	6.27	5.77	6.01	8.79	7.29	2.58	6.97	1.20	0.58	2.87	4.83
GPSL	6.38	5.41	6.03	8.17	7.02	2.61	7.21	1.22	0.59	2.68	4.73
GPF	6.00	6.64	6.06	8.52	7.42	2.71	6.06	1.36	0.66	2.76	4.82
GPLF	6.83	6.10	6.40	8.49	6.92	3.35	6.54	1.22	0.71	2.67	4.93
GPSF	7.09	6.42	6.43	9.97	7.99	3.33	8.37	1.40	0.66	3.20	5.48
GPSLF	6.84	7.09	7.16	10.26	8.21	3.69	7.99	1.39	0.69	3.17	5.65
GPC	6.36	6.16	5.98	8.07	6.96	3.11	6.74	1.22	0.66	2.78	4.80
GPLC	6.24	5.74	5.70	7.44	7.31	2.63	6.41	1.19	0.62	2.57	4.58
GPSC	6.93	6.38	6.21	8.84	7.28	3.33	7.63	1.33	0.66	3.32	5.19
GPSLC	6.99	5.99	6.19	8.93	7.80	3.71	7.15	1.28	0.65	3.09	5.18
GPSFC	7.41	6.94	6.91	8.90	8.39	4.24	8.32	1.73	0.79	3.41	5.70
GPSLFC	7.71	7.45	6.43	9.83	8.92	4.45	8.09	1.96	0.71	3.39	5.89
GPFC	5.14	6.61	6.61	8.25	7.52	3.56	7.39	1.33	0.78	2.73	4.99
GPLFC	6.17	5.85	6.51	9.56	6.85	3.75	7.23	1.69	0.71	3.19	5.15
Grand Average											5.042

In Table 2.6 and Figure 2.1 we present a summary of the results. Table 2.7 presents the

Table 2.4: PD for each version of GRASP with the same seed and 15000 iterations, considering 100 jobs.

	Instances for 100 jobs										
Name	1	2	3	4	5	6	7	8	9	10	APD
GP	7.17	1.05	3.18	1.05	6.38	4.31	2.96	1.10	1.30	1.89	3.04
GPL	6.77	1.12	3.03	1.64	6.33	4.20	3.28	0.97	1.25	1.74	3.03
GPS	7.48	1.38	4.11	1.31	6.75	5.35	3.92	1.18	1.49	2.17	3.51
GPSL	7.74	1.68	4.47	1.73	6.49	5.08	4.19	1.15	1.49	2.03	3.60
GPF	8.02	1.69	5.18	1.42	7.49	5.55	5.29	1.44	2.31	3.52	4.19
GPLF	7.77	2.38	4.42	1.84	6.96	6.27	5.43	1.73	2.60	3.28	4.27
GPSF	8.81	2.63	5.83	2.59	8.83	7.25	6.49	1.89	2.85	3.69	5.08
GPSLF	7.93	3.07	6.61	2.52	8.43	7.08	7.00	2.00	2.63	3.60	5.09
GPC	7.81	1.48	3.69	1.33	6.33	5.20	3.05	1.09	1.48	2.34	3.38
GPLC	7.77	1.88	4.45	1.76	6.88	5.10	4.55	1.12	1.73	2.29	3.75
GPSC	7.57	2.35	5.32	2.03	7.15	5.78	4.66	1.43	1.79	2.54	4.06
GPSLC	8.38	2.53	5.55	1.95	7.30	6.00	5.01	1.79	1.68	2.73	4.29
GPSFC	9.61	4.07	6.43	2.90	8.75	7.40	6.40	2.70	2.81	4.16	5.52
GPSLFC	9.84	3.50	6.57	2.55	8.72	7.04	6.63	2.62	3.15	4.07	5.47
GPFC	9.14	2.35	4.89	2.61	7.04	5.88	5.80	1.87	2.59	3.06	4.54
GPLFC	9.34	2.80	5.82	2.04	8.04	5.09	4.82	1.97	2.71	3.57	4.62
Grand Average											4.216

Table 2.5: PD for each version of GRASP with the same seed and 15000 iterations, considering 150 jobs.

	Instances for 150 jobs										
Name	1	2	3	4	5	6	7	8	9	10	APD
GP	1.02	1.26	3.78	0.70	4.26	4.25	3.50	0.81	7.73	0.97	2.83
GPL	0.98	1.15	3.42	0.71	5.04	3.59	3.69	0.81	8.47	1.07	2.89
GPS	1.05	1.60	3.87	0.74	5.17	5.24	4.06	0.86	8.65	1.02	3.23
GPSL	1.03	1.78	3.71	0.73	5.03	5.15	4.12	0.86	8.63	1.36	3.24
GPF	1.42	2.94	6.23	1.09	7.18	6.89	6.54	1.36	9.60	2.44	4.57
GPLF	1.54	3.08	5.16	1.36	6.86	6.91	7.61	1.03	12.67	2.55	4.88
GPSF	1.91	4.02	6.44	1.18	8.27	8.04	7.32	1.63	12.83	3.34	5.50
GPSLF	2.29	3.93	6.14	1.49	7.61	8.52	8.23	1.71	12.02	2.68	5.46
GPC	1.16	2.05	4.38	0.83	5.59	4.97	4.67	0.87	8.53	1.28	3.43
GPLC	0.96	2.44	4.34	0.70	4.74	4.93	4.74	0.89	9.64	1.38	3.48
GPSC	1.07	2.36	4.87	0.86	6.23	5.80	5.36	0.97	10.11	1.42	3.90
GPSLC	1.26	2.40	5.19	0.79	5.78	5.71	4.92	0.98	10.54	1.49	3.90
GPSFC	2.48	4.14	6.92	1.90	8.52	8.46	8.37	2.00	14.27	3.14	6.02
GPSLFC	2.38	4.68	7.08	1.94	8.04	7.49	8.06	2.14	13.96	2.72	5.85
GPFC	1.87	3.87	6.18	1.32	7.90	5.63	6.75	1.94	12.31	2.77	5.06
GPLFC	1.88	4.16	7.11	1.32	7.71	6.53	6.11	1.79	13.55	3.05	5.32
Grand Average											4.348

average computation time spent for each version considering different instance sizes: 60, 100 and 150 jobs.

From these data, it is apparent that the GP and GPL approaches present the best average results without additional computation time. In general, the different versions have a good behavior except the last four or five.

The poor performance of some versions are caused by the junction of the linear probability function ($F2$) and the random construction phase (C2). These two parameters cause an

Figure 2.1: APD for the six versions with the best results, considering 60, 100 and 150 jobs. Time in seconds

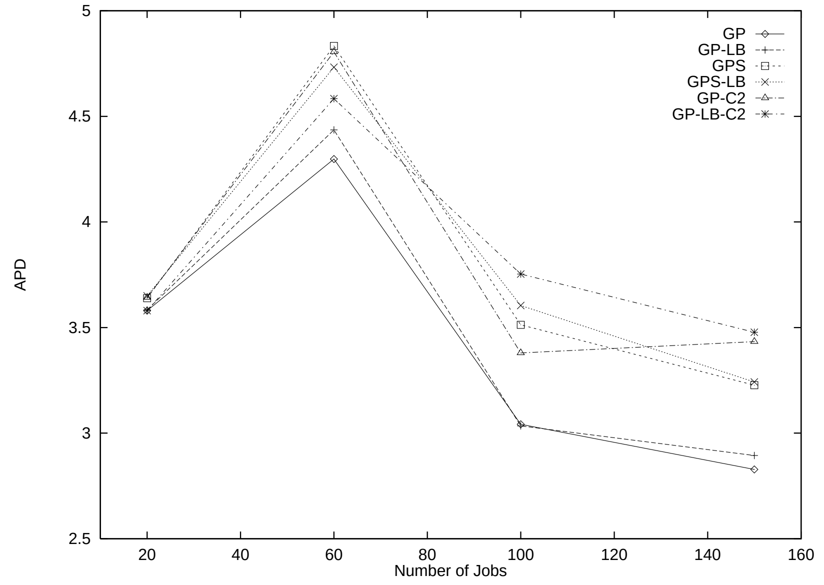


Table 2.6: APD for each version of GRASP and for each instance size.

Version	20 jobs	60 jobs	100 jobs	150 jobs	Overall APD
GP	3.580	4.298	3.042	2.828	3.437
GPL	3.584	4.436	3.035	2.894	3.487
GPS	3.639	4.833	3.513	3.227	3.803
GPSL	3.650	4.732	3.604	3.242	3.807
GPF	3.609	4.819	4.190	4.568	4.297
GPLF	3.576	4.923	4.267	4.878	4.411
GPSF	3.647	5.485	5.085	5.497	4.928
GPSLF	3.643	5.648	5.087	5.463	4.960
GPC	3.642	4.805	3.380	3.433	3.815
GPLC	3.580	4.584	3.753	3.478	3.849
GPSC	3.672	5.193	4.062	3.905	4.208
GPSLC	3.671	5.178	4.293	3.905	4.262
GPSFC	3.707	5.705	5.523	6.020	5.239
GPSLFC	3.679	5.893	5.471	5.850	5.223
GPFC	3.598	4.993	4.524	5.056	4.542
GPLFC	3.634	5.150	4.621	5.319	4.681

extended search in the solution space. This kind of approach could bring excellent results or the worst results. This characteristic could be very interesting when using a larger local search or for a multi-criteria approach.

In a similar way we can conclude that the best performances occur when only few modifications are made in the greedy construction phase procedure. That is translated to our algorithm as the use of the $F1$ probability function and the deterministic schedule of the jobs.

It is worth noting that as the number of jobs increases, the APD values decrease, as expected. The effectiveness of the lower bound improves with the increase in the number of

Table 2.7: Average computation times considering 60, 100 and 150 jobs. Time in seconds

	60	100	150
GP	18.59	61.38	144.79
GPL	18.46	57.66	145.74
GPS	18.35	57.34	143.89
GPSL	18.13	60.32	144.34
GPF	19.00	59.36	147.74
GPLF	18.95	57.91	148.18
GPSF	18.68	58.46	148.91
GPSLF	18.57	59.62	145.70
GPC	18.99	60.84	146.18
GPLC	18.91	58.68	146.57
GPSC	18.70	58.91	149.77
GPSLC	18.65	60.51	145.16
GPSFC	19.05	60.30	146.72
GPSLFC	18.96	61.48	147.25
GPFC	19.41	59.13	149.43
GPLFC	19.35	58.53	150.10

Table 2.8: APD results for the GP version considering different number of iterations [thousands](second part).

			GP			
Iterations	1	2	3	4	5	**APD**
15000	0.912	1.236	3.290	0.712	4.730	2.176
30000	0.912	1.111	3.042	0.712	4.716	2.099
50000	0.907	1.107	3.042	0.712	4.171	1.988
100000	0.907	1.107	3.042	0.712	4.014	1.957
200000	0.903	1.107	2.995	0.712	4.014	1.946
300000	0.903	1.107	2.939	0.699	4.014	1.932

jobs.

2.3.6.6 Second part.

In this part of the tests we analyze the heuristic performance for a different number of iterations. We choose five versions of GRASP and a set of five problems with 150 jobs. We keep the seed used in the first part (1.00).

In Tables 2.8 to 2.12, we present the percentage deviation for the five instances and its average for the chosen versions.

From these results we can conclude the GP and GPL present better responses to this kind of problem. We can also observe that we obtain a solution improvement, but the increase in the number of iterations also affects the computation time. The average CPU time for $300,000$ iterations is near $2,500$ seconds (150 jobs).

For the reasons above, the use of a large number of iterations depends more on the situation that is to be solved, that is, on the trade off between the benefits of a long run and time expended. In our particular case, a large number of iterations is interesting for the monthly

Table 2.9: APD results for the GPL version considering different number of iterations [thousands](second part).

GP-LB

Iterations	1	2	3	4	5	**APD**
15000	0.952	1.209	3.262	0.676	4.512	2.122
30000	0.930	1.209	3.243	0.648	4.512	2.108
50000	0.930	1.209	3.019	0.648	4.512	2.063
100000	0.930	1.209	2.701	0.648	4.507	1.999
200000	0.930	1.102	2.626	0.648	4.071	1.875
300000	0.930	1.089	2.626	0.648	4.066	1.872

Table 2.10: APD results for the GPLF version considering different number of iterations [thousands](second part).

GPLF

Iterations	1	2	3	4	5	**APD**
15000	1.520	3.098	4.883	1.187	7.090	3.556
30000	1.520	3.098	4.883	1.110	7.043	3.531
50000	1.520	3.098	4.883	1.050	7.043	3.519
100000	1.317	3.098	4.883	0.945	6.929	3.434
200000	1.088	2.787	4.776	0.927	6.929	3.301
300000	1.066	2.658	4.776	0.927	6.929	3.271

Table 2.11: APD results for the GPLFC version considering different number of iterations [thousands](second part).

GPLFC

Iterations	1	2	3	4	5	**APD**
15000	1.731	3.373	6.542	1.379	7.417	4.089
30000	1.731	3.351	6.215	1.187	7.417	3.980
50000	1.731	3.351	5.986	1.187	7.412	3.934
100000	1.731	3.320	5.986	1.187	7.412	3.927
200000	1.326	3.320	5.921	1.187	6.464	3.644
300000	1.326	3.320	5.533	1.187	6.431	3.559

Table 2.12: APD results for the GPS version considering different number of iterations [thousands](second part).

GPS

Iterations	1	2	3	4	5	**APD**
15000	0.982	1.391	3.799	0.731	5.175	2.416
30000	0.982	1.391	3.687	0.726	5.175	2.392
50000	0.982	1.391	3.636	0.726	4.654	2.278
100000	0.969	1.378	3.636	0.726	4.654	2.273
200000	0.969	1.378	3.369	0.726	4.464	2.181
300000	0.956	1.378	3.369	0.703	4.464	2.174

schedule program. However, in the daily decision process of the company, new schedules are needed to support decisions, and the use of a larger number of iterations is not very convenient. These daily problems are usually related to new contracts, new clients or new due dates.

2.3.6.7 Third part.

In this part of the test the focus is on the heuristic's seed dependence. We choose only one instance with 150 jobs and we set the number of iterations to $15,000$. The seeds used are randomly chosen.

In Table 2.13 and Figure 2.2 we present the results. In this case is not necessary to use the APD form because we are interested in the variability of the solution. The difference between the worst and the best result for each version is also shown. The best result for each version is in italics. The best overall result is in bold typeface.

These results are very interesting, especially if associated with the previous ones. In the second part of the tests, it is possible to realize that the APD improvement made by the increase of the number of iterations vary between 0.242 to 0.53. And in this test, it is possible to see that the improvement could reach the 1% of the objective value. We can conclude that it appears to be more interesting to use several seeds with shorter runs, than one seed with a larger run.

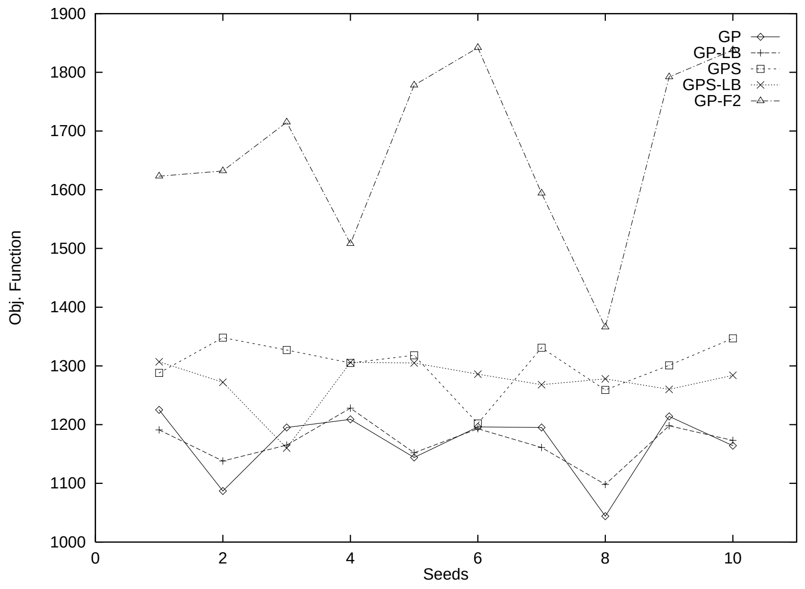
Table 2.13: Results considering different seeds (third part)

Version	1	2	3	4	5	6	7	8	9	10	Diff.
GP	1225	1087	1195	1209	1144	1196	1195	**1044**	1214	1164	181
GPL	1191	1138	1165	1228	1152	1193	1161	*1098*	1198	1173	130
GPS	1288	1348	1327	1305	1318	*1202*	1331	1259	1301	1347	146
GPSL	1307	1272	*1160*	1306	1305	1286	1268	1278	1260	1284	147
GPF	1623	1632	1715	1508	1778	1842	1594	*1366*	1792	1838	476
GPLF	1623	1632	1715	1508	1778	1842	1594	*1366*	1792	1838	476
GPSF	1908	*1757*	1878	1916	1863	1904	1940	1939	1915	2013	256
GPSLF	1920	1845	2004	1964	1997	1863	2011	1857	*1668*	1883	343
GPC	1178	1390	1512	1322	1283	1387	*1266*	1406	1268	1407	334
GPLC	1376	1383	1336	1345	*1162*	1390	1232	1356	1297	1295	228
GPSC	1442	1460	*1365*	1516	1443	1423	1470	1462	1476	1479	151
GPSLC	1458	1494	1396	1415	1476	1481	*1384*	1544	1480	1426	160
GPSFC	1986	2071	2007	1981	2048	2013	2000	2027	1955	*1929*	142
GPSLFC	1868	*1824*	1970	1892	2079	2000	2003	1969	2001	1884	255
GPFC	*1640*	1807	1748	1929	1871	1860	1841	1750	1865	1806	289
GPLFC	1796	1797	1841	1724	1708	1777	1702	1810	1927	*1649*	278

2.3.6.8 Fourth part.

In this experiment we use path relinking as a form of intensification of the local search, applied in each iteration or after 2000 iterations. We used three instances of 100 jobs to analyze how PR affects the performance of the heuristic.

Figure 2.2: Heuristic's seed dependence,(third part)



We use the *GP* version considering 50000 iterations and 6 different frequencies of path relinking (FPR), 1, 2000, 5000, 10000, 15000 and 20000. The results are presented in Table 2.14.

We can observe that the use of PR at different steps does not cause a significant reduction of CPU time. With these tests we can conclude that PR is not relevant in the computational time spent by the heuristic.

The use of path relinking is shown to be effective without an increase of CPU time. For our implementation, the most expensive procedure is the local search.

Table 2.14: Results applying PR after different numbers of iterations (FPR) for the *GP* version.

	Inst. 1		Inst. 2		Inst. 3	
FPR	Obj. Func.	time [sec]	Obj. Func.	time [sec]	Obj. Func.	time [sec]
1	1027	115.77	290	117.13	623	119.81
2000	1300	116.11	340	115.58	757	119.59
5000	1300	116.08	340	115.64	757	119.57
10000	1300	115.61	340	115.63	757	120.24
15000	1300	115.62	340	115.62	757	120.19
20000	1300	115.67	340	115.66	757	120.16

2.3.6.9 Fifth part.

Until now, we always used the same machine configuration, a group of four with identical characteristics and two unrelated machines. For this particular configuration, the use of GRASP shows a satisfactory performance.

In this experiment we test the flexibility of GRASP by analyzing the performance of the method for other configurations of machines. The tested configurations are:

- $M6$ - six identical machines.
- $M321$ - A group of three identical machines, two identical machines but different from the rest and one unrelated machine.
- $M3111$ - A group of three identical machines and other three unrelated machines.
- $M222$ - Three different groups of two identical machines.
- $M2211$ - Two different groups with two identical machines and two unrelated machines.
- $M21111$ - A group of two identical machines and four unrelated machines.

In this experiment we used five instances of 150 jobs. The seed remains the same for all the tests (1.00), and the number of iterations is set to $15,000$.

Because of the different configurations we can not compare the results directly. Still, in Table 2.15 we can observe that the heuristic produces interesting results for these new instances. The use of this methodology seems to be very suitable for the UPMSP environment.

Table 2.15: APD for different configurations of machines

	M6	M3111	321	222	2211	21111
GP	0.386	0.719	0.635	0.478	0.561	0.616
GPL	0.378	0.720	0.636	0.482	0.558	0.618
GPLF	0.491	1.008	0.936	0.699	0.837	0.896
GPLFC	0.533	1.041	1.037	0.771	0.884	0.989
GPS	0.392	0.726	0.652	0.480	0.571	0.625

2.4 Variable Neighborhood Search

The VNS architecture and notation used here follow those proposed by Hansen and Mladenovic (2003).

Lets consider a set of machines $\mathcal{M} = \{1, 2, .., m\}$ and a set of jobs $\mathcal{J} = \{1, 2, .., n\}$ with positive processing times p_{jk} and positive weights w_j, for each job $j \in \mathcal{J}$ and each machine $k \in \mathcal{M}$.

As it was previously stated, the problem consists in scheduling all the jobs minimizing the sum of the completion time, also known as makespan, plus the sum of the weighted delays.

In order to process a job j after a job j' a setup time $s_{jj'k}$ is required and depends both on the sequence of jobs $j \prec j'$ and on the machine $k \in \mathcal{M}$ where they are processed.

2.4.1 VNS algorithm

Define the completion time of job j recursively as $C_j = C_{j'} + p_{jk} + s_{j'jk}$, where $C_{j'}$ is zero if job j is the first job scheduled or the completion time of job j' otherwise. The tardiness ρ_j of each job is calculated as $\rho_j = max((C_j - d_j), 0)$, where d_j is the due date for job j. Our goal is to minimize $C_{max} + \sum_{j=1}^{n} \nu_j.\rho_j$.

First we define a job sequence J_k for machine k as a permutation of the elements of the subset $\mathcal{I}_k \subseteq 2^{\mathcal{J}}$, where $2^{\mathcal{J}}$ is the set of all subsets of $\mathcal{J}$, and:

$$\mathcal{J}_k \cap \mathcal{J}_{k'} = \emptyset \qquad \forall k \in \mathcal{M}, k \neq k'$$

$$\bigcup_{k \in \mathcal{M}} \mathcal{J}_k = \mathcal{J}$$

Let $\mathcal{F}$ be the set of all feasible solutions and consider a solution $\mathcal{S} = \{J_1, J_2, .., J_m\} \in \mathcal{F}$. A *Neighborhood structure* associates each $\mathcal{S} \in \mathcal{F}$ with a neighborhood $\mathcal{N}_l(\mathcal{S}) \subseteq \mathcal{F}$ of solution $\mathcal{S}$.

For this approach three neighborhoods structures are defined:

1. Job swaps on one machine. One machine is chosen and all possible job swaps are considered.

2. Job swaps between two different machines. Two machines are chosen and all possible job swaps from different machines are considered.

3. Job transfers from one machine to another. One machine is chosen and all possible job movements from this machine to any other are considered.

Note that the neighborhoods themselves $\mathcal{N}_1(\mathcal{S})$, $\mathcal{N}_2(\mathcal{S})$, and $\mathcal{N}_3(\mathcal{S})$ respectively, are determined by both its respective structure and the solution it is being applied to.

The size of Neighborhood $\mathcal{N}_1(\mathcal{S})$ is $O(m.n^2)$, of Neighborhood $\mathcal{N}_2(\mathcal{S})$ is $O(m^2.n^2)$ and the size of Neighborhood $\mathcal{N}_3(\mathcal{S})$ is $O(m.n^2)$.

Algorithm2.3 shows the VNS structure used in this dissertation.

2.4.2 The initial solution

Any method capable of generating a feasible solution would be sufficient for the first part of our algorithm. However, as shown by Johnson et al. (1989) and Matsuo et al. (1989), a good initial solution can reduce considerably the computation time.

In this work we propose the use of an algorithm based on NEH, (Nawaz et al., 1983). The NEH has proven to be very advantageous over other polynomial approaches of obtaining an initial solution for the flow shop scheduling problem.

Since the target of this implementation are large instances, the gain provided by this algorithm is worth to its additional computational time, which is small if compared to the

Algorithm 2.3: Basic VNS structure

```
Find an Initial Solution S*;
l ← 1;
for Iterations ← 1 to MaxIterations do
    S ← S*;
    Shake procedure: Find a random solution S' ∈ $\mathcal{N}_l(S)$;
    S" ← Local Search($\mathcal{N}_l(S')$);
    if S" < S* then
        S* ← S";
        l ← 1;
    else
        l ← l+1;
    end
end
end
```

time spent by the local searches. Our implementation has a time complexity $O(n^3.m)$, and it is designed as shown in Algorithm 2.4.

Algorithm 2.4: NEH algorithm adaptation for the parallel machine case.

```
Sort the jobs in order of due dates;
for each job j do
    MKS ← INT-MAX;
    for each machine k do
        for each position p in machine k do
            MKS'= INSERT job j on k at position p;
            if MKS' < MKS then
                $MKS \leftarrow MKS'$;
            else
                Remove j from k at position p;
            end
        end
    end
end
end
```

2.4.3 Random solutions

Every time a neighborhood is selected, a random procedure is called. This procedure selects a random solution from the selected neighborhood structure. Therefore, three procedures are created in the following manner, one for each neighborhood structure ($\mathcal{N}_l$):

1. For $\mathcal{N}_1(S)$:

Algorithm 2.5: Local search 1. Job swaps in one machine.

```
1 for each $k \in \mathcal{M}$ do
2     for each $j_1$ in $k$ do
3         for each $j_2$ in k, $j_1 \neq j_2$ do
4             if Solution considering $j_1$ and $j_2$ swapped < Current Solution then
5                 Swap($j_1$,$j_2$);
6             end
7         end
8     end
9 end
```

- Choose a machine k at random.
- Choose 2 jobs j_1 and j_2 at random in k.
- Swap jobs j_1 and j_2.

2. For $\mathcal{N}_2(S)$:

   - Choose 2 machines k, k' at random.
   - Choose a job j_1 in k and a job j_2 in k', both at random.
   - Swap jobs j_1 and j_2.

3. For $\mathcal{N}_3(S)$:

   - Choose 1 job j_1 and 1 machine k' at random, where j_1 is not allocated in k'.
   - Choose a valid position *pos* in k' at random.
   - Transfer job j_1 to k' at position *pos*.

2.4.4 The local searches

There are several variations of the VNS structure. In our version, we use a specific local search for each neighborhood. The local searches are listed below:

LS 1. Job swaps at one machine. This local search analyzes every possible swap on one machine. Even when the chosen machine is not the one with the greatest completion time, the objective function can be improved by improving the job delays. The algorithm presented in Algorithm 2.5 has a time complexity $O(m.n^2)$.

LS 2. Job swaps on different machines. In this local search, all job swaps between jobs belonging to different machines are evaluated. A larger amount of solutions are searched. The time complexity of our algorithm is $O(m^2.n^2)$, see Figure 2.6

Algorithm 2.6: Local search 2. Job swaps belonging to different machines.

```
for each $k \in \mathcal{M}$ do
    for each $j_1$ in $k$ do
        for each $k' \in \mathcal{M}, k \neq k'$ do
            for each $j_2 \in k'$ do
                if Solution considering $j_1$ and $j_2$ swapped < Current Solution then
                    Swap($j_1$,$j_2$);
                end
            end
        end
    end
end
```

Algorithm 2.7: Local search 3. Job insertion from the machine with worst value of makespan to a machine with the better one.

```
Find the machine with the highest makespan $k$. Find the machine with the lowest
makespan $k'$, $k \neq k'$. for each $j$ in $k$ do
    for each valid position pos in $k'$ do
        if Solution considering $j$ transferred from $k$ to $k'$ in position pos < Current
        Solution then
            Transfer $j$ from $k$ to $k'$ on position pos.
        end
    end
end
```

LS 3. Job insertion. This procedure searches for new solutions transferring jobs from the machine with the highest makespan to the machine with the lowest one. The time complexity of our implementation is $O(n^2)$, see Algorithm 2.7.

The algorithm always tries to use the local search LS 1 first. If after an iteration no improvement is made, then another neighborhood is used (l is incremented), and every time a new solution is found, the first local search is used (l= 1).

For instances with long range due dates, local search 1 is effective only when the chosen machine has the highest completion time. For instances with short range due dates, local search 1 can improve the objective function by improving some of the job delays, saving computational time. The feasible space searched by the other two local searches are much larger than the first one.

The bad behaved instances which have both some short and long due dates are also handled in a convenient way using all the local searches. Further reading about neighborhood search techniques can be found on (Ahuja et al., 2002).

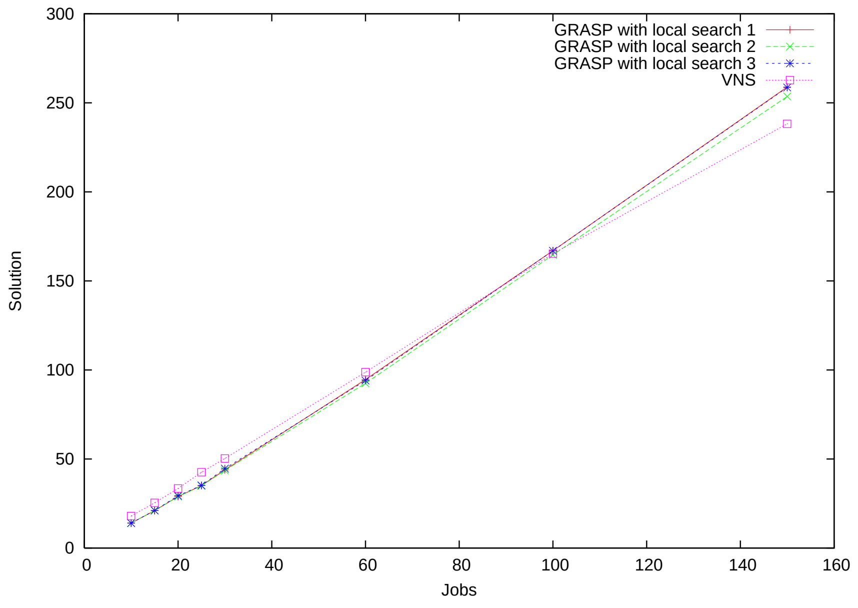


Figure 2.3: Results for instances with long range due dates. VNS is indicated by the curve with squares as vertexes.

2.5 Computational results

In this section we present the computational results of several experiments considering scheduling problems with different categories of due dates. All the algorithms are coded in C, using version 4.0.3 of the GCC compiler. The experiments run on a Pentium 4 computer, 3.0 GHz CPU and 1 GB of RAM memory under a Debian GNU/Linux environment.

Random instances are used in this work. Details about how the instances were generated are presented in Section 3.4. All instance files can be found at the *UFMG Scheduling Group* home page[4].

Table 2.16: Three different versions of GRASP

Name	Local Search
GRASP 1	LS 1: Job Swaps on the same machine.
GRASP 2	LS 2: Job Swaps on different machines.
GRASP 3	LS 3: Insertion of a Job on a different machine.

[4]http://www.dcc.ufmg.br/lapo/wiki/index.php/Scheduling_Team

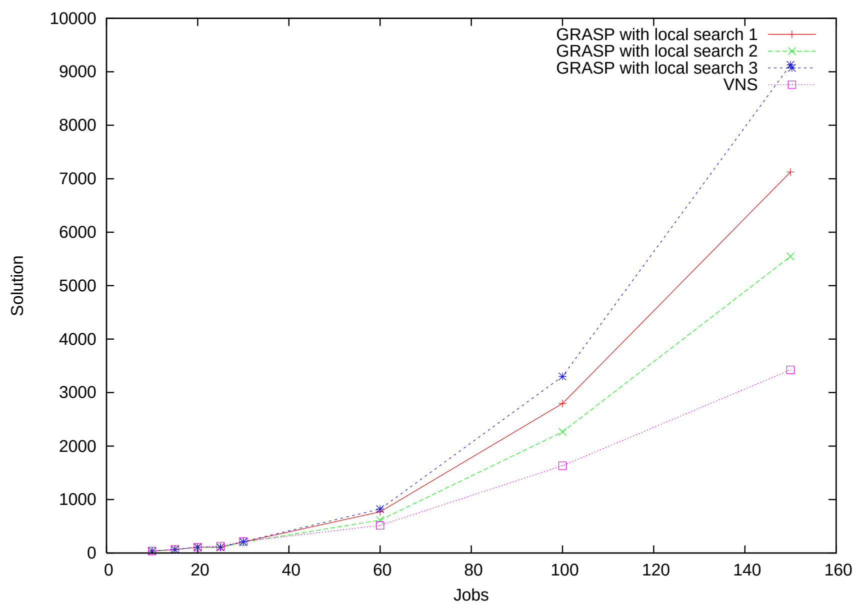


Figure 2.4: Results for instances with short or long range due dates. VNS is indicated by the curve with squares as vertexes.

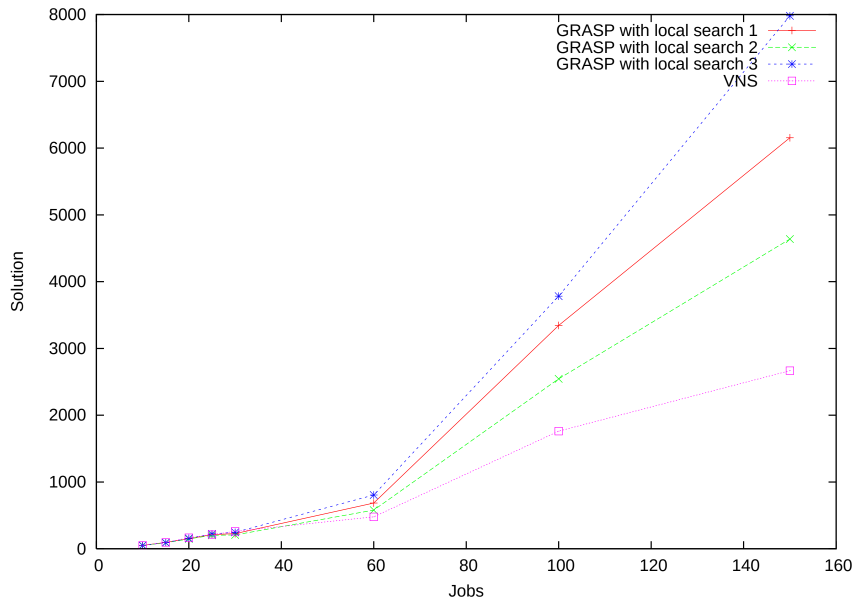


Figure 2.5: Results for instances with short range due dates. VNS is indicated by the curve with squares as vertexes.

2.5.1 Results

For this experiment, 8 sets, with 10, 15, 20, 25, 30, 60, 100 and 150 jobs, are used. Each set is composed by 30 instances with due dates ranges progressively shorter. Each instance is solved using ten different seeds and the average solution and computation time is considered. The number of iterations performed by each algorithm is fixed in 10,000. Once again, the results are presented considering the average percentage deviation.

The results presented in this section are focusing two main objectives, to prove the efficiency of VNS for this scheduling problem and comparing the VNS approach with GRASP. For this second objective, we are using three different versions of GRASP based on the previous description of the heuristic § 2.3 but incorporating the three local searches defined for the VNS approach, Table 2.16 shows the different versions of GRASP.

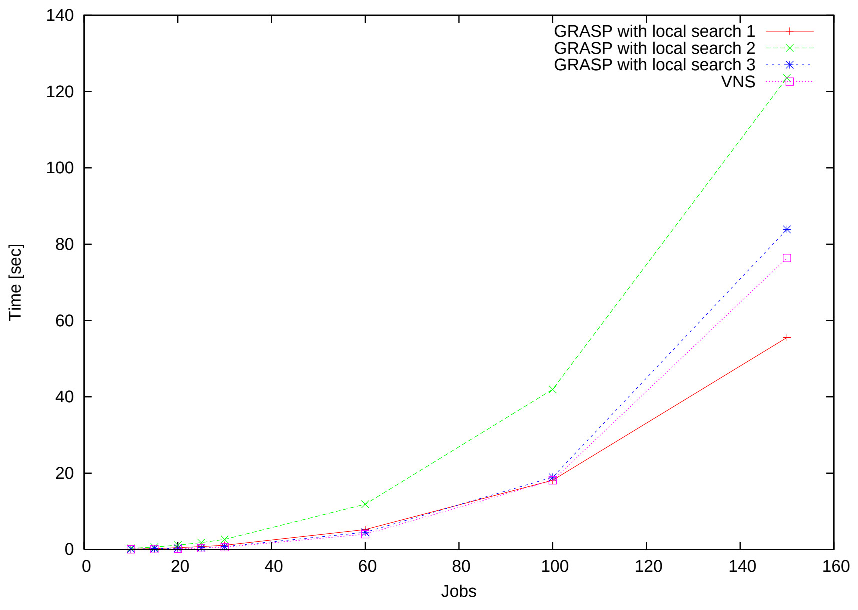


Figure 2.6: Computation time [seconds] for instances with short or long due dates. VNS is indicated by the curve with squares as vertexes.

Figures 2.5, 2.3 and 2.4 compare the computational results of the three versions of GRASP and the VNS, considering three types of instances: short range due dates, long range due dates and with both, short and long range due dates.

In all the three categories of due dates, the VNS algorithm provides better average solutions when considering more than 60 jobs.

For short range due dates (Figure 2.5) the VNS algorithm presents a superior performance and by analyzing the GRASP curves it is possible to realize that the local search 2 suit this case better. For long range due dates (Figure 2.3), the focus of the algorithm is on the makespan, and the performance of them all is similar, but when considering more than 100 jobs the VNS algorithm begins to present better results. For the case with mix range due dates (Figure 2.4), as expected, once again the VNS obtains a better performance, when considering 60 or more jobs.

As for the computational time, the combination of the local searches allows the VNS to obtain good results. As we can see in Figure 2.6, the VNS gets better results than two versions of GRASP, specially for larger instances.

Figure 2.7 presents the APD confidence intervals for the group of instances with 150 jobs. We conclude, that the VNS algorithm provides better average results than GRASP 1 and 3, in 95% of the cases. Even with an interception between the confidence intervals for the VNS and the GRASP 2, it is clear the VNS results have a propensity to be superior with larger instances.

Table 2.17 shows the APD values for each instance, each value is the average of ten runs. The first group of instances (1-10) have a short range due dates, form 11-20 the range is longer and finally instances (21-30) have long range due dates. We conclude that the algorithm is providing good results, specially for bigger instances.

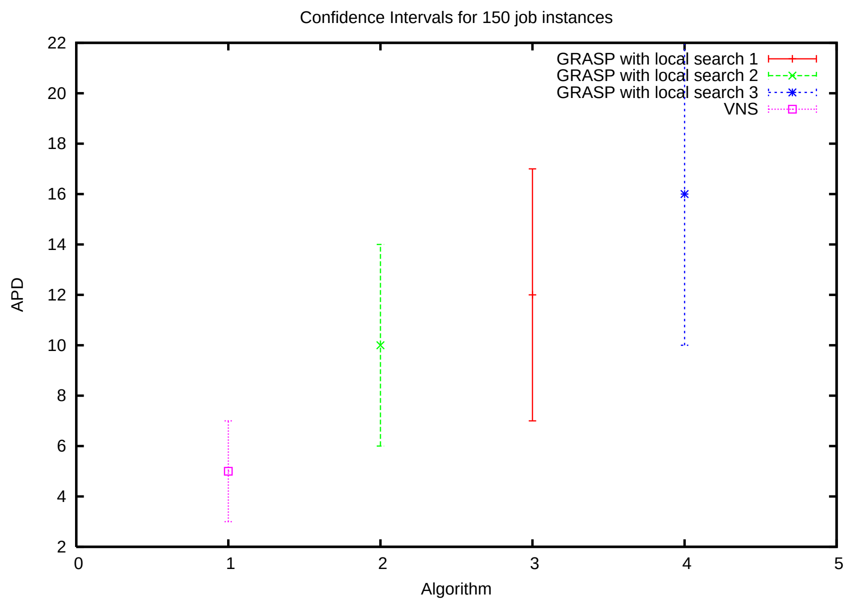


Figure 2.7: APD confidence intervals for 150 job instances. GRASP1, GRASP2 and GRASP3 represents the GRASP algorithms using local searches 1,2 and 3, respectively (see subsection 2.4.4).

Table 2.17: APD results for the VNS algorithm. Each column presents results for a different number of jobs.

Inst	**10**	**15**	**20**	**25**	**30**	**60**	**100**	**150**
1	3.228	4.582	5.879	6.056	6.452	5.646	13.918	14.062
2	3.875	4.648	3.975	4.475	6.340	7.522	16.052	12.135
3	4.234	3.676	4.580	5.045	5.714	5.132	13.792	17.426
4	3.056	3.800	4.915	4.754	7.050	9.338	10.408	21.599
5	2.503	3.128	3.697	3.378	5.004	6.263	12.494	17.611
6	4.982	4.324	3.992	5.992	5.890	7.530	12.495	21.608
7	3.326	4.760	3.938	5.426	4.745	9.332	13.060	16.352
8	4.654	4.181	6.926	6.000	5.456	8.195	12.688	15.076
9	7.763	3.795	4.778	5.292	6.692	7.366	15.565	16.572
10	2.793	4.640	4.705	3.706	5.721	7.074	14.845	20.215
Avg	4.041	4.153	4.739	5.012	5.906	7.340	13.532	17.266
11	0.906	0.530	0.572	0.733	0.516	0.375	0.338	0.335
12	3.744	0.835	0.406	0.383	0.563	0.424	0.378	0.375
13	1.400	0.594	0.549	0.744	0.536	0.388	0.345	0.330
14	0.532	0.946	1.110	0.382	0.800	0.438	0.339	0.320
15	0.901	0.535	0.382	0.437	0.424	0.388	0.353	0.304
16	2.021	0.536	0.383	0.500	0.358	0.465	0.421	0.451
17	2.458	0.819	0.394	0.520	0.473	0.434	0.383	0.322
18	0.640	0.580	1.194	0.496	0.494	0.387	0.358	0.316
19	0.910	0.542	0.492	0.536	0.548	0.368	0.360	0.331
20	0.546	1.046	0.517	0.452	0.497	0.414	0.358	0.627
Avg	1.406	0.696	0.600	0.518	0.521	0.408	0.363	0.371
21	1.184	0.457	0.585	0.542	0.417	0.430	0.369	0.340
22	0.850	0.613	0.442	0.423	0.478	0.412	0.373	0.384
23	0.586	0.522	0.418	0.459	0.402	0.393	0.373	0.319
24	0.621	0.437	0.457	0.381	0.454	0.394	0.332	0.322
25	0.499	0.560	0.350	0.525	0.437	0.377	0.363	0.352
26	1.018	0.608	0.391	0.490	0.433	0.428	0.381	0.359
27	1.083	0.523	0.464	0.532	0.523	0.422	0.424	0.319
28	0.739	0.698	0.735	0.445	0.530	0.406	0.346	0.335
29	0.574	0.548	0.579	0.553	0.423	0.446	0.363	0.347
30	0.640	0.494	0.451	0.469	0.437	0.413	0.401	0.346
Avg	0.779	0.546	0.487	0.482	0.453	0.412	0.373	0.342
Total Avg	2.076	1.799	1.942	2.004	2.294	2.720	4.756	5.993

2.6 Concluding remarks and Future Research

In this chapter we propose two heuristic approaches that proved to be very efficient to tackle a scheduling problem with realistic constraints.

The GRASP algorithm proved to be very efficient and flexible, it was very easy to setup its parameters and use different local searches. The results obtained with GRASP were competitive.

The algorithm based on VNS to solve large instances of this scheduling problem was also proposed and tested. We run a series of experiments comparing the VNS algorithm with three versions of GRASP, each one with a different local search. Analyzing the computational results, we conclude that the VNS provides very good average results for instances with 60 jobs or more, specially for short range due dates. The combination of three different local searches proved to be very effective with a simple implementation and with no need of adjusting several parameters.

The NEH also proved to be remarkable as a procedure to find initial solutions. And before taking any conclusions between both approaches, it is important to remember that the GRASP versions do not use the NEH as first heuristic and a significant part of the difference between both algorithms could be related to this fact. More tests are needed to analyze the relevance of using NEH as first approach. In Appendixes B and C several tests are shown analyzing the relationship between GRASP and exact algorithms and between GRASP and NEH.s

Since the VNS approach proved itself advantageous to solve large instances of scheduling problems with realistic constraints, further work would include a deeper study of various local search techniques among improvements in other relevant parts of the VNS structure.

Chapter 3

Models and a Branch and Bound Algorithm

3.1 Introduction

In this chapter, two different models are proposed, compared and tested. A Branch-and-Bound algorithm is also presented and tested.

The main objectives of working with mathematical models for these complex problems are to understand its nature and to detect advantages to improve and/or create new efficient approaches to solve them.

It is important to remember that the problem addressed in this work is a scheduling problem with unrelated parallel machines, due dates, and setup times that depend on both job sequence and machine. It consists of programming several jobs to be processed by several machines. Each job should be scheduled to a specific machine, and the order each machine will process its jobs should be decided. It takes each job a different time to be processed by each machine. When a machine finishes processing a job, it has a setup period before processing the next. This setup time exists so that the machine can be prepared for the next job (e.g. cleaned or reconfigured) and is also different for each pair of jobs and each machine. Each job also has a due date (maximum time when the job should be completed) and a weight (priority). Though we allow the due date to be exceeded, there will be a penalty in the objective function depending on the job weight.

3.2 Models

There are a few ways to get this problem modeled, but before introduce and analyze two of them, we define a common notation. This notation will be valid through the dissertation and when needed, a specific notation will be added. The general notation is presented bellow.

`Parameters`

N	set of jobs to be processed.
M	set of machines.
p_{im}	Processing time of job i by machine m.
$s_{ii'm}$	Time for setting up machine m, from processing job i to processing job i'.
d_i	Due date of job i.
ν_i	Tardiness penalty coefficient of job i.
$g_{ii'm}$	Time between the beginning of job i and the conclusion of setup (i, i'). $g_{ii'm} = p_{im} + s_{ii'm}$.
C_m	Number of available time units in machine m.
G	Sufficiently large positive constant.

3.2.1 First Model

The first model to be introduced is a model based on a group constraints proposed by Manne (1960), for the job shop scheduling problem. As is possible to see, this model presents an easy reading and understanding. The group of specific variables used in this model is presented below.

- t_i: processing start time of job i;
- α_{im}: $\begin{cases} 1 & \text{if job } i \text{ is processed on machine } m, \\ 0 & \text{otherwise;} \end{cases}$
- $\beta_{ii'm}$: $\begin{cases} 1 & \text{if jobs } i \text{ and } i' \text{ are processed on machine, and } i \text{ is processed before } i', \\ 0 & \text{otherwise;} \end{cases}$
- ρ_i: tardiness of job i;
- Z: makespan (maximum completion time). $\begin{cases} Z_m & \text{makespan of machine } m \\ Z_t & \text{Total makespan} \end{cases}$

Let us first define E^m as the group of all pairs of jobs (i, i') that will be produced by the same machine m. The MILP model is presented below.

$$\min\left(Z + \sum_{i}^{N} (\rho_i \cdot \nu_i)\right) \tag{3.1}$$

$$\sum_{m}^{M} \alpha_{im} = 1, \qquad \forall i \in N \tag{3.2}$$

$$Z \geq t_i + p_{im} - (1 - \alpha_{im}) \cdot G, \qquad \forall i \in N, m \in M \tag{3.3}$$

$$d_i + \rho_i \geq t_i + p_{im} - (1 - \alpha_{im}) \cdot G, \qquad \forall i \in N, m \in M \tag{3.4}$$

$$(1-\alpha_{im})\cdot G+(1-\alpha_{i'm})\cdot G+(1-\beta_{ii'm})\cdot G+t_{i'}\geq t_i+p_{im}+s_{ii'm},\forall(i,i')\in E^m,m\in M \quad (3.5)$$

$$(1-\alpha_{im})\cdot G+(1-\alpha_{i'm})\cdot G+\beta_{ii'm}\cdot G+t_i\geq t_{i'}+p_{i'm}+s_{i'im}, \qquad \forall(i,i')\in E^m,m\in M \quad (3.6)$$

$$\alpha_{im}\in\{0,1\}, \qquad \forall i\in N,m\in M$$

$$\beta_{ii'm}\in\{0,1\}, \qquad \forall(i,i')\in E^m,i'\in N,m\in M$$

$$t_i,\rho_i\geq 0, \qquad \forall i\in N$$

$$Z\geq 0$$

Constraints (3.5) and (3.6) describe the precedence relationship between the jobs. These are the constraints proposed by Manne (1960).

This model considers that all processing times are larger than setup times, if this situation does not happen the model must be slightly modified.

The use of variables β is also worth to notice. In our model, these variables indicate a precedence relation between two jobs, but not necessarily immediately before each other. This approach presents better performance than when variables Betas mean specific setup situations.

In Constraints (3.4), the tardiness ρ_i of job i is calculated. The value of the tardiness is penalized in the objective function.

In the industry, it is common to find two approaches: Scheduling jobs considering the capacity of each machine and scheduling jobs assuming infinity capacity of the machines. The model above can work perfectly with both approaches.

The makespan, in other words the specific date on which processing of all jobs is finished, is calculated by using Constraints (3.3).

We must also be sure that each job is assigned to just one machine, and this is expressed in equations (3.2). Finally, the integrality and nonnegative constraints concludes the model.

3.2.2 Second Model

In this approach the model used is based on a series of constraints proposed by Wagner (1959). The model tries to schedule a job in the specific position in which it will be processed by the machine, instead of considering the relation between each pair of jobs, as in the previous model.

The decision variables used in this model are listed below.

- $t_m^{(o)}$: processing start time of the o^{th} job on machine m,
- $\alpha_{im}^{(o)}$: $\begin{cases} 1 & \text{if job } i \text{ is processed by machine } m \text{ at the } o^{th} \text{ position,} \\ 0 & \text{otherwise;} \end{cases}$
- $\beta_{ii'm}^{(o)}$: $\begin{cases} 1 & \text{if jobs } i \text{ and } i' \text{ are processed by machine } m \text{ at the } o^{th} \text{ and } (o+1)^{th} \text{ positions} \\ & \text{respectively,} \\ 0 & \text{otherwise.} \end{cases}$
- ρ_i: tardiness of job i;

- Z: total makespan.

With the definition of the decision variables we are ready to state and analyze the model based on Wagner's job shop model. This model is presented below.

$$\min\left(Z+\sum_{i}^{N}(\rho_i\cdot\nu_i)\right) \tag{3.7}$$

$$\sum_{m}^{M}\sum_{o=1}^{|N|}\alpha_{im}^{(o)}=1, \qquad \forall i\in N \tag{3.8}$$

$$\sum_{i}^{N}\alpha_{im}^{(o)}\leq 1, \qquad \forall m\in M, o=1,\dots,|N| \tag{3.9}$$

$$\sum_{i}^{N}\alpha_{im}^{(o)}\leq\sum_{i}^{N}\alpha_{im}^{(o-1)}, \qquad \forall m\in M, o=2,\dots,|N| \tag{3.10}$$

$$d_i+\rho_i\geq t_m^{(o)}+p_{im}-\left(1-\alpha_{im}^{(o)}\right)\cdot G, \qquad \forall m\in M,\forall i\in N, o=1,\dots,|N| \tag{3.11}$$

$$Z\geq t_m^{(|N|)}+\sum_{i}^{N}\left(\alpha_{im}^{(|N|)}\cdot p_{im}\right), \qquad \forall m\in M \tag{3.12}$$

$$t_m^{(1)}=0, \qquad \forall m\in M \tag{3.13}$$

$$\beta_{ii'm}^{(o-1)}\geq 1-\left(2-\alpha_{im}^{(o-1)}-\alpha_{i'm}^{(o)}\right)\cdot G, \qquad \forall m\in M,\forall i,i'\in N, o=2,\dots,|N| \tag{3.14}$$

$$t_m^{(o)}\geq t_m^{(o-1)}+\sum_{i}^{N}\left(\alpha_{im}^{(o-1)}\cdot p_{im}\right)+\sum_{i}^{N}\sum_{i'}^{N-\{i\}}\left(\beta_{ii'm}^{(o-1)}\cdot s_{ii'm}\right), \qquad \forall m\in M, o=2,\dots,|N| \tag{3.15}$$

$$\begin{aligned}
&\alpha_{im}^{(o)},\beta_{ii'm}^{(o)}\in\{0,1\}, && \forall i\in N, i'\in N, m\in M, o=1,\dots,|N|\\
&\rho_i\geq 0, && \forall i\in N\\
&t_m^{(o)}\geq 0, && \forall m\in M, o=2,\dots,|N|\\
&Z\geq 0
\end{aligned}$$

In this case, the model tries to associate a certain job to a processing position of a machine, instead of analyze a reference position between any pair of jobs.

The objective function (3.7) is similar to the first model, it minimizes the makespan added to the weighted delays. Each job is processed exactly once (3.8), and there is at most one job at each position on each machine (3.9).

There must be a job at position $o-1$ if there is another job at position o on the same machine for $o\geq 2$ (3.10). The processing start time of the job at the o^{th} position on machine m added to the job's processing time must be less than or equal to the job's due date added to a possible delay ρ_i (3.11).

The makespan is greater than or equal to the processing start time for the job at the last position on each machine. Since the processing start time for each position is always greater than or equal to the processing start time for the previous position, this constraint determines the makespan (3.12). The processing start time for every job at the first position is equal to zero (3.13).

The setup time from job i to job i' is used between positions $o-1$ and o on machine m if jobs i and i' are on machine m and job i is at position $o-1$ and job i' is at position o (3.14).

The processing start time for the jobs at the other positions is equal to the processing start time of the job at the previous position added to its processing time on the machine and the setup time between the two positions (3.15), using the value from (3.14).

If $(\alpha_{im}^{(o-1)} = \alpha_{i'm}^{(o)} = 0)$, from Constraints (3.14), then $\beta_{ii'm}^{(o-1)} \geq 1-2\cdot G$, meaning that $\beta_{ii'm}^{(o-1)}$ might be 0 or 1. Since this is a minimization problem, Constraints (3.15) will force $\beta_{ii'm}^{(o-1)}$ to be equal to 0.

3.3 The Branch and Bound Algorithm

Algorithm 3.1: A generic Branch & Bound

```
Procedure: Branch
begin
    node
end
if IsACompleteSolution(node) then
    $UB \leftarrow$ SolutionValue(node);
    return
end
forall $n \in$ ChildrenOf(node) do
    if $LB(n) < UB$ then
        Branch(n);
    end
end
$UB =$ InitialSolution();
Branch(firstNode);
return UB;
```

A B&B is a specific enumeration tree strategy. In a B&B, there are three main procedures: *initialization*, *branching* and *bounding*. During the *initialization*, a fast heuristic is used to find a good initial solution, which serves as an upper bound (*UB*).

Branching divides the problem into smaller subproblems. Each subproblem represents a partial solution and is represented by a node in the tree. A search strategy must be associated with the branching to decide which node should be branched next. The *UB* helps to prune nodes from the search tree that have a lower bound (*LB*) greater than *UB* (minimization problem).

The *bounding* procedure calculates the *LB* for each node in order to decide which node should be pruned and which should be branched next. The framework for a generic Branch and Bound is shown in Algorithm 3.1. In the following sections, the three procedures are customized for this scheduling problem and described in more detail.

Algorithm 3.2: Main block of the B&B and the initialization algorithm(GRASP)

```
$bestSol$; %$Value of best solution so far$;
$localSol$; %$Value of local solution$;
GRASP($iterations$); Branch1(1); return $bestSol$;
 Procedure: GRASP
begin
    $iterations$
end
$jobs \leftarrow$ Order($N$);
for $i \leftarrow 1, iterations$ do
    $jobsRand \leftarrow$ ReOrder($jobs$);
    BuildSol($jobsRand$);
    LocalSearch();
    PathRelinking();
    if $localSol < bestSol$ then
        $bestSol \leftarrow localSol$;
    end
end
```

3.3.1 Initialization Based on GRASP

During the initialization, a complete initial solution is found to serve as an *UB*. Any node in the enumeration tree with a *LB* greater than *UB* can be pruned. In this work, an algorithm based on the metaheuristic GRASP is used to find this first solution. This metaheuristic consists of two phases: the construction of a feasible solution and a local search. These two phases are repeated for each iteration. The effectiveness of GRASP as an upper bound has been previously studied (Rocha et al., 2004). A description is available in Algorithm 2.3.

Initially, the jobs are ordered by the earliest due date rule (*EDD*). During the construction phase, the algorithm randomly reorders the array by means of a specific probability distribution. In this chapter, the probability function is $f(x) \sim 1/x$, where $f(x)$ represents the probability for the x^{th} job to be chosen next. Finally, a greedy function is used to schedule the jobs. The local search, arbitrarily chosen, swaps all the existing pairs of jobs assigned to different machines. If a swap improves the solution, the new solution is stored and the old one is abandoned. Nothing is done otherwise.

Path relinking (PR) is also used at the end of each iteration to intensify the local search. There are several ways to implement this technique. For this application, PR works as follows: first, a pool of good solutions is retained. Every time PR is used, a solution is randomly chosen from this pool and all solutions in the path from the solution found in the local search to the selected solution from the pool are analyzed. If a better solution is found, it is added to the pool.

3.3.2 *Branching*

The branching procedure develops the enumeration tree. The branching scheme is divided into 2 phases. In the first phase, the jobs are assigned to the machines, but no processing sequence is defined. The job sequence on each machine is decided in the second phase.

The first branching scheme starts ordering the jobs by the EDD rule. The jobs are assigned first to the machine capable of finishing each one earliest, then on the next machine, and so on. Each time a job is assigned to a machine, a new node in the enumeration tree is created. The scheme is described in Algorithm 3.3.

The second branching scheme decides the processing order on each machine. The order is decided for each machine at a time. On each machine, the first job to be processed is chosen, then the second, until there are no more jobs assigned to that machine. Each time a job is assigned to a specific position on a machine, a new node in the enumeration tree is created. This scheme is described in Algorithm 3.4 and in Figure 3.1.

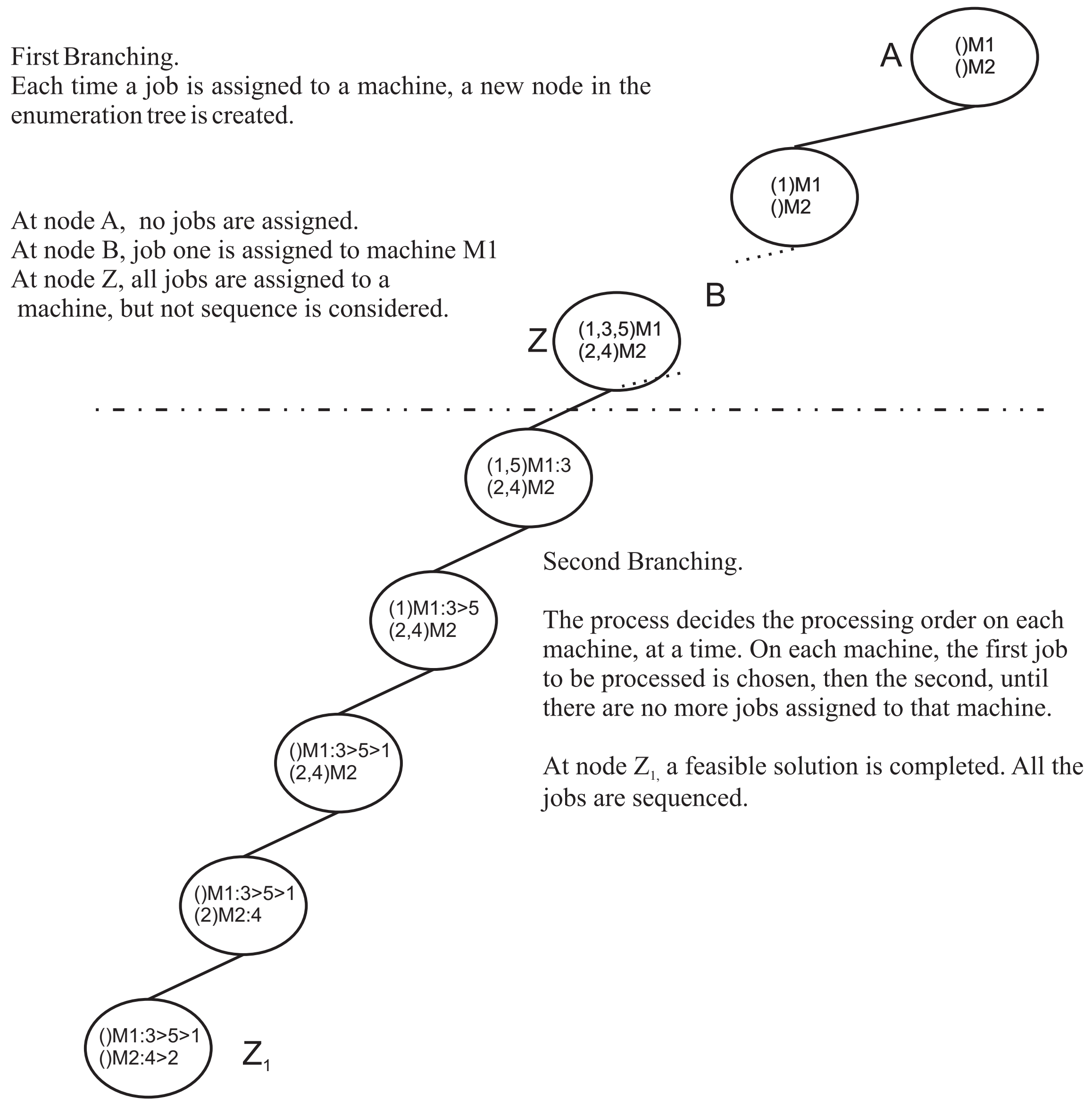
To avoid generating the complete enumeration tree, a bounding procedure is used to find a LB for each node. For a certain node i, if LB(i) is worse than UB, the node is pruned. The bounding procedure is described in the next section.

Algorithm 3.3: First branching scheme

```
Procedure: Branch1
begin
    $i$
end
if $AllJobsAlocated()$ then
    Branch2(1, 1);
    return
end
$job \leftarrow jobs[i]$;
for $m \leftarrow$ NextMachine() do
    AlocateJob($job, m$);
    if $LB(presentSol) < bestSol$ then
        Branch1($i + 1$);
    end
    DealocateJob($job, m$);
end
```

3.3.3 *Bounding*

The bounding procedure is probably the most important procedure on a B&B. A tight LB allows the algorithm to prune a great number of nodes, eliminating a lot of unnecessary processing. In this work, the bounding procedure calculates separately the LB for the makespan (LB^k) and for the weighted tardiness (LB^t) on each node. To calculate the LB^k in the first branching scheme, the procedure considers two forms, but only the largest value is used, as shown in Equation (3.16).


First Branching.
Each time a job is assigned to a machine, a new node in the enumeration tree is created.

At node A, no jobs are assigned.
At node B, job one is assigned to machine M1
At node Z, all jobs are assigned to a machine, but not sequence is considered.

Second Branching.

The process decides the processing order on each machine, at a time. On each machine, the first job to be processed is chosen, then the second, until there are no more jobs assigned to that machine.

At node $Z_{1,}$ a feasible solution is completed. All the jobs are sequenced.

Figure 3.1: First and Second branching scheme.

- First, it adds the smallest processing times of all unassigned jobs and the processing times of all assigned jobs to n times the smallest setup time u, where n is the smallest number of setups needed for all machines (i.e. number of jobs minus the number of machines). This gives LB^{kt}.
- Next, it adds the processing times of each job assigned to machine m to the n' smallest setup times among them, where n' is the smallest number of setup times necessary for m (i.e. number of jobs assigned to m minus one). This gives LB^{ka1}_m.

LB^k is formally defined in Equations (3.16-3.18), where N^u is the set of unassigned jobs, N^a_m is the set of jobs assigned to machine m and S^a_m is the set of setup times among the jobs from N^a_m.

$$LB^k = \max\left(\frac{LB^{kt}}{|M|}, \max_m^M \left(LB^{ka1}_m\right)\right) \tag{3.16}$$

$$LB^{kt} = \sum_i^{N^u} \min_m^M (p_{im}) + \sum_m^M \sum_i^{N^a_m} p_{im} + n \cdot u \tag{3.17}$$

$$LB^{ka1}_m = \sum_i^{N^a_m} (p_{im}) + \sum_{\{ii'm\}}^{S^a_m} (s_{ii'm}) \tag{3.18}$$

During the second branching scheme, since there are no unassigned jobs, Equations (3.19-3.20) are used to calculate the LB^k. The procedure adds the partial makespan (the makespan considering only the scheduled positions) to the processing times of the n unscheduled jobs and the n smallest setup times among them. This is done for each machine, and the larger value found gives the LB^k. Considering N^{as}_m as the set of jobs on machine m scheduled to a specific position and N^{au}_m as the set of jobs on machine m that have not been scheduled to a position in the processing order, in Equations

Algorithm 3.4: Second branching scheme

```
Procedure: Branch2($i, m$)
begin
    if SolutionIsComplete() then
        $bestSol \leftarrow presentSol$;
        return
    end
    if OrderIsDecided(m) then
        Branch2($1, m + 1$);
        return
    end
    for $job \leftarrow$ NextJob(m) do
        AlocateJobInPosition($job, m, i$);
        if LB(presentSol) $<$ bestSol then
            Branch2($i + 1, m$);
        end
    end
end
```

(3.19-3.20), S_m^{as} is the set of setup times used by the jobs in N_m^{as} and S_m^{au} is the set of the n smallest setup times among the jobs in N_m^{au} and the last job added to N_m^{as}, where n is the number of necessary setups for the jobs in N_m^{au}.

$$LB^k = \max_m^M \left(LB_m^{ka2}\right) \tag{3.19}$$

$$LB_m^{ka2} = \sum_i^{N_m^a} (p_{im}) + \sum_{\{ii'm\}}^{S_m^{as}} (s_{ii'm}) + \sum_{\{ii'm\}}^{S_m^{au}} (s_{ii'm}) \tag{3.20}$$

The lower bound for the weighted tardiness (LB^t) is calculated in a similar way to the LB^k. It is given according to Equations (3.21-3.26), where LB^{tu} considers only the unassigned jobs and LB_m^{ta} considers only the jobs assigned to machine m.

$$LB^t = LB^{tu} + \sum_m^M LB_m^{ta} \tag{3.21}$$

$$LB^{tu} = \sum_i^{N^u} \left[w_i \cdot \max\left(\min_m^M(p_{im}) - d_i, 0\right)\right] \tag{3.22}$$

$$LB_m^{ta} = LB_m^{ta1} + \max\left(LB_m^{ta2}, LB_m^{ta3}\right) \tag{3.23}$$

$$LB_m^{ta1} = \sum_i^{N_m^{as}} \left[w_i \cdot \max\left(t_i + p_{im} - d_i, 0\right)\right] \tag{3.24}$$

$$LB_m^{ta2} = \sum_i^{N_m^{au}} \left[w_i \cdot \max\left(t'_i + p_{im} - d_i, 0\right)\right] \tag{3.25}$$

$$LB_m^{ta3} = \min_i^{N_m^{au}} (w_i) \cdot \sum_i^{N_m^a} \max\left(t''_i + p_{im} - d_i, 0\right) \tag{3.26}$$

LB_m^{ta1} represents the tardiness penalty for the scheduled jobs and t_i is the start time of job i. The lower bound for the unscheduled jobs is calculated in two different ways (LB_m^{ta2} and LB_m^{ta3}), and only the largest value found is used. LB_m^{ta2} considers as if all jobs could be processed in the next available position, and t'_i is the smallest start time for job i. LB_m^{ta3} considers if the jobs were processed in a nondecreasing order using the due date and processing time in that order as sorting rule and a fixed setup time between each job. This setup time is equal to the smallest setup time considering the unscheduled jobs assigned to machine m and is used to calculate t''_i, the start time of job i according to this sequence.

It is important to point out a few differences between these variables during the first and second branching scheme. In the first branching scheme, t'_i is always zero, since there are no jobs scheduled at any position. In the second branching scheme, N^u is always empty, because all jobs are assigned to some machine. Therefore, LB^{tu} is always zero. Even though LB^{tu} will only be greater than zero when a job's processing time is larger than its due date, little variations in the LB can mean a great number of pruned nodes. Therefore, this formula is maintained.

The proposed lower bounds are weak bounds especially when the setup times can vary in a large range. One last detail is that no lower bound formula considers the job delays, make them even more weak when tight due dates are considered. This fact is also discuss in Chapter 4 when a method is proposed to improved these lower bound formulas.

3.4 Instances

To analyze the algorithms and models developed for this problem, several classes of instances are defined. In each class there is a change in one of the inputs. All values are randomly generated from an uniform distribution. Their maximal and minimal values are listed in Table 3.1, where $U(x, y)$ is a value generated from an uniform distribution between x and y.

Table 3.1: Standard values of the instances

Input data	Standard value
Processing time	$U(5, 200)$
Setup time	$U(25, 50)$
weight or Priority	$U(1, 3)$
Due date	$U\left(\text{maximal processing time}, \frac{2 \cdot h}{q}\right)$
q	1

To generate the due dates, several variations of formulas found in other works, like (Pereira-Lopes and de Carvalho, 2006) or (Ho and Chang, 1995) were tested. To ensure that the ratio $\frac{\text{weighted tardiness}}{\text{total solution}}$ stays the same in all sizes of instances of each class, the following algorithm to calculate the maximal value for the due date was adopted: in the order of generation, each job is assigned to the machine capable of finishing it first. The makespan of this solution is referred to as h. The maximum value for the due date is given by $\frac{2 \cdot h}{q}$, where q indicates the congestion level of the scheduling system (Pereira-Lopes and de Carvalho, 2006). The larger the q, the more congested the system will be, and the more tardy jobs will be.

The created classes are:

A: Contains standard values for all variables.
They are defined in Table 3.1.

B: The due dates are decreased.
$q = 2$.

C: The due dates are decreased.
$q = 3$.

D: The due dates are decreased.
$q = 4$.

E: The due dates are decreased.
$q = 5$.

F: The processing time is slightly decreased.
$p = U(5, 150)$.

G: The processing time is greatly decreased.
$p = U(5, 100)$.

H: The setup time is slightly increased.
$s = U(25, 100)$.

I: The setup time is greatly increased.
$s = U(25, 150)$.

Algorithm 3.5: Modified Floyd-Warshall

```
S;% Matrix with the setup times of a specific machine;
P;% Vector with the processing times of a specific machine;

Procedure: Floyd-Warshal($S, P$)
begin
    $n \leftarrow S.rows$;
    for $k \leftarrow 1$ to $n$ do
        for $i \leftarrow 1$ to $n$ do
            for $j \leftarrow 1$ to $n$ do
                $s_{ij} \leftarrow \min(s_{ij}, s_{ik} + p_k + s_{kj})$;
            end
        end
    end
end
```

We generated instances with 4 machines and 4 to 12 jobs, and with 6 machines and 6 to 25 jobs. For each problem size, 40 instances were randomly generated using different seeds.

After generated, the instances were slightly modified. For the first MIP model and the B&B to be valid, the setup and processing times must satisfy the triangular inequality $s_{ij} \leq s_{ik} + p_k + s_{kj}$. In some production lines, specially in chemical industry, the processing of job k is part of the setup from job i to j. With this in mind, the processing time of job k was included in the triangular inequality.

Since the setup times were generated randomly, they needed to be corrected to satisfy the inequality. In order to do that, the Floyd-Warshall algorithm (F-W) was modified as shown in Algorithm 3.5 to include a weight in each node to be considered as well as the arc's weight. The original F-W is used to find the shortest path between all pairs of nodes in a directed graph (Cormen et al., 1990). All instance files can be found at the *UFMG Scheduling Group* home page [1].

[1]http://www.dcc.ufmg.br/lapo/wiki/index.php/Scheduling_Team

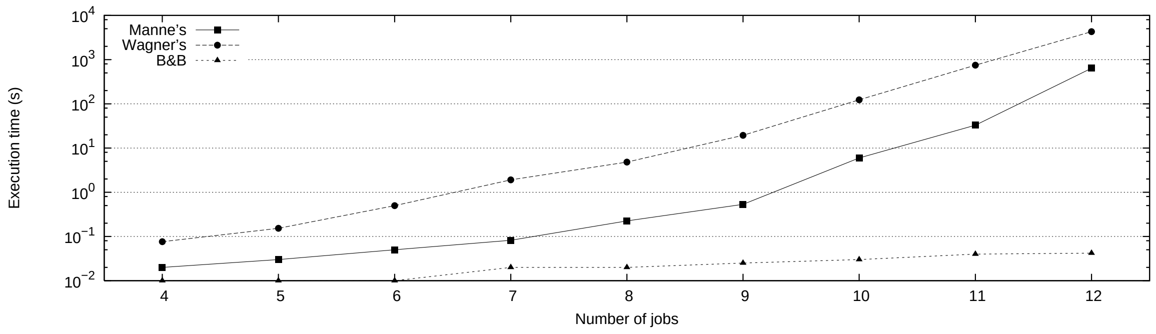


Figure 3.2: MIP vs. B&B

Table 3.2: Spent CPU time and variation by the two MIP models and the B&B

Jobs	MIP based on Manne's			MIP based on Wagner's			Branch and Bound		
	Avg. ± Dev. (%)	Min.	Max.	Avg. ± Dev. (%)	Min.	Max.	Avg. ± Dev. (%)	Min.	Max.
4	0.02 ± 00.00	0.02	0.02	0.08 ± 04.24	0.06	0.09	0.01 ± 00.00	0.01	0.01
5	0.03 ± 00.00	0.03	0.03	0.15 ± 06.67	0.12	0.22	0.01 ± 00.00	0.01	0.01
6	0.05 ± 04.90	0.04	0.06	0.50 ± 08.58	0.35	0.72	0.01 ± 00.00	0.01	0.01
7	0.08 ± 05.67	0.06	0.10	1.91 ± 07.06	1.15	2.64	0.02 ± 00.00	0.02	0.02
8	0.23 ± 13.35	0.14	0.47	4.82 ± 09.11	3.08	7.11	0.02 ± 00.00	0.02	0.02
9	0.53 ± 11.54	0.32	0.95	19.40 ± 10.66	10.18	30.58	0.03 ± 06.32	0.02	0.03
10	5.97 ± 22.99	1.10	15.17	123.55 ± 10.36	80.44	222.78	0.03 ± 00.00	0.03	0.03
11	32.93 ± 35.83	3.97	130.89	747.79 ± 19.30	250.04	1,366.97	0.04 ± 00.00	0.04	0.04
12	637.60 ± 54.57	25.61	3,825.17	4,297.73 ± 10.72	2,091.25	6,706.19	0.04 ± 03.00	0.04	0.05

3.5 Tests and Results

We present results comparing the solutions found by the developed B&B to the solutions found by both models when solved by CPLEX 9.0[2] in Section 3.5.1. Tests with the B&B solving the various classes described in Section 3.4 are shown in the following sections.

We use an interquartile mean for all tests, discarding 50% of the values found. Since there are 20 instances of each class, the 5 smallest and the 5 greatest values found for each class are discarded from the sample. The interquartile mean is less sensitive to outliers than the mean, but uses more information from the sample than the median.

During the tests, we tend to vary only one job from instance to instance. But, a problem with n jobs is not always easier to solve than with $n+1$ jobs. Depending on the number of machines and how the jobs are spread across them, the later may be a lot easier than the former. In the graphics we are able to see some disturbances like this.

GRASP was fixed at 1,000 iterations to provide the upper bound, and all graphics are shown in logarithmic form.

3.5.1 Comparing the B&B and the MIP Models

In this test we compare the CPU time spent the by two MIP models and the B&B to find the optimal solution. Both models were solved by CPLEX, and were given the same starting solution that was

[2] *CPLEX Software: an optimizer for linear problems developed by ILOG. More information at their website: http://www.ilog.com/products/cplex/*

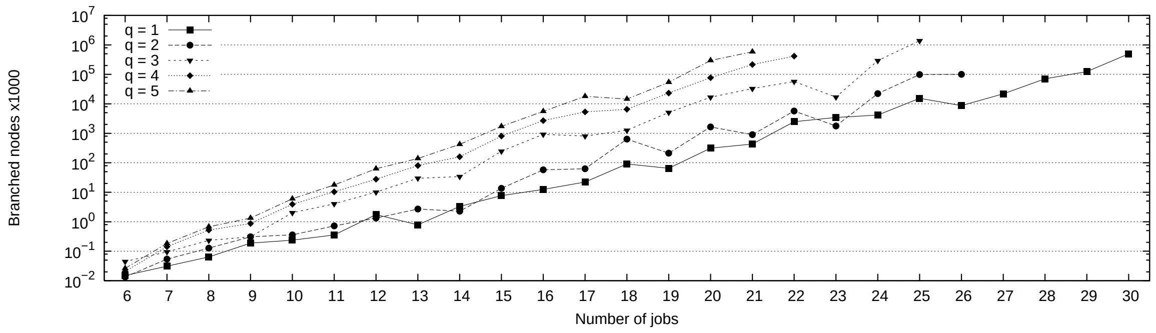


Figure 3.3: Varying the due dates

found by GRASP and given to the B&B. Instances of class A for 4 machines were used in this test. Results are shown in Figure 3.2 and detailed in Table 3.2.

The first model performs close to the B&B in the smaller instances. As the number of jobs grows, so does the difference between them. The deviations of the first and second models also increase as the number of jobs grows, but the deviation of the B&B varies without an apparent rule. The second model performs worst than the first and the B&B, but it presents itself more stable. It has a smaller deviation than the first model in the larger instances. Its spent CPU time also grows at a lower rate than the first model. This leads us to believe it will perform faster than the first model in larger instances, but we were unable to test it due to time constraints. The B&B reached the same optimal solutions obtained by the MIP models.

3.5.2 Due Dates

In this test we try to evaluate how the variation of due dates influence in the execution of the B&B. Instances of classes A through E for 6 machines were used in this test. Results are shown in Figure 3.3 and detailed in Table 3.3.

Instances of class E (very tight due dates) are the hardest to be solved by the algorithm, followed by class D (tight due dates). Classes A through C present nearly the same difficulty. This shows that as the due dates decrease, the difficulty for the B&B increases. One may think the opposite should occur, since that as the due dates decreases, so do the number of interesting solutions, causing the search universe to decrease. But when taking a look at the bounding phase of the B&B described in Section 3.3.3, it is possible to understand why this does not happen. To calculate the lower bound, the bounding procedure considers the unscheduled jobs as either being next on the schedule or as all of them having the lowest priority and the lowest setup time among them. This formula may not give strong lower bounds when the delay penalty is too great, but guarantees the solution optimality.

3.5.3 Processing Time

Here we try to evaluate how the variation of the processing times influence the execution of the B&B. Instances of classes A, F and G are used. Results are show in Figure 3.4 and detailed in Table 3.4.

Instances with large processing times were the easiest to be solved by the B&B. We believe this happens because the search space of interesting solutions is decreased when the difference of the

Table 3.3: Number of branched nodes and standard deviation varying the due dates with the B&B

Jobs	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$
6	15.10 ± 43.77	13.40 ± 22.91	44.70 ± 50.00	20.60 ± 16.46	26.10 ± 14.94
7	31.00 ± 27.81	53.70 ± 33.89	95.50 ± 37.47	142.40 ± 39.12	183.10 ± 36.54
8	63.90 ± 28.92	126.90 ± 24.87	232.60 ± 20.64	523.00 ± 25.30	672.00 ± 23.67
9	188.90 ± 37.37	308.80 ± 32.79	307.70 ± 18.08	871.40 ± 24.98	1,327.20 ± 23.90
10	236.40 ± 27.99	359.30 ± 25.08	2,016.00 ± 25.10	3,922.60 ± 18.97	6,019.90 ± 13.36
11	357.20 ± 19.23	722.20 ± 22.11	4,059.40 ± 21.90	10,313.10 ± 19.12	17,738.50 ± 18.02
12	1,752.20 ± 36.07	1,337.30 ± 18.64	10,061.30 ± 24.85	27,843.80 ± 21.27	62,966.60 ± 28.44
13	783.80 ± 28.39	2,693.40 ± 14.96	29,887.70 ± 21.62	80,541.60 ± 20.76	139,322.90 ± 17.70
14	3,299.50 ± 15.20	2,272.10 ± 12.70	34,118.40 ± 35.83	160,033.40 ± 27.31	423,668.50 ± 15.32
15	7,711.60 ± 15.88	13,565.80 ± 33.00	246,870.70 ± 32.84	805,843.50 ± 27.61	1,718,413.80 ± 16.59
16	12,510.40 ± 13.06	57,540.40 ± 33.58	923,280.20 ± 30.60	2,708,598.30 ± 26.25	5,573,765.90 ± 20.50
17	22,271.00 ± 25.73	62,561.10 ± 18.87	805,266.80 ± 13.45	5,347,459.60 ± 18.40	17,987,409.50 ± 31.21
18	91,347.20 ± 16.67	632,824.20 ± 36.04	1,264,338.90 ± 21.05	6,512,628.90 ± 18.04	14,503,974.00 ± 31.86
19	64,742.90 ± 21.77	212,173.00 ± 35.83	5,110,468.30 ± 39.54	23,166,388.70 ± 57.28	54,040,469.80 ± 25.83
20	314,864.90 ± 19.65	1,647,258.50 ± 37.72	16,784,999.80 ± 12.85	76,735,050.60 ± 15.57	294,607,021.60 ± 23.35
21	433,137.00 ± 12.46	902,952.20 ± 29.87	32,806,058.90 ± 37.08	215,955,785.00 ± 38.31	585,073,708.80 ± 31.67
22	2,484,061.60 ± 37.60	5,711,464.70 ± 27.28	57,002,791.50 ± 23.09	414,051,018.30 ± 38.34	
23	3,441,076.80 ± 16.90	1,779,265.80 ± 20.22	16,912,864.00 ± 20.03		
24	4,173,514.90 ± 24.58	22,185,304.50 ± 24.51	292,227,152.00 ± 31.16		
25	15,413,490.80 ± 19.07	98,369,868.90 ± 31.47	1,387,958,704.60 ± 15.12		

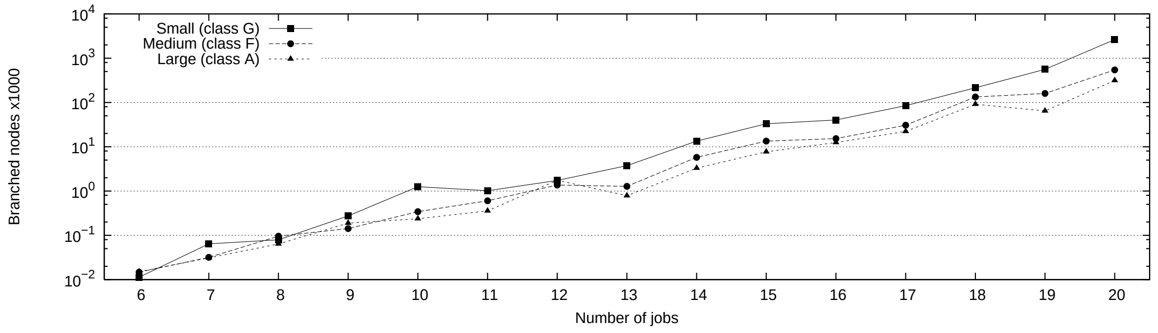


Figure 3.4: Varying the processing times

Table 3.4: Number of branched nodes and standard deviation varying the processing times with the B&B

Jobs	Small	Medium	Large
6	11.30 ± 43.98	14.70 ± 40.00	15.10 ± 43.77
7	64.30 ± 30.30	31.80 ± 24.53	31.00 ± 27.81
8	79.30 ± 25.03	95.30 ± 31.80	63.90 ± 28.92
9	274.90 ± 24.01	141.90 ± 27.89	188.90 ± 37.37
10	1,252.20 ± 50.11	341.80 ± 27.90	236.40 ± 27.99
11	1,014.30 ± 17.71	603.30 ± 18.34	357.20 ± 19.23
12	1,735.30 ± 20.44	1,366.90 ± 31.50	1,752.20 ± 36.07
13	3,742.20 ± 12.56	1,275.00 ± 18.67	783.80 ± 28.39
14	13,456.20 ± 18.45	5,783.90 ± 11.49	3,299.50 ± 15.20
15	33,117.60 ± 17.76	13,402.70 ± 15.00	7,711.60 ± 15.88
16	40,263.50 ± 12.73	15,270.40 ± 13.18	12,510.40 ± 13.06
17	84,750.90 ± 13.55	30,694.90 ± 17.17	22,271.00 ± 25.73
18	215,261.90 ± 27.03	133,605.40 ± 33.81	91,347.20 ± 16.67
19	560,559.90 ± 14.01	160,188.50 ± 21.41	64,742.90 ± 21.77
20	2,640,069.60 ± 18.36	545,886.50 ± 13.38	314,864.90 ± 19.65

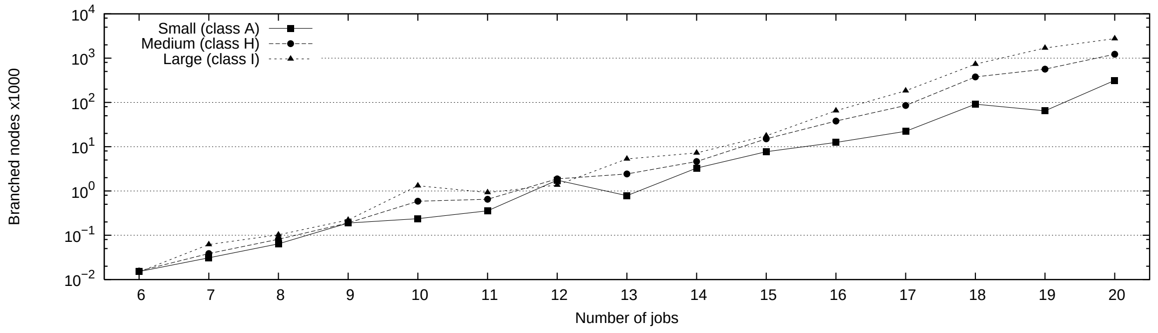


Figure 3.5: Varying the setup times

processing time of a job over two machines may be too great. When a job is scheduled on a machine where its processing time is greater than on other machines, the lower bound increases, and if it gets greater than the upper bound, the node is pruned. It is easy to realize that the greater the processing time, the quicker the lower bound will increase, and the quicker the node will be pruned.

Table 3.5: Number of branched nodes and standard deviation varying the setup times with the B&B

Jobs	Small	Medium	Large
6	15.10 ± 43.77	15.60 ± 37.76	14.90 ± 37.58
7	31.00 ± 27.81	38.50 ± 31.12	61.90 ± 36.20
8	63.90 ± 28.92	80.90 ± 24.87	102.40 ± 24.81
9	188.90 ± 37.37	190.00 ± 32.38	221.30 ± 29.60
10	236.40 ± 27.99	586.90 ± 20.83	1,309.30 ± 27.06
11	357.20 ± 19.23	648.30 ± 20.20	922.10 ± 17.80
12	1,752.20 ± 36.07	1,879.90 ± 27.26	1,359.90 ± 17.34
13	783.80 ± 28.39	2,427.50 ± 29.97	5,299.10 ± 15.08
14	3,299.50 ± 15.20	4,620.60 ± 18.01	7,256.60 ± 13.24
15	7,711.60 ± 15.88	15,045.40 ± 18.26	17,668.60 ± 12.33
16	12,510.40 ± 13.06	37,956.70 ± 19.86	64,910.70 ± 10.67
17	22,271.00 ± 25.73	85,211.10 ± 21.77	183,705.90 ± 18.64
18	91,347.20 ± 16.67	376,655.90 ± 19.12	733,566.90 ± 28.43
19	64,742.90 ± 21.77	563,697.60 ± 11.78	1,699,133.80 ± 20.97
20	314,864.90 ± 19.65	1,223,759.50 ± 14.87	2,766,689.50 ± 11.22

3.5.4 Setup Time

In this test we try to evaluate how the variation of the setup times influence the execution of the B&B. Instances of classes A, F and G are used. Results are show in Figure 3.5 and detailed in Table 3.5.

Instances with small setup times were the easiest to be solved by the B&B. Here we can see a weak lower bound greatly harming the performance of the algorithm. As explained in Section 3.3.3, the bounding procedure considers only the smallest setup times when calculating the lower bound. When the setup times may vary in a great interval, the lower bound is weakend, harming the performance of the B&B.

3.5.5 Larger Instances

We are able to find the optimal solution for problems up to 30 jobs with the B&B. For larger problems, we limit the B&B resolution time to 7,200 minutes (two hours), and measure the improvement on the solution provided by GRASP. Since the B&B does not prove optimality if the time limit is exceeded, its solution is compared to the solution found by GRASP after 20,000 iterations (GRASP2) to evaluate if GRASP with a greater number of iterations is a better approach. The average improvement on the GRASP solution after 1,000 it. given by Formula (3.27), as well as the average distance between the B&B and the GRASP2 solution given by Formula (3.28) are shown in Table 3.6.

$$\frac{\text{GRASP sol.} - \text{B\&B sol.}}{\text{GRASP sol.}} \tag{3.27}$$

$$\frac{\text{B\&B sol.} - \text{GRASP2 sol.}}{\text{B\&B sol.}} \tag{3.28}$$

With 30 jobs, the B&B improves the solution found by GRASP after 1,000 it. by an average of 9.8%, and the distance from the B&B solution to the GRASP2 solution is −2.31% in average. A negative number means the B&B solution is in average better than the GRASP2 solution. Since the B&B found and proved the optimal solutions for the instances up to 30 jobs, a negative distance was expected in this case. But as the problem size increases, the GRASP2 solution becomes more

Table 3.6: Comparing the B&B and GRASP performance on larger instances

Jobs	Improvement	Distance
30	9.80 %	-2.31 %
40	2.02 %	1.69 %
50	0.25 %	2.93 %
60	0.26 %	2.64 %
70	0.00 %	3.31 %
80	0.07 %	3.10 %
90	0.21 %	2.20 %
100	0.09 %	2.19 %

interesting and the improvement the B&B makes on the GRASP solution decreases. With 70 jobs, the B&B is unable to improve the GRASP solution in any of the 20 instances.

As the number of jobs increases, the B&B loses efficiency since it must solve a TSP (Traveling Salesman Problem) for each machine during the second branchind scheme. In theses situations, the solution provided by GRASP should be more interesting.

3.6 Concluding remarks and future research

In this chapter, we considered a scheduling problem with unrelated parallel machines, due dates and setup times that depend both on the machine and the sequence. A B&B algorithm using GRASP as an initialization procedure, and two MIP models were proposed and tested. Also as a contribution, we generated a set of instances with different values of due dates, setups and processing times.

Concerning the performance of the algorithms and models, we solved instances with up to 25 jobs. We showed that the B&B has a much better performance than the two MIP models solved by CPLEX 9.0. An increase in the variation of the setup times may increase the number of nodes branched by the B&B, whereas an increase in the due dates or processing times may decrease the number of nodes in the enumeration tree.

Future research studies include a comparison between GRASP and the B&B involving larger instances, the improvement of the way B&B takes advantage of the solutions provided by GRASP, the development of a dichotomic B&B and the analysis of its performance when considering identical machines. A greater cooperation among the solving methods and a parallel version of the B&B may be also considered.

Chapter 4

Lower Bounds

In the literature, the optimization of the makespan and the sum of weighted delays are often considered as two different problems, but in this work, they are considered together due to the fact that this problem is inspired on a real case and the company requires that. As presented in chapter 3, the lower bounds used in this work are computed using logic and construction functions. The aim of this chapter is to proposed a time-indexed formulation for the parallel machine scheduling problem and an algorithm based on Lagrangian relaxation to improve the problem's lower bounds.

As it is possible to see in Section 3.3, the proposed branch-and-bound algorithm works using a depth-first strategy. This approach allows the algorithm to find better solutions very fast, but at the same time the lower bound is not improved. The CPU time spent in the whole search has a strong dependence on the initial lower bound, for that reason, to obtain better lower bounds could improve the CPU time spent by the B&B algorithm.

The chapter is organized as follows: In Section 4.1 the time-index model for the original problem is presented. In Section 4.2 the relaxed model and the proposed algorithm are presented and tested for small instances.

4.1 Time-Indexed Formulation

Time-indexed formulations are used in scheduling problems because its linear relaxation provides stronger lower bounds than other models, as well as, they are useful to guide approximation algorithms, (Dyer and Wolsey, 1990; Queyranne and Schulz, 1994). On the other hand, the main drawback of these models is their size. For scheduling problems these formulations were initially proposed by Dyer and Wolsey (1990), for a single-machine case, and have been studied by many authors, (Sousa and Wolsey, 1992; Queyranne and Schulz, 1994; van den Akker et al., 1998, 1999; Savelsbergh et al., 1998). To the best of our knowledge, we do not know any work proposing a time-indexed model considering unrelated parallel machines and sequence dependent setup times.

The time-indexed formulation is based on the division of time into T periods. T represents the planning horizon, and a period t starts at t and ends at $t+1$. Considering a practical point of view, this time-discretization is made by using the minimum fraction of time considered in the process. For example in the real case, company Z works with 30 minutes as time unit for its Gantt chart, and this discretization can be used in the model.

4.1.1 Parameters and variables

The parameters are almost the same than in the previous models. The main point is setting a planning horizon T, by fixing this parameter, the consideration of infinity variables is avoided.

`Parameters`

N	number of jobs to be processed.
M	number of machines.
p_{im}	Processing time of job i by machine m.
$s_{ii'm}$	Time for setting up machine m, from processing job i to processing job i'.
$\tau_{ii'm}$	Time for setting processing job i plus setting up machine m, from processing job i to processing job i'. ($\tau_{ii'm} = p_{im} + s_{ii'm}$)
d_i	Due date of job i.
ν_i	Tardiness penalty coefficient of job i.
T	Planning horizon.

`Decision Variables`

x_{im}^t	$x_{im}^t = 1$, if job i started at machine m in period t, $x_{im}^t = 0$ otherwise,
C_i	Completion time of Job i .
Z	Global makespan.
ρ_i	Job i delay.

4.1.2 Mathematical models

In this first model, we are dealing with the same problem as in the previous chapters. The objective function remains the same, that is, the sum of the makespan plus weighted delays, Equations 4.1.

$$Min\, Z + \sum_{i=1}^{N} \nu_i \rho_i \tag{4.1}$$

Subject to

$$\sum_{m=1}^{M} \sum_{t=1}^{T-p_{im}} x_{im}^t = 1 \qquad \forall i \tag{4.2}$$

$$x_{im}^t + \sum_{s=t+1}^{t+\tau_{ijm}} x_{jm}^s \leq 1 \qquad \forall i \forall t (t \leq T - p_{im}) \forall m \forall j (i \neq j) \tag{4.3}$$

$$C_i \geq \sum_{m=1}^{M} \sum_{t=1}^{T-p_{im}} (t + p_{im}).x_{im}^t \qquad \forall i \tag{4.4}$$

$$\rho_i \geq C_i - d_i \qquad \forall i \tag{4.5}$$

$$Z \geq C_i \qquad \forall i \tag{4.6}$$

$$\rho_i \geq 0 \qquad \forall i \tag{4.7}$$

$$C_i \geq 0 \qquad \forall i \tag{4.8}$$

$$Z \geq 0 \tag{4.9}$$

$$x_{im}^t \in \{0,1\} \qquad \forall i \forall t \forall m \tag{4.10}$$

Constraints 4.2 ensure that each job is scheduled exactly once. Constraints 4.3 take care about the sequence dependent setup times, by no allowing the schedule of an extra job before the previous processing job plus the setup time (τ) are completed, Figure 4.1 describes the relation between consecutive jobs for a better understanding of the model and its constraints.

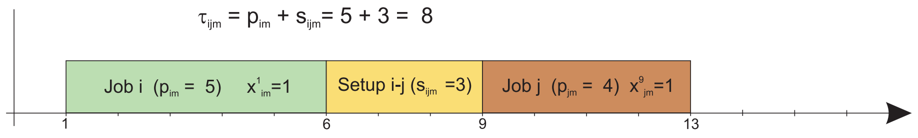

$\tau_{ijm} = p_{im} + s_{ijm} = 5 + 3 = 8$
Job i (p_{im} = 5) x^1_{im}=1
Setup i-j (s_{ijm} =3)
Job j (p_{jm} = 4) x^9_{jm}=1
1
6
9
13

Figure 4.1: Time-indexed variable example

Constraints 4.4 calculate the completion time for each job, and the inequalities 4.5 and 4.6, compute the job's delay and the makespan, respectively.

As it is possible to see, the allocation of a binary variable (e.g. x_{im}^t) implies in schedule a job (i) on a machine (m), beginning on a certain date (t), with these data we are able to compute its completion time and its delay, if exists. There are two major complicating factors in this model. First, the sequence dependent setup times and the number of constraints for this matter is huge. Second, to consider the makespan as part of the objective function. The makespan is a recursive function that makes impossible to known after fixing a variable, how this allocation is going to affect the objective function, unless all other variables are already fixed.

As previously stated, the aim of this chapter is to proposed an algorithm to improve the lower bound presented in Chapter 3. All our lower bounds consider only the makespan. For this reason, we are going to split our problem and find a lower bound to the weighted delays sum. The model is now reformulated, considering only the weighted delays as objective function. With this new model we clearly improve our linear relaxation bound; the model is presented below.

$$Min \sum_{i=1}^{N} \nu_i \rho_i \tag{4.11}$$

Subject to

$$\sum_{m=1}^{M}\sum_{t=1}^{T-p_{im}} x_{im}^t = 1 \qquad \forall i \qquad (4.12)$$

$$x_{im}^t + \sum_{s=t+1}^{t+\tau_{ijm}} x_{jm}^s \leq 1 \qquad \forall i, \forall t (t \leq T - p_{im}), \forall m, \forall j (i \neq j) \qquad (4.13)$$

$$\rho_i \geq \sum_{m=1}^{M}\sum_{t=1}^{T-p_{im}} (t + p_{im}).x_{im}^t - d_i \qquad \forall i \qquad (4.14)$$

$$\rho_i \geq 0 \qquad \forall i \qquad (4.15)$$

$$x_{im}^t \in \{0,1\} \qquad \forall i, \forall t, \forall m \qquad (4.16)$$

Considering this model and relaxing the group of Constraints 4.13, it is possible to deal with the problem of considering too many constraints. In next section, a Lagrangian relaxation is presented and an algorithm for improving the lower bound is proposed.

4.2 Lagrangian relaxation

4.2.1 Introduction

Lagrangian relaxation was carry out after the innovating work of Held and Karp (1970, 1971). The main idea behind the Lagrangian relaxation is to identify and remove a set of constraints that are considered difficult to solved and added them to the objective function penalized by a set of Lagrangian multipliers, (Geoffrion, 1974), (Fisher, 1981). Considering a problem (P) defined as,

$$(P) \qquad z = min_x\{c'x : Ax \leq b, Bx \leq c, x \in \mathbb{R}^n\}$$

Where $b, c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1,n}$, $B \in \mathbb{R}^{m_2,n}$. Assume that the problem, with the absence of constraints $Ax \leq b$, is easily solvable. The Lagrangian problem (P_λ) is obtained by relaxing the group of complicating constraints and add them to the objective function. The use of a non-negative penalty terms (Lagrangian multipliers, λ) for each constraint avoid their violation.

$$(P_\lambda) \qquad z'(\lambda) = min_x\{c'x + \lambda'(Ax - b) : Bx \leq c, x \in \mathbb{R}^n\}$$

It is easy to check that, for each value $\lambda \in \mathbb{R}_+^{m_1}$ the result of the dual problem P_λ is a lower of the original one (P), i.e., $z'(\lambda) \leq z$. The problem now is to find the best possible Lagrangian multipliers for the problem (P_λ), this problem is usually called Dual Problem or Lagrangian Dual (D).

$$(D) \qquad d = max_\lambda\{z'(\lambda) : \lambda \in \mathbb{R}_+^{m_1}\}$$

Since $z'(\lambda) \leq z \ \forall \lambda \in \mathbb{R}_+^{m_1}$, then, the problem D is a relaxation of the original problem P, $d \leq z$. This method has been used in several combinatorial problems and there are a vast number of papers with applications and new approaches, some of them are, (Held and Karp, 1970, 1971), (Geoffrion, 1974), (Fisher, 1981), (Potts, 1985), (Frangioni, 2005).

4.2.2 Relaxing the problem

Relaxing the second group of constraints (4.13) and considering $\lambda_{ijm}^t \geq 0 \, \forall i, \forall j (i \neq i), \forall t (t \leq T - p_{im})$, $\forall m$ we are able to obtain, the Langrangian Problem

$$P(\lambda) = \qquad Min \sum_{i=1}^{N} \nu_i \rho_i + \sum_{i=1}^{N} \sum_{j=1,(i \neq j)}^{N} \sum_{m=1}^{M} \sum_{t=1}^{T-p_{im}} \lambda_{ijm}^t \left(x_{im}^t + \sum_{s=t+1}^{t+\tau_{ijm}} x_{jm}^s - 1 \right) \tag{4.17}$$

Subject to

$$\sum_{m=1}^{M} \sum_{t=1}^{T-p_{im}} x_{im}^t = 1 \qquad \forall i \tag{4.18}$$

$$\rho_i \geq \sum_{m=1}^{M} \sum_{t=1}^{T-p_{im}} (t + p_{im}).x_{im}^t - d_i \qquad \forall i \tag{4.19}$$

$$\rho_i \geq 0 \qquad \forall i \tag{4.20}$$

$$x_{im}^t \in \{0, 1\} \qquad \forall i, \forall t, \forall m \tag{4.21}$$

Considering a $x_{im}^t = 1, \forall i, \forall t, \forall m$, it is possible to compute the completion time and delay for each x_{im}^t. Following that direction, it is possible to define and calculate a new cost β, for each possible allocation,

$$\beta_{im}^t = \nu_i (\max(0, t + p_{im} - d_i)) + \sum_{j=1,(i \neq j)}^{N} \left(\lambda_{ijm}^t + \sum_{s=t-\tau_{jim}}^{t} \lambda_{jim}^s \right) \tag{4.22}$$

With the new costs β, the problem to solved is the problem $P'(\beta)$, that is, decide where and when each job is going to be allocated (m and t).

$$P'(\beta) = \qquad Min \sum_{i=1}^{N} \sum_{m=1}^{M} \sum_{t=1}^{T-p_{im}} x_{im}^t \cdot \beta_{im}^t \tag{4.23}$$

$$\sum_{m=1}^{M} \sum_{t=1}^{T-p_{im}} x_{im}^t = 1 \qquad \forall i \tag{4.24}$$

$$x_{im}^t \in \{0, 1\} \qquad \forall i, \forall t, \forall m \tag{4.25}$$

$$\tag{4.26}$$

Once this new problem is solved, we can calculate the lower bound for these λ values by using equation (4.17). The problem now is to find the best possible values for the Lagrangian multipliers (λ), to obtain the best lower bound, Problem $P(\lambda)$. For our original problem the final lower bound will be the sum of the makespan plus the weighted delays lower bounds.

$$max_{\lambda} \{ P(\lambda) : \lambda_{ijm}^t \geq 0, \forall i, \forall j, \forall t, \forall m \} \tag{4.27}$$

4.2.3 Proposed Algorithm

It can be shown that $P(\lambda)$ is a piecewise linear function. Solving the dual problem is therefore a non-differentiable optimization problem. A successful and well known technique for this problem is the Subgradient Optimization, see (Fisher, 1981), (Beasley, 1993). Based on the Subgradient Optimization procedure Algorithm 4.1 shows the pseudo-code of our proposed algorithm.

Algorithm 4.1: Pseudo-code of the proposed Algorithm.

```
   Procedure: Lower_Bound()
1  begin
2      Initialize($\pi$, LB_best, $\theta$) $\leftarrow (2, LB\_best, 1.05)$;
3      $UB \leftarrow$ GRASP();
4      $\lambda \leftarrow 0^M$;
5      while $\pi \leq 0.0001$ do
6          $\beta \leftarrow$ costs($\lambda$);
7          $Inspect\_Solution \leftarrow$Inspection($\beta$);
8          $LB \leftarrow$ Compute_LB($Inspect\_Solution$);
9          if $LB > LB\_best$ then
10             $LB\_best \leftarrow LB$;
11             $non\_imp \leftarrow 0$;
12         else
13             $non\_imp = non\_imp + 1$;
14             if $non\_imp = 300$ then
15                 $\pi \leftarrow \pi/2$;
16                 $non\_imp = 0$;
17             end
18         end
19         $SG \leftarrow$Compute_SG($Inspect\_Solution$);
20         $Step \leftarrow \pi \cdot (\theta \cdot UB - LB)/||SG_i||^2$;
21         for all $\lambda^t_{ijm}$ do
22             $\lambda^t_{ijm} \leftarrow max(0, \lambda^t_{ijm} + Step \cdot SG^t_{ijm})$;
23         end
24     end
25     return $LB\_best$;
26 end
```

The proposed method works as follow. In line 2, the main parameters are initialized. In lines 3, the GRASP heuristic is called and the upper bound variable (UB) gets its best solution value. In line 4, the Lagrangian multipliers (λ) are set to zero. From lines 5 to 19, the iterative process take place. Initially the beta-costs (β) are compute using Equation 4.22, the relaxed problem ($\pi(\lambda)$) is solved, and a lower bound is compute using Equation 4.17, lines 5, 6 and 7. If necessary, the best lower bound is update. In line 15, the π factor is reduced after 300 iterations with no lower bound improvement.

The subgradient values (SG) are calculated in line 19, using the last solution of the relaxed problem. With the SG new values, a scalar step size is obtained, line 20. Finally, in line 22,the Lagrangian multipliers are updated. The algorithm returns the incumbent lower bound in line 25.

4.2.4 Preliminary Results

Preliminary tests are performed to evaluate the effectiveness of the method. For these tests we are using the solver CPLEX 9.0, with the callable library. The algorithm was coded in C, in a GNU/Debian environment. For this preliminary experiments we solved the Dual Problem using CPLEX 9.0, instead of using a specific algorithm for the relaxed problem. For that reason, we considered only small instances and compared the results with the previous lower bounds. Table 4.1 summarizes the results of 188 experiments considering instances with 6 up 16 jobs. Column MKS-LB presents the average values of the makespan lower bounds using the methods presented in Section 3.3.3, Column DEL-LB presents the lower bound obtained by the Lagrangian approach. Finally, Column Imp indicates the number of times the LB was improved by the relaxation approach.

Table 4.1: Comparison between lower bounds. Columns MKS-LB and DEL-LB present the average values of the lower bounds for makepan and delays, respectively. Column Imp indicates the number of times the LB was improved by the relaxation approach.

NJobs	MKS-LB	DEL-LB	Imp
6	59.111	43.111	13
7	69.063	43.326	7
8	76.077	51.179	10
9	83.176	58.300	10
10	82.333	48.459	6
11	83.059	77.639	11
12	83.950	59.192	6
13	82.105	53.351	6
14	89.000	36.828	10
15	91.150	55.271	7
16	99.250	48.419	7
Average	81.661	52.280	

Is important to remember that with the separation of the objective function the final lower bound will be the sum of MKS-LB and DEL-LB, for this reason, the lower bound is improved every time the DEL-LB has a nonzero value. The results show a 70% of improvement in the lower bound values and a 50% of the instances have their lower bounds improved by the algorithm. Even with these strong results, it is important to remark that these values depend on how the instances are generated. The proposed algorithm is going to obtain best results in instances with tight due dates, precisely when the delays are more frequent.

4.3 Conclusions and future works

In this chapter, we introduced a time-indexed formulation and proposed an algorithm based on its Lagrangian relaxation to improve the instances lower bounds.

The relaxation of the model seems very attractive because we are able to decompose a difficult problem to a new one that can be solved in a very efficient way. Nevertheless, only small instances were tested and the results for larger problems could change considerably.

An efficient implementation of the algorithm, avoiding the use of CPLEX is taking place, it is expected a significant CPU time reduction allowing the test of larger problems.

Part II

Flow Shop

Chapter 5

Permutacional Flow Shop Problem

5.1 Introduction

The Flow Shop Problem (FSP) is a scheduling problem in which n jobs have to be processed by m machines. The problem is to find the sequence of jobs for each machine to minimize completion time, also known as makespan (Baker, 1993). This problem is NP-Hard for $m > 3$ (Coffman, 1976; Kan, 1976). Several papers in the literature address this problem, proposing models, heuristics, and bounds. Dannenbring (1977) tested several heuristics. Nawaz et al. (1983) presented a polynomial time algorithm (NEH), finding interesting results. Until now, NEH is one of the best polynomial time heuristics for this problem. Taillard (1990) presented an improvement in the complexity of the NEH algorithm, a heuristic based on tabu search, and a useful characterization of the distribution of the objective function. Taillard proposed a series of test problems with strong upper bounds. The running time required by Taillard's tabu search heuristic was not given and focus was limited to solution quality. Ben-Daya and Al-Fawzan (1998) implemented and tested an improved variant of Taillard's tabu search, reporting times and comparing their performances with Ogbu and Smith's simulated annealing algorithm (Ogbu and Smith, 1990). However, they did not match all of Taillard's results for large instances. Stützle presented and tested an Iterated Local Search (ILS) heuristic obtaining good results. Ruiz and Maroto (2005) compared 25 methods, from very basic ones, such as Johnson's algorithm (Johnson, 1954), to more sophisticated ones, such as tabu search and simulated annealing. The results of their study concluded that NEH was the best polynomial-time heuristic while Stützle's ILS, (Stutzle, 1998), and Reeves's genetic algorithm, (Reeves, 1995), were the best metaheuristic-based heuristics. Interesting results can be found in (Nagano and Moccellin, 2002), the authors proposed a polynomial time heuristic (N&M) with competitive results against the NEH algorithm. Ruiz et al. (2003) proposed a new memetic algorithm for this problem, obtaining improved results when compared with the ILS and tabu search. The same authors followed up with another paper in the same direction (Ruiz et al., 2006), proposing and testing two genetic algorithms and obtaining strong results. Agarwal et al. (2006) implemented a heuristic improvement procedure based on adaptive learning and applied it to the NEH algorithm, leading to additional improvements. However, for larger instances, their results were of poor quality and their algorithm was computationally intensive. These results seemed to present a few issues, that we discuss in Section 5.5. Ruiz and Stutzle (2006) described a simple algorithm with which they obtained good results and presented six new upper bounds.

In (Agarwal et al., 2006; Ben-Daya and Al-Fawzan, 1998; Taillard, 1993) and almost all other papers dealing with the FSP, the problem of interest was a special case called Permutation Flow Shop Problem (PFSP) in which the jobs have identical ordering sequences on all machines. This widespread approach is useful because it allows a simpler implementation, specially for genetic algorithms, and it is known that a PFSP solution is a good approximation of the FSP solution.

In this chapter, we consider the PFSP and propose two hybrid algorithms, called hybrid A and hybrid B. Both Algorithms use the heuristic NEH, GRASP-ILS, Path-Relinking, and Memetic Algorithm, see Figure 5.3. The chapter is organized as follows. In section 5.2 the mathematical model is presented. In Sections 5.3 and 5.4, we describe the GRASP-ILS and the Memetic algorithm. In Section 5.5, we present the computational experiments and their analysis. Section 5.6 concludes this part of the research.

5.2 Model

It is well known that any job shop model can be used to model a flow shop problem, see (Błażewicz et al., 1996). In the particular case of a Permutation Flow Shop Problem (PFSP), the model is based on Wagner's job shop model, (Wagner, 1959), and can be found in (Stafford, 1988) and (Błażewicz et al., 1996). The notation used is presented below,

- x_{ij}: $\begin{cases} 1 & \text{if job } i \text{ is scheduled on position } j \text{ in the permutation ,} \\ 0 & \text{otherwise;} \end{cases}$
- w_{jm}: idle time on machine m before start processing the $j-th$ in the permutation;
- y_{jm}: idle time of job $j-th$, after finishing on machine m while waiting for machine $m+1$ become free;
- Z: makespan (maximum completion time);

$$\min Z$$

Subject to

$$\sum_{j=1}^{|N|} x_{ij} = 1, \qquad \forall i \in N \quad (5.1)$$

$$\sum_{i=1}^{|N|} x_{ij} = 1, \qquad \forall j \in N \quad (5.2)$$

$$\sum_{i=1}^{|N|} p_{im}.x_{ij+1} + y_{j+1m} + w_{j+1m} = y_{jk} + \sum_{i=1}^{|N|} p_{im+1}.x_{ij} + w_{j+1m+1}, \quad (5.3)$$

$$\forall m = 1 \dots |M| - 1, \forall j = 1 \dots |N| - 1 \quad (5.4)$$

$$\sum_{j=1}^{|N|} \sum_{i=1}^{|N|} p_{im}.x_{ij+1} + \sum_{j=1}^{|N|} x_{jm} = Z, \quad (5.5)$$

$$\sum_{m=1}^{k-1} \sum_{i=1}^{|N|} p_{im}.x_{i1} = w_{1k} \qquad \forall k, k = 2, \dots, |M| - 1 \quad (5.6)$$

$$y_{1k} = 0, \qquad k = 1, \dots, |M| - 1 \quad (5.7)$$

$$x_{ij} \in (0, 1), \qquad \forall i, j \in N \quad (5.8)$$

$$w_{jk} \geq 0, \qquad \forall i \in N \forall k \in M \quad (5.9)$$

$$y_{ik} \geq 0, \qquad \forall i \in N \forall k \in M \quad (5.10)$$

$$Z \geq 0 \quad (5.11)$$

The process of assigning a job to a certain position in the permutation is granted by Equations (5.1) and (5.2). Equations (5.3) compute the times between any pair of jobs, taken into account idle times. The makespan is compute by Equation (5.5). The idle time for all the machines, besides the first one while waiting for the arrival of the first job, is compute by Equations (5.6). Equations (5.7) grant that the first job in the permutation can pass immediately to the successive machine. Finally, non-negative constraints complete the model, from (5.8) to (5.11).

5.3 GRASP-ILS

As it is well known, a traditional GRASP and ILS have two main phases, a construction phase and a local search. Readers unfamiliar with GRASP and ILS are referred to (Feo and Resende, 1995; Festa and Resende, 2002; Resende and Ribeiro, 2003b) and (Lourenco et al., 2003), (Stutzle, 1998), respectively. In our implementation, we also use path-relinking (Feo and Resende, 1995).

5.3.1 Construction phase

Usually, the construction phase is a greedy algorithm, with some randomness, in which a new solution is obtained at each iteration. In our case, we use the NEH heuristic to construct the first solution. The pseudo-code of the NEH algorithm is presented in Algorithm 5.1.

Algorithm 5.1: NEH Algorithm

```
1 Sort the jobs by decreasing sums of processing times;
2 Schedule the first two jobs minimizing the partial schedule;
3 for $i = 3$ to $n$ do
4     Insert the $i$-th job in one of the $i$ possible places of the partial solution, minimizing
      the partial schedule.
5 end
```

5.3.2 ILS

As in the implementations of ILS, the GRASP-ILS heuristic has a perturbation phase which perturbs the current solution by making two-swap movements at random positions. If an improved solution is found, the current solution is updated; if no improvement is found, the current solution is updated with probability $p_T > 0$. This is a simulated-annealing-like acceptance criterion (Ruiz and Stutzle, 2006; Stutzle, 1998). If n is the number of jobs, m is the number of machines, and p_{ij} are the processing time for job i at machine j, then the temperature T used in the acceptance process is

$$T = 0.5 \cdot \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}}{n \cdot m \cdot 10}.$$

5.3.3 GRASP

In a standard GRASP, at each iteration a new solution is constructed, usually blending a random procedure with a greedy heuristic. In the heuristic proposed in this work, we maintain a pool of good-quality solutions on which to apply a path-relinking procedure. After 20 iterations without improvement, we reconstruct the pool using a partial NEH algorithm while preserving most of the structure of the current best solution.

5.3.4 Local Search

We use a well-known local search that has been previously applied to this and other combinatorial optimization problems. Tabu search, ILS, and genetic algorithms have used the same type of local search scheme, e.g. (Ruiz and Stutzle, 2006). This scheme, which we call `LSInsertion` is based on the insertion procedure of the NEH algorithm. For every job, we remove the job from its original position and attempt to insert it in one of the n possible positions. If an improvement is obtained, then the process is repeated. This procedure works by choosing the positions at random and terminates after a complete search without improvement. It is very similar to local search LS3 used in the VNS algorithm in § 2.4.4.

5.3.5 Path-relinking

Path-relinking (PR) is used as a form of intensification of the search procedure. PR is applied from the incumbent solution to a solution randomly selected from the pool (Feo and Resende, 1995). Beginning at a random position and considering only swaps, three possible moves are analyzed and the best move is selected. At each step, the new solutions are analyzed and the best one is saved. In

Figure 5.1 an example of the procedure is shown. A similar approach and some variations can be found in (Feo and Resende, 1995).

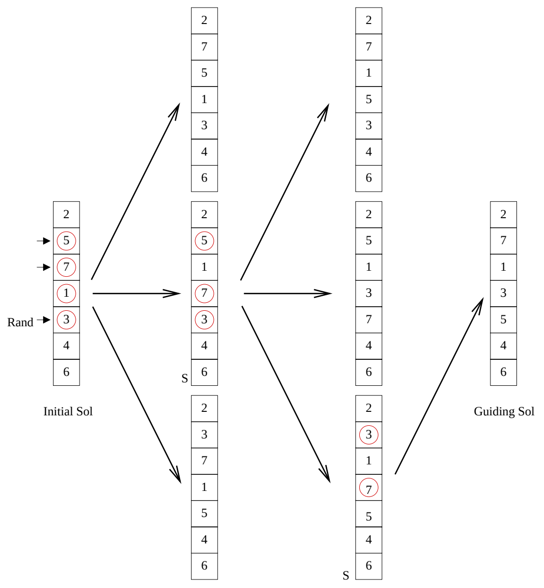


Figure 5.1: Example of the path-relinking procedure. At each step, a random position is chosen and from this point, in a cyclic order, three possible changes are analyzed. The letter ‘S’ indicates which solution is selected for the next step. This solution corresponds to the best move analyzed. Only five steps are performed if the guiding solution is not reached first.

The PR procedure terminates when the whole path between the solutions is analyzed or after testing a small number of solutions in the path, returning the best solution found in the process. In our implementation, we only allow five steps, and at each step only three possible solutions are analyzed.

The main characteristic of the PR is that at each new step the solutions maintain most of the original structure. When the procedure begins from the best solution, the goal is to preserve most of this successful structure with some influence of the selected pool solution.

5.3.6 Hybrid GRASP-ILS

Combining these ideas results in the hybrid GRASP-ILS heuristic whose pseudo-code is shown in Algorithm 5.2. In this algorithm, `Solution`, `CurrSol` and `BestSolution`, represent different solutions and their corresponding makespan values. `PoolSolution` is a random solution chosen from the pool.

In line 1 of the pseudo-code, the jobs are sorted in decreasing order of the sums of their processing times and the order is saved in the array `JobSorted`. In line 2, the NEH heuristic is called to produce the initial solution that is inserted in the pool in line 3. In line 4, the local search is applied on the initial solution and the local optimum can itself be inserted into the pool in line 5. `NonImproves`, the counter of iterations without incumbent solution improvement, is initialized in line 7.

The loop in lines 8 to 38 constitutes the main iterations of the heuristic. From line 9 to line 11 the current solution `CurrSol` is perturbed and the local search is applied to the perturbed solution. Lines 13 to 21 are only executed every `PRF` iterations, where `PRF` is the path-relinking frequency parameter. In line 13 the path-relinking procedure is applied from the incumbent solution (`BestSolution`) to a solution selected at random from the pool (`PoolSolution`). If certain criteria are met, the path-relinking solution (`SolPR`) is inserted into the pool in line 14. In lines 15 to 21 and 23 to 29 the current solution can be updated when the current solution is improved or with a positive probability otherwise. The pool of solutions is replaced after a certain number of nonimproving iterations (`NImp`).

This pool of solutions contains a set of good, or elite, solutions found during the process. As we can see in Algorithm 5.2, after each local search and after the path-relinking, the algorithm tries to introduce the new solution into the pool. For this insertion, not only the quality of the solution is considered but also the similarity with the other solutions in the pool. This idea is not new and the reader can refer to (Aiex et al., 2003) for example. In our case we do not allow the insertion of solutions that are too similar to the ones already in the pool. A solution is inserted if its objective value is the best so far or if its objective value is better than the worst objective value in the pool and the solution is different enough from the solutions in the pool.

The similarity between two solutions is computed by counting the number of different jobs for a certain position. The importance of a diversified pool is discussed in the next section.

5.4 Memetic Algorithm

When working with the PFSP we only deal with a permutation vector, which has an easy representation. This is one of the main advantages of working with the PFSP. The idea of the post-optimization is to use the information gathered by the GRASP-ILS heuristic to search for new solutions near solutions in the pool. For this reason and based on the experience of Ruiz and Maroto (2005) and Ruiz et al. (2006) we use a Memetic Algorithm (MA), reader can refer to Moscato (1989) and Moscato (2002). The MA works using the pool of solutions produced by the GRASP-ILS as the initial population in addition to a number of random solutions to allow it to search other regions of the feasible solution space.

As the MA is used along with GRASP-ILS, we need a simple structure and only a few operators. A mutation procedure, a path crossover, and the cold restart are the only operators used. These operators are described next.

5.4.1 Mutation Operator (M)

This operator works as a perturbation approach of GRASP-ILS. Its main goal is to allow the algorithm to search beyond the neighborhood defined by the local search. This operator makes two swap moves at random positions. It is applied only to the best or second best solution, randomly chosen in the population.

5.4.2 Path Crossover (PX)

This procedure is similar to path-relinking and it is used as a crossover by Ahuja et al. (2000) for solving quadratic assignment problems. After the selection of two parents, the crossover consists of the construction of a path between those parents. This operator begins the procedure from a random position and continues until it reaches this position again. The parent's alleles (jobs) are compared. If they are the same, then the offspring inherits that allele. If the alleles differ, two distinct swap moves can be done, one on each parent. The algorithm always selects the move which results in a better objective function value. The offspring resulting from this crossover is the best solution generated during the construction of the path. Figure 5.2 shows an example of the PX operator.

We also tested a few other crossover operators, such as multi-parent and similar block order crossovers, but these did not produce results as good as PX or were excessively expensive to compute.

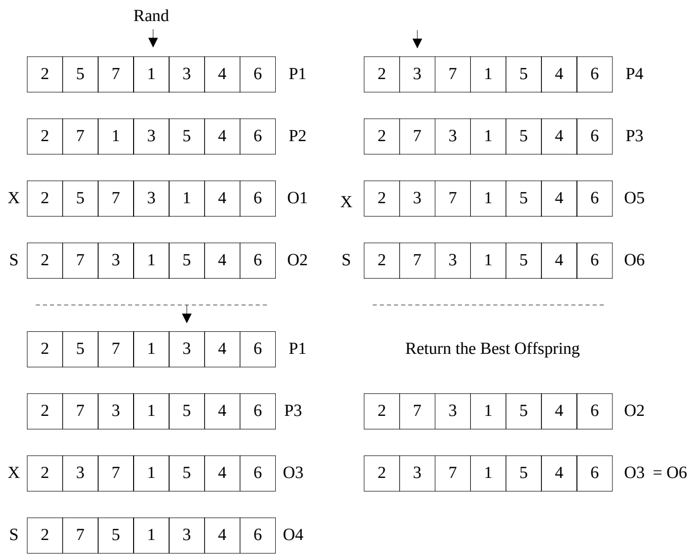


Figure 5.2: PX – S indicates the offspring with better objective function and X the discarded offspring. The first position to check is randomly chosen.

5.4.3 Cold Restart

Cold restart is used in many genetic algorithms with the idea of creating a more diversified population. This is useful when the algorithm reaches a state of stagnation, that is, when no improvement occurs after a certain number of generations. Cold restart changes the population by modifying, replacing, and maintaining part of the solutions. The three best solutions are left unchanged. Approximately half of the population results from applying the mutation operator on one of the three best solutions. About one fourth of the population is generated by applying the iterated greedy algorithm of Ruiz and Stutzle (2006) to the incumbent solution. The remaining elements of the population are replaced by randomly generated solutions.

5.4.4 Hybrid algorithms

The hybrid algorithms combine the GRASP-ILS and MA heuristics. This combination can be done in several ways, but in this work we use two approaches called hybrid A and B.

Initially, the solution of the polynomial-time heuristic NEH is used as the initial pool solution. If this solution can be locally improved, then the improved solution is also inserted into the pool. In both cases the pool of elite solutions is used by the path-relinking component of the hybrid heuristics.

The difference between both hybrid approaches lays in when the MA heuristic is called. In the first algorithm, we use the MA as a post-optimization procedure to intensify the search around the best solutions found during GRASP-ILS phase. After running GRASP-ILS for a fixed number of iterations, the MA uses the pool of elite solutions produced in the GRASP-ILS phase as its initial population. Path-relinking also plays a role in this hybrid heuristic in that it contributes to the pool as well as makes use of elements of the pool for intensification.

In the second approach, the GRASP-ILS algorithm calls the MA algorithm after the algorithm reaches a state of stagnation for a fixed number of iterations. In this case, the MA algorithm uses the pool of elite solutions as the initial population for a certain number of generations. After that, the GRASP-ILS continues to run with the pool provided by the MA output.

The algorithm structures are shown in Figure 5.3, for now on we will refer to these cases as hybrid A and hybrid B, respectively. The diversity of solutions in the pool is critical in both the GRASP-ILS and MA phases. It allows different promising regions of the solution space to be explored.

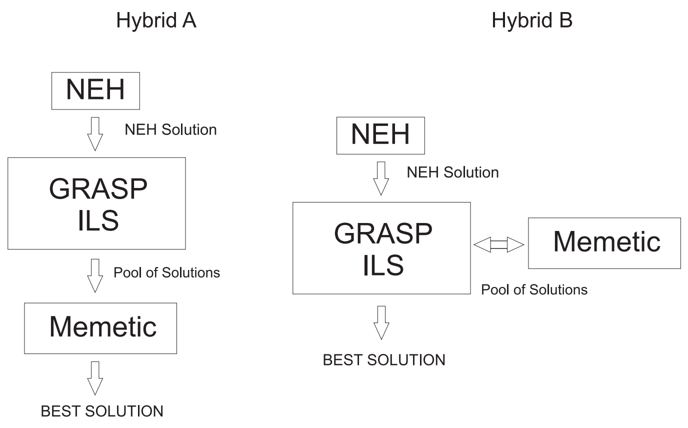


Figure 5.3: Hybrid algorithm structures.

5.5 Computational Results

To analyze the performance of the hybrid heuristics, we designed several experiments using benchmark instances proposed by Taillard (1990) as our testbed. These experiments are composed of two parts. In the former, we define tests to calibrate certain parameters of our algorithm and in the latter, we analyze the behavior of the algorithm and its performance.

All the algorithms tested in this part of the thesis were implemented by the authors in the C programming language and compiled with GNU GCC version 3.2.3, using compiler options *-O6 -funroll-all-loops -fomit-frame-pointer -march=pentium4*. CPU times were computed using the function *getrusage()*. The experiments for the performance analysis were run on a Dell PowerEdge 2600 computer with dual 3.2 GHz 1 Mb cache XEON III processors and 6 Gb of memory RAM under a Red Hat Linux 3.2.3-53. The algorithm used for random-number generation is an implementation of the Mersenne Twister algorithm described in (Matsumoto and Nishimura, 1998).

5.5.1 Calibration

We first conduct a set of experiments to engineer the algorithm, i.e. set parameters and load balance the various components of the algorithm. Though we conducted extensive computational testing with numerous crossover operators, initial pool solution heuristics, local search algorithms, path-relinking schemes, population and pool sizes, and other parameters, we limit this discussion to the calibration of the memetic when using the PX as a crossover operator, the results of adding the path-relinking to the GRASP-ILS heuristic, and the relationship of both components of the hybrid heuristic.

We determine the parameter `PRF` (frequency of path-relinking), the crossover operator to be used in the MA, and the load balance between the GRASP-ILS and MA components.

5.5.1.1 Frequency of path-relinking (PRF)

The GRASP-ILS heuristic works with a pool of 15 solutions. Since path-relinking is not activated at each GRASP-ILS iteration, the main parameter to be analyzed is the frequency in which it is called. This is parameter `PRF` in Algorithm 5.2.

To test the effect of using path-relinking, we compared five versions of the hybrid algorithm which differed only with respect to the `PRF` parameter. This included a variant in which `PRF` $= 0$, i.e. the pure ILS proposed by Stutzle (1998). To perform these tests, we allowed the algorithm to run for 200 CPU seconds, regardless of instance size. The 30 chosen instances have 50, 100, and 200 jobs, with 20 stages, and they proved to be the most difficult of this benchmark. The use of time as a stopping criterion was considered because the path-relinking strategy directly affects CPU time and it is this tradeoff of quality and time that we want to understand.

For each instance tested, each variant was run five times using different random number generator seeds and the average percentage increase over the best known solution ($\overline{API}$) was calculated. These results are summarized in Table 5.1. $\overline{API}$ is computed considering the best upper bound known to the date. Equation (5.12) shows the formula for $\overline{API}$, where R is the number of repetitions, X the solution of the heuristic, and UB the best known upper bound.

$$\overline{API} = \frac{1}{R}\sum_{i=1}^{R}\left(\frac{X-UB}{UB}\cdot 100\right) \tag{5.12}$$

The results in Table 5.1 indicate that GRASP-ILS with path-relinking outperforms the pure ILS. Furthermore, on average the use of path-relinking frequency parameter `PRF` $= 2$ results in a better performance than any other frequency tested.

5.5.1.2 Crossover comparison

The memetic algorithm component of the hybrid heuristic has several parameters that need to be set. We conducted extensive experimentation to set these parameters. These experiments resulted in the following parameter choices. The size of the population was set to 20 solutions of which five are

Table 5.1: Average percentage increase over the best known solution for GRASP-ILS varying the path-relinking frequency parameter `PRF`.

Jobs · Machines	`PRF` = 2	`PRF` = 5	`PRF` = 10	`PRF` = 20	Pure ILS
	1.36104	1.78701	1.70909	1.49610	1.76623
	2.11123	2.00864	1.77106	1.76026	2.12203
	2.52129	2.39495	2.01044	2.48283	2.42790
	1.57358	1.57358	1.68099	1.83673	1.53598
50 · 20	1.54528	1.53974	1.47328	1.71144	1.62836
	1.74763	1.81275	1.97558	1.94844	1.78019
	1.57085	1.29015	1.67881	2.04049	1.73819
	2.53048	2.96938	2.86643	2.62260	2.92062
	1.84344	1.68314	1.58696	1.83810	1.46407
	1.29546	1.09371	1.12025	1.22113	1.39634
Average	1.810	1.815	1.787	1.896	1.878
	2.51532	2.56369	2.52499	2.42180	2.73460
	2.17911	2.30197	2.17911	2.17265	2.16618
	2.09217	2.01882	2.03795	2.21336	2.17509
	1.80252	1.82166	1.80252	1.88228	1.80252
100 · 20	2.30915	2.35350	2.05892	2.22363	1.95439
	2.22816	2.31301	2.20930	2.16216	2.22187
	2.11232	2.23676	2.08679	2.27186	2.34844
	2.78706	2.75269	2.83706	2.56522	2.59334
	2.82709	2.78247	2.76335	2.75697	2.80478
	1.82157	1.82468	1.75008	1.94591	2.04228
Average	2.267	2.297	2.225	2.2616	2.284
	1.74185	1.82939	1.76686	1.82582	1.81510
	2.36722	2.39579	2.37079	2.40650	2.35116
	2.37578	2.54346	2.53993	2.45521	2.45698
	1.98512	2.09846	2.21179	2.23481	2.25961
200 · 20	1.39977	1.64491	1.72129	1.72484	1.68043
	2.25042	2.22540	2.05917	2.17535	2.17535
	1.92118	1.95109	2.05489	1.88600	1.90887
	2.23222	2.25163	2.28163	2.34869	2.33104
	2.10686	2.20515	2.32666	2.46247	2.34989
	2.56741	2.52851	2.61339	2.77783	2.70179
Average	2.095	2.167	2.195	2.230	2.203
Overall Average	**2.057**	**2.093**	**2.069**	**2.129**	**2.122**
Overall Standard Deviation	0.417	0.444	0.419	0.372	0.411

randomly generated at each new generation. The probability of applying the crossover is 40%, the mutation 1% and after applying the operators local search is carried out with a probability of 5%. To avoid superfluous computations, all the solutions have a flag which indicates if local search was applied to the solution. After completing a generation, local search is applied to the best solution with a probability of 10% if its flag indicates that local search has not been previously applied. We next compare the performance of the memetic algorithm when using two crossovers.

After comparing several crossovers Ruiz et al. (2006) choose the *Similar Block Order Crossover* (SBOX), which presented a better performance than the other operators. The SBOX transfers to the offspring similar blocks of alleles from its parents. Two sequences of alleles are considered a similar block if there are at least two consecutive identical alleles in the same position in both parents. After the recognition of the similar blocks, a crossover point is randomly chosen and the remaining alleles are copied from their parents. By using the crossover point it is possible to generate two offsprings after each crossover. We compare the performance of SBOX with the PX operator described in Section 5.4.2.

As seen in Table 5.2, we were unable to reproduce the results obtained by Ruiz et al. (2006). The difference between the performances could be attributed factors such as differences in coding, data structure design, compilers, hardware discrepancies, or even computer configuration. Furthermore, Ruiz, Maroto, and Alcaraz report *elapsed* time instead of CPU time.

To evaluate these operators, once again we use the 30 instances from Taillard's benchmark with 50, 100, and 200 jobs and 20 stages. The experiment consisted of allowing each algorithm to perform five runs with different seeds where both algorithms are identical with the exception of the crossover used. Table 5.2 shows the results, where for each operator we present the $\overline{API}$ and the number of times in which the best solution was found using the crossover.

Table 5.2: Comparison between crossover operators listing $\overline{API}$, the average percentage increase over the best known solution, and the number of times in which each variant obtained the best solution in the comparison.

	SBOX crossover		PX crossover	
Jobs · Machines	$\overline{API}$	times best found	$\overline{API}$	times best found
50 x 20	2.28890	4	2.25186	6
100 x 20	2.60578	6	2.57615	6
200 x 20	2.38082	3	2.41622	7

We use the 'times best found' metric because sometimes the average value does not show all the information needed to make the correct choice. In this particular case, we can see that for the 200-job instances the variant with the SBOX crossover presents a lower average value but the algorithm using PX crossover obtains better results more times. In these tests, the use of PX as crossover has a better average performance than SBOX, finding best solutions in most cases. All other parameters used in the algorithm were fixed in the comparison.

5.5.1.3 Load balancing GRASP-ILS and MA for the hybrid A algorithm

When combining components GRASP-ILS and MA into a hybrid heuristic we need to determine what portion of the total running time will be allocated to each component. We call this choice load balancing. We conducted experiments to determine a good load balancing. Once again, we consider the same instances as before and use the 200 CPU seconds as the stopping criterion. The experiment consisted of allowing each variant (with distinct load balance) to perform five runs with different seeds. The possibilities of using the pure GRASP-ILS or the pure MA are also considered in the experiment.

Table 5.3 and Figure 5.4 present the results. The figure shows the confidence intervals for the different configurations of the hybrid algorithm. The variants are represented by their respective load balances. For example, variant 95/5 has a load balance with 95% of the CPU time used by the GRASP-ILS and 5% by the MA. As can be seen in the table and figure, the 95%-5% load balance variant has the best average performance.

Table 5.3: Average relative percentage deviation for different GRASP-ILS and MA load balances. G/M indicates G% of running time allocated to GRASP-ILS and M% of running time allocated to MA.

Jobs · Machines	GRASP-ILS	95/5	80/20	50/50	20/80	5/95	Memetic
	1.39221	1.36104	1.53247	1.48052	1.75065	1.71429	1.91169
	2.20302	2.11123	2.23542	1.87905	2.32721	2.23002	2.12203
	2.58720	2.52129	2.52678	2.48833	2.55974	2.60368	2.50481
	1.76155	1.57358	1.79914	1.69710	1.79914	2.05693	2.10526
50 x 20	1.78898	1.54528	1.78898	1.78344	1.91083	2.07699	2.06037
	2.05156	1.74763	2.05156	2.05156	2.05156	2.10583	2.09498
	1.76518	1.57085	1.79757	1.74359	1.84615	1.81916	2.35358
	2.56841	2.53048	2.56299	2.56299	2.91520	3.00732	3.07234
	1.85413	1.84344	1.85413	1.85413	1.91825	1.82207	2.23350
	1.46005	1.29546	1.39634	1.41757	1.40165	1.08840	2.05999
Average	1.94323	1.81003	1.95454	1.89583	2.04804	2.05247	2.25186
	2.47662	2.51532	2.75395	2.72170	2.73460	2.88617	3.02806
	2.16618	2.17911	2.26964	2.54122	2.64791	2.75461	3.01326
	2.09217	2.09217	2.14958	2.21017	2.16233	2.58970	2.49721
	1.81528	1.80252	1.80890	1.83442	1.73233	2.04498	1.88547
100 x 20	2.27431	2.30915	2.36300	2.50871	2.41368	2.22997	2.44219
	2.23130	2.22816	2.22816	2.32558	2.70585	2.69956	2.66813
	2.11232	2.11232	2.09317	2.17294	2.29419	2.53350	2.59413
	2.85268	2.78706	2.85893	2.99328	3.06827	3.16825	2.74957
	2.82709	2.82709	2.83665	2.89721	2.87809	3.04064	3.05976
	1.82157	1.82157	1.89307	2.02362	2.15107	1.83712	2.11999
Average	2.26695	2.26745	2.32551	2.42289	2.47883	2.57845	2.60578
	1.81331	1.74185	1.81331	1.82760	1.91157	1.83117	1.95266
	2.50647	2.36722	2.49755	2.55467	2.46006	2.41899	2.53861
	2.52405	2.37578	2.53993	2.54699	2.52758	2.46404	2.60524
	2.27554	1.98512	2.35346	2.52701	2.44200	2.43492	2.62617
200 x 20	1.57030	1.39977	1.66800	1.65024	1.78169	1.64491	1.89182
	2.37912	2.25042	2.38806	2.39878	2.29154	2.32192	2.47386
	2.06193	1.92118	2.08128	2.03730	2.00739	1.79803	2.17628
	2.25693	2.23222	2.24634	2.38574	2.45280	2.54632	2.54632
	2.18728	2.10686	2.28020	2.20872	2.34811	2.44282	2.41780
	2.68411	2.56741	2.73362	2.78490	2.86270	2.70179	2.93343
Average	2.22590	2.09479	2.26018	2.29220	2.30854	2.26049	2.41622
Overall Average	**2.14536**	**2.05742**	**2.18007**	**2.20364**	**2.27847**	**2.29714**	**2.42462**

5.5.2 Performance Analysis

We next investigate the algorithm's dependence on the initial random number generator seed and in its running time. The tables below show the performance of the algorithm after ten runs for each instance. In these experiments, we use number of iterations as the stopping criterion, so that the experiments can be more accurately reproduced.

The hybrid B algorithm does not need a load balancing calibration, but after performing similar experimentation two important parameters were set, the algorithm calls the GA only after a stagnation of 20 iterations, and it is allowed to run 20 generations after returning the improved pool of elite solutions.

5.5.2.1 Time to target

Many local search based combinatorial optimization heuristics, including GRASP, GRASP with path-relinking, and memetic algorithms have running time to the optimal solution that are distributed

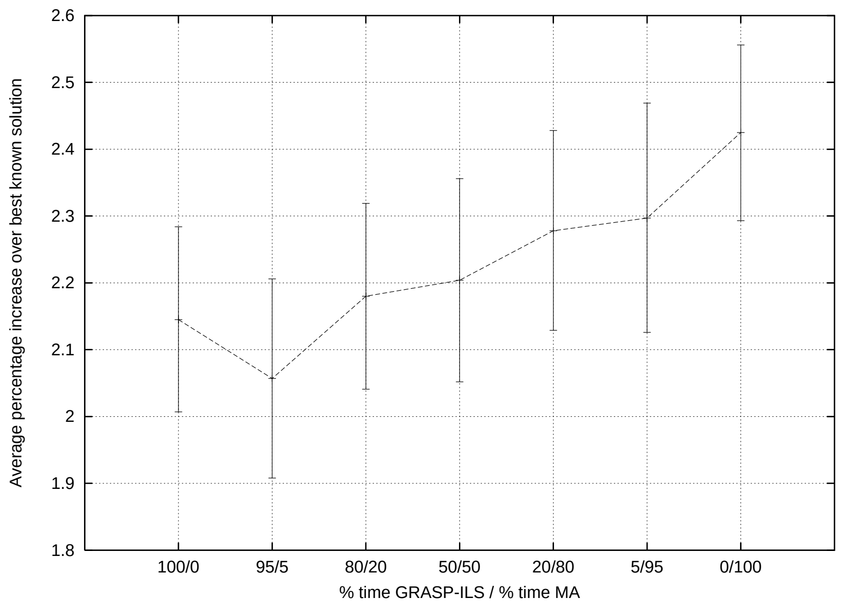


Figure 5.4: 95% confidence intervals for hybrid algorithms.

according to a shifted exponential distribution (Aiex et al., 2002). To study the random variable *time to target solution value* we ran the hybrid A and hybrid B algorithms using one hundred different seeds for one instance with 100 jobs and 20 stages[1]. The seeds were randomly generated and are distinct from the ones used in the calibration process.

The hybrid A algorithm uses 95% of the processing time allowed with GRASP-ILS and the last 5% with the MA. In this test the maximum CPU time allowed to each run is 10,000 seconds. For the hybrid B algorithm, the MA is called after a period of 20 iterations without improving the incumbent solution, and after 20 generations the MA algorithm returns the pool to the GRASP-ILS algorithm.

A target of 2.5% was chosen for this experiment. For this particular case, the best known solution has a value of 6314, therefor the target was set to 6471. The algorithms will stop after obtaining a solution less or equal to 6471 or after 10,000 seconds. Time-to-target plots (ttt-plots) are produced for these values using ttt-plots code, see (Aiex et al., 2005). Figure 5.5 shows the ttt-plots obtained.

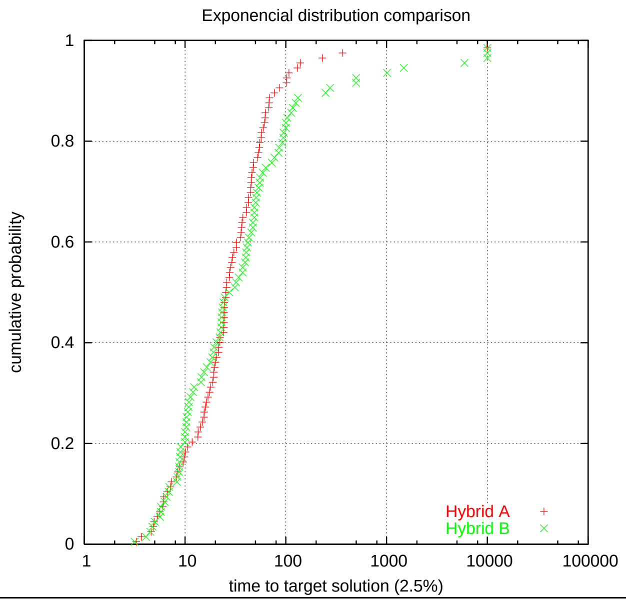
Of interest is that within 80 seconds, almost 80% of the cases for both algorithms reached the target and within 1000 seconds more than 90%. These results show that the choice of the seed is not a critical aspect for obtaining a reasonable performance. By this comparison it is possible to realize the hybrid A algorithm presents a performance slightly better than the hybrid B algorithm.

5.5.2.2 Performance

In this section we present the results obtained by using our algorithm in the 120 instances from Taillard's Benchmark. The algorithm performs a fixed number of iterations for both the GRASP-ILS

[1]The instance used is this experiment is one from Taillard's benchmark, called ta085.

Figure 5.5: CPU time (seconds) needed to reach a solution no further than 2.5% of the best known upper bound. The time is on a logarithmic scale.



and the memetic part.

Some authors use a relation between the number of jobs and the number of machines to set a time for testing the performance of the algorithms. The problem with this approach is that for this particular benchmark, the difficulty of the instances do not seem to increase with the increase of the number of jobs. We resume this discussion in next section.

In Tables 5.4 and 5.5 the average values of our experiments are presented, for both hybrid algorithms, and compared with the best known solutions found in the literature. The best known results were found by different authors and using different strategies. A percentage deviation equal to zero indicates that the best known solution was found by the algorithm. It is possible to see that the standard deviation (StdDev) remains with low values, indicating a robust method. The obtained results are not only competitive but also can be used in real life problems.

When comparing the hybrid approaches, it is possible to observe, the performance of the hybrid A algorithm is slightly better than algorithm hybrid B.

5.5.2.3 Remarks regarding the instances

It is interesting to analyze the difficulty of our algorithms for some instances. It is quite obvious and expected that the increase in the number of machines increases the effort needed by our algorithms to find a good solution. In part, this could be explained by considering that the makespan for an specific permutation can be calculated by using a function depending on the number of jobs and machines, and the time wasted on this evaluations explain the need of extra time to reach the same result quality.

Nevertheless it is quite remarkable to notice that the worst performance of our algorithms happens for the group of instances with 100 jobs and 20 stages (100x20) and for 200x20. And these results are

Table 5.4: Average relative percentage increase for Taillard instances. Column CPU 1 indicates the average time (in CPU seconds) spent to find the best solution. Column CPU 2 indicates the average time (in CPU seconds) taken by the algorithm hybrid A.

						Hybrid A
Jobs · Machines	API	Best	Worst	StdDev	CPU 1	CPU 2
20 · 5	**0.079**	0.000	1.138	0.234	0.28	5.06
20 · 10	**0.328**	0.000	1.194	0.327	5.44	18.74
20 · 20	**0.224**	0.000	0.826	0.233	8.00	28.64
50 · 5	**0.003**	0.000	0.071	0.014	15.18	125.24
50 · 10	**0.713**	0.000	2.407	0.602	206.10	1186.92
50 · 20	**1.222**	0.324	3.791	0.783	1743.90	4647.40
100 · 5	**0.012**	0.000	0.285	0.044	129.16	680.38
100 · 10	**0.256**	0.000	1.263	0.270	322.32	1422.72
100 · 20	**1.430**	0.702	2.183	0.388	1905.28	4350.28
200 · 10	**0.270**	0.009	0.754	0.164	498.12	1433.04
200 · 20	**1.672**	0.782	2.615	0.342	2523.44	4218.04
500 · 20	**0.981**	0.570	1.459	0.213	2493.02	3894.82
Average	**0.604**					

Table 5.5: Average relative percentage increase for Taillard instances. Column CPU 1 indicates the average time (in CPU seconds) spent to find the best solution. Column CPU 2 indicates the average time (in CPU seconds) taken by the algorithm hybrid B.

						Hybrid B
Jobs · Machines	API	Best	Worst	StdDev	CPU 1	CPU 2
20 · 5	0.058	0.000	0.810	0.167	0.439	4.584
20 · 10	0.275	0.000	2.011	0.382	3.472	14.813
20 · 20	0.231	0.000	0.653	0.186	4.404	28.169
50 · 5	0.019	0.000	0.353	0.071	8.026	93.952
50 · 10	0.700	0.000	1.605	0.485	260.749	1179.363
50 · 20	1.390	0.297	3.787	0.603	1479.574	4108.018
100 · 5	0.006	0.000	0.095	0.020	153.866	687.559
100 · 10	0.258	0.017	0.858	0.256	263.291	1175.743
100 · 20	1.857	0.798	3.624	0.550	1096.554	3575.116
200 · 10	0.257	0.009	0.754	0.184	287.079	1044.240
200 · 20	1.863	0.835	2.766	0.431	1105.303	2129.254
500 · 20	0.902	0.471	1.388	0.203	3149.858	4816.021
Average	**0.651**					

consistent with previous results no matter the method used. The main question to answer is why for 500x20 the algorithm has a better performance. In Taillard's webpage we can find that for 500x20 there are seven out of ten instances with results proved to be optimal solutions, for 200x20 there four out of ten optimal solutions and for the case 100x20 only one optimal result has been found so far and none for 50 x 20.

Next we discuss some new facts that need review. In 2006 Agarwal et al. (2006) discuss an approach to solve the PFSP. In this paper the authors claim to have found a better upper bound for one instance of Taillard's benchmark by using an adaptive-learning approach (over well known polynomial heuristics).

The instance is ta040 (50 jobs and 5 stages). In the paper, they found a permutation with 2774 as a makespan value, but they did not present the real permutation values. The problem is that in Taillard's website[2] the optimal value for this instance is 2782.

Something similar occurs in the instance ta068 (100 jobs and 5 stages) in which we found two problems: the first one is that the upper bound is not correct, the authors use 5034, while the correct upper bound and supposedly also the optimal value is 5094. This could be only a typing mistake, but the second problem is that the solutions found for two methods presented in that paper have 5082 as a makespan value, that is, smaller than the optimal solution value.

Another comment about that paper is that the upper bounds for most of the larger instances are changed with the lower bounds. In those cases the results presented by the authors tend to be better than they are claiming to be.

It is very important to review these questions in order to avoid future difficulties for other researchers, especially because these instances are referenced in almost every paper dealing with FSP. And for future publications, it seems to be more appropriate to present the solutions when a new upper bound or a significant result is reached.

5.6 Concluding Remarks and future research

In this part of the work we discussed the Permutational Flow Shop Problem. This problem has been extensively studied and there are several algorithms published with excellent performance.

We presented two implementations of hybrid algorithms based on GRASP, ILS and a memetic algorithm. In one approach we use the GRASP-ILS algorithm first maintaining a pool of good and diversified solutions in order to apply path-relinking and to construct a population for the memetic algorithm. In the second approach, we use the memetic algorithm to improve the pool of elite solutions every time the GRASP-ILS algorithm reaches a state of stagnation for an specific number of iterations.

The memetic algorithm uses a crossover based on path-relinking (PX) that was shown to be very effective. The best performance is obtained using the hybrid A algorithm, when we allow the GRASP-ILS to perform a large search almost 95% of the time and then the memetic algorithm try to obtain a better solution using the output of the previous one.

The results are quite competitive showing that for the hardest instances of Taillard's benchmark, 100 jobs and 20 stages, the algorithm can find a solution no worse than 2.5% of the best known upper bound in less than 80 seconds for 80% percent of the cases.

We also discussed some characteristics about the difficulty of the benchmark and some new results presented in a recent paper that need to be reviewed.

The next step in our research is the consideration of sequence dependent setups and due dates in all stages, or only in some of them. We are also considering for a multi-objective approach, trying to obtain a flexible algorithm that could face different real-life circumstances. We believe that the use of this kind of hybrid approaches will be most advantageous in these more complicated cases, especially when dealing with multi-objective problems.

[2]http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/problemes.html

Algorithm 5.2: GRASP-ILS Algorithm.

```
JobSorted ← Sort the jobs by decreasing sums of processing times;
Solution ← NEH(JobsSorted);
PoolInsertion(Solution);
Solution ← LSInsertion(Solution);
PoolInsertion(Solution, Proximity);
CurrSol ← BestSolution;
NonImproves ← 0;
for $i \leftarrow 1, \ldots,$ MaxIterations do
    Solution ← Perturbation(CurrSol);
    Solution ← LSInsertion(Solution);
    PoolInsertion(Solution, Proximity);
    if $i$ mod PRF $= 0$ then
        SolPR ← PathRelinking(BestSolution, PoolSolution);
        PoolInsertion(SolPR, Proximity);
        if CurrSol > SolPR then
            CurrSol ← SolPR;
            else
                if RND(0, 1) < exp(−SolPR − CurrSol)/$T$ then
                    CurrSol ← SolPR;
                end
            end
        end
    end
    if CurrSol > Solution then
        CurrSol ← Solution;
        else
            if RND(0, 1) < exp(−Solution − CurrSol)/$T$ then
                CurrSol ← Solution;
            end
        end
    end
    if BestSolution *is improved* then
        NonImproves ← 0;
        else
            NonImproves ← NonImproves + 1;
        end
    end
    if NonImproves = NImp then
        RestartPOOL();
    end
end
```

Part III

General Conclusions and Future Research

Chapter 6

Summary and concluding remarks

In the first part of the dissertation, our research focuses on a scheduling problem with features that are not often found in the literature. The problem considers sequence and machine dependent setup times, unrelated parallel machines, and due dates. The objective is to schedule the jobs in order to minimize the sum of the makespan and the weighted delays.

In Chapter 2 two heuristics were proposed and tested. The first heuristic was based on GRASP. This approach proved to be very simple and flexible; it is relatively simple to modify this algorithm to other purposes and situations and it has only a few main parameters that are easily adjusted. In our particular case the time spent by the algorithm is very reasonable according to real-world constraints.

The second algorithm was based on VNS. We run a series of experiments comparing the VNS algorithm with three versions of GRASP, each one with a different local search. Analyzing the computational results, we conclude that the VNS provides very good average results for instances with 60 jobs or more, especially for short range due dates. The combination of three different local searches proved to be very effective with a simple implementation and with no need of adjusting several parameters. The NEH also proved to be remarkable as a procedure to find initial solutions.

In Chapter 3 a B&B algorithm using GRASP as an initialization procedure, and a two MIP model were proposed and tested. Also as a contribution, we generated a set of instances with different values of due dates, setups and processing times.

We showed that the B&B has a much better performance than the two MIP models solved by CPLEX 9.0. An increase in the variation of the setup times may increase the number of nodes branched by the B&B, whereas an increase in the due dates or processing times may decrease the number of nodes in the enumeration tree.

In Chapter 4 a time-indexed formulation is introduced and an algorithm based on its Lagrangian relaxation is proposed to improve the instance's lower bound. The relaxation of the model seems very attractive because we are able to decompose the original problem to a relaxed problem that can be solved in a very efficient way. Nevertheless, only small instances were tested and the results for larger problems could considerably change.

In the second part of the dissertation, we deal with flow shop problems. We presented two implementations of hybrid algorithms based on GRASP, ILS and a Memetic Algorithm for the permutation case. In one approach (hybrid A) we use the GRASP-ILS algorithm first maintaining a pool of good and diversified solutions in order to apply path-relinking and to construct a population for the Memetic

Algorithm. In the second approach (hybrid B), we use the Memetic Algorithm to improve the pool of elite solutions every time the GRASP-ILS algorithm reaches a state of stagnation for an specific number of iterations.

The best performance is obtained using the hybrid A algorithm, when we allow the GRASP-ILS to perform a large search almost 95% of the time and then the Memetic Algorithm try to obtain a better solution using the output of the previous one. The results are quite competitive showing that for the hardest instances of Taillard's benchmark, 100 jobs and 20 stages, the algorithm can find a solution no worse than 2.5% of the best known upper bound in less than 80 seconds for 80% percent of the cases.

6.1 Future works

In this final section we comment some of the ongoing projects and future works that have emerged out of the research presented in this work. Considering the parallel machine case, we are currently working on three projects. In the first project we are working on a Local Branching based algorithm. Initially we are considering the MILP model proposed in Section 3.2.1. The objective is to compare the Local Branching approach for the three proposed models.

In many different scenarios, specially in production planning very large scale problems must be solved in a short time, for that reason, the second project analyzes the implementation of a heuristic with high-throughput, considering a distribute architecture.

The third project tackles the integration of two levels of production planning, proposing an iterative scheme to generate a production plan that takes into account scheduling constraints due to changeover setup times in single-level manufacturing systems. In this integration we are using the GRASP algorithm presented in this dissertation.

Considering the flow shop environment, we are working on the Flexible Flow Shop Problem (FFSP)[1]. In this case, we have identical parallel machines at each stage, that can perform the same tasks for all jobs. For this problem, we are working on lower bounds, models and heuristics approaches.

[1]Also know as Compound flow shop or Hybrid flow shop

Appendix A

Models comparison

In this appendix, we want to compare the three proposed models in terms of number of variables and contraints. Table A shows the comparison of the three proposed models. One important detail is that the time-indexed formulation needs an upper bound for the horizon considered. This situation is directly related to the number of constraints the model has, but can be easily solved by allowing GRASP to perform one iteration.

Table A.1: Comparison between the proposed models.

	Manne	Wagner	time-indexed (T=500)
variables	351	6071	24291
Objectives nonzeros	11	11	10
Linear Constraints	682	5650	218459

In this evaluation the time horizon was arbitrarily set to 500. The first solution obtain by GRASP is 352 time units, so the number of variables will considerably decrease.

Appendix B

GRASP and exact solutions

In this chapter we present the results of 60 experiments for small instances, 5, 10 and 15 jobs. In the all the instances our GRASP with its basic configuration find all the optimal solutions. In this experiment, GRASP is set to perform 20000 iterations. Tables B.1, B.2 and B.3 summarize the experiment. Column *F.Obj* shows the objective function value. Column *Iteration* and *Time Best* present the number of the iteration and the CPU time [seconds], in which the best solution as found. Finally, Column *Time* presents the CPU time of the whole process [seconds].

Table B.1: GRASP solution for instances with 5 jobs and 2 unrelated machines.

Instance	*F.Obj*	*Iteration*	*Time Best*	*Time*
00	213	0	0.00	0.39
01	278	0	0.00	0.43
02	263	13	0.00	0.39
03	642	0	0.00	0.41
04	385	1	0.00	0.40
05	187	0	0.00	0.42
06	283	22	0.00	0.39
07	246	1	0.00	0.39
08	208	0	0.00	0.41
09	551	0	0.00	0.41
10	212	0	0.00	0.44
11	307	81	0.00	0.39
12	379	9	0.00	0.37
13	237	3	0.00	0.47
14	272	0	0.00	0.40
15	213	1	0.00	0.43
16	307	0	0.00	0.42
17	415	0	0.00	0.38
18	502	0	0.00	0.42
19	400	1	0.00	0.44
		6.60	0.00	0.41

Table B.2: GRASP solution for instances with 10 jobs and 6 unrelated machines.

Instance	*F.Obj*	*Iteration*	*Time Best*	*Time*
00	143	14	0.01	4.32
01	173	31	0.01	4.47
02	418	11	0.01	4.33
03	136	19	0.01	4.46
04	185	0	0.00	5.16
05	216	99	0.02	4.27
06	195	27	0.01	4.40
07	104	2	0.00	4.45
08	157	10	0.00	4.09
09	169	3	0.00	4.27
10	141	177	0.04	4.43
11	194	28	0.01	4.36
12	330	4	0.00	4.19
13	190	8	0.01	4.43
14	97	13	0.00	4.33
15	165	286	0.06	4.29
16	120	0	0.00	4.48
17	284	14	0.01	4.34
18	141	1189	0.27	4.42
19	123	45	0.01	4.40
Average		99.00	0.02	4.39

Table B.3: GRASP solution for instances with 15 jobs and 6 unrelated machines.

Instance	*F.Obj*	*Iteration*	*Time Best*	*Time*
00	346	134	0.07	10.48
01	162	25	0.02	10.77
02	263	4	0.01	10.51
03	234	29	0.02	11.08
04	351	336	0.19	11.18
05	172	15	0.01	10.83
06	271	25	0.02	10.44
07	241	1370	0.69	10.19
08	165	635	0.34	10.87
09	209	247	0.15	11.53
10	355	8833	4.79	10.92
11	174	1242	0.65	10.54
12	283	12261	6.26	10.30
13	156	0	0.00	10.76
14	142	116	0.06	10.67
15	176	10	0.01	10.67
16	206	26	0.02	10.62
17	174	9	0.01	9.75
18	262	1585	0.83	10.58
19	344	1145	0.61	10.62
Average		1402.35	0.74	10.67

Appendix C

NEH as first solution

The first solution of our two heuristic is based on a well known heuristic called NEH. This heuristic was initially formulated for the Permutation Flow Shop Problem, and we adapted to the PMSP. In this appendix 20 tests are summarize. This experiment has the objective of analyze the quality of the NEH solution against the solution obtained by any other of the proposed heuristics.

For this experiment instances with 100 jobs and 6 unrelated parallel machines were chosen. The algorithm is going to use NEH as a construction procedure. Table C summarizes the results. Columns *NEH* and *TimeN* present the results obtained by the NEH algorithm and its CPU time. Columns *GRASP* and *TimeG* present the final results after 10000 iterations and its CPU time, respectively.

Table C.1: NEH and GRASP solutions for instances with 100 jobs and 6 unrelated machines.

Instances	NEH	Time	GRASP	Time
00	22445	0.07	7388	454.21
01	21304	0.07	9577	395.81
02	33846	0.07	13271	436.78
03	28510	0.07	9891	392.71
04	31537	0.07	11461	402.45
05	30809	0.07	12691	413.47
06	35120	0.07	20669	386.4
07	22742	0.07	12145	376.12
08	32457	0.07	9927	388.53
09	34323	0.07	13276	383.16
10	22955	0.07	5348	395.34
11	39648	0.07	18927	368.9
12	33873	0.07	14972	394.95
13	25167	0.07	8106	390.97
14	28351	0.07	10293	413.24
15	28958	0.07	6648	398.91
16	31516	0.07	8858	368.22
17	30078	0.06	9205	376.39
18	32693	0.07	7513	393.53
19	35039	0.07	15272	392.34

Bibliography

Adamopolos, G. and Pappis, C. (1998). Scheduling under common due date on parallel unrelated machines. *European Journal of Operational Research*, 105(3):494–501.

Agarwal, A., Colak, S., and Eryarsoy, E. (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research*, 169(3):801–815.

Ahuja, R. K., Ergun, O., Orlin, J. B., and Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3):75–102.

Ahuja, R. K., Orlin, J. B., and Tiwari, A. (2000). A greedy genetic algorithm for the quadratic assigment problem. *Computers & Operations Research*, 27(10):917–934.

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

Aiex, R., Resende, M., and Ribeiro, C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373.

Aiex, R., Resende, M., and Ribeiro, C. (2005). TTTPLOTS: A perl program to create time-to-target plots. Technical report, AT&T Labs,Technical Report TD-6HT7EL. To appear in *Optimization Letters.*

Armentano, V. A. and Ronconi, D. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers & Operations Research*, 26(3):219–235.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling.* John Wiley & Sons Inc, New York.

Baker, K. R. (editors S C Graves et al.1993). Requirements planning. In *Handbooks in OR & MS.*

Beasley, J. E. (1993). Lagrangian relaxation. In Reeves, C. R., editor, *Modern heuristic techniques for combinatorial problems*, pages 243–303. John Wiley & Sons, Inc., New York, NY, USA.

Ben-Daya, M. and Al-Fawzan, M. (1998). A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research*, 109(1):88–95.

Binato, S., Hery, W., Loewenstern, D., and Resende, M. (2002). A greedy randomized adaptive search procedure for job shop scheduling. In *Essays and surveys on metaheuristics*, pages 58–79. C.C. Ribeiro and P. Hansen, editors.Kluwer Academic.

Błażewicz, J., Ecker, K., Pesch, E., Schimdt, G., and Węglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*. Springer - Verlag, Berlin.

Błażewicz, J., Lenstra, J., and Kan, A. R. (1983). Scheduling subject to resource constraints. *Discrete Applied Mathematics*, 5:11–24.

Brucker, P. (2004). *Scheduling Algorithms*. Springer-Verlag, Berlin.

Cheng, T. and Q. Ding, B. L. (2004). A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152(1):1–13.

Cheng, T. and Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292.

Coffman, E. (1976). *Computer and Job-Shop Scheduling Theory*. Wiley, New York.

Cormen, T., Leiserson, C., and Rivest, R. (1990). *Introduction to Algorithms*, chapter 26: All-Pairs Shortest Paths. The MIT Press, Cambridge, MA.

Dannenbring, D. G. (1977). An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182.

de Paula, M. R., Gómez Ravetti, M., Mateus, G. R., and Pardalos, P. M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, 18(2). In Press, http://dx.doi.org/10.1093/imaman/dpm016.

Dyer, M. E. and Wolsey, L. A. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2–3):255–270.

Feo, T. and Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Feo, T., Sarathy, K., and McGahan, J. (1996). A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, 23(9):881–895.

Feo, T., Venkatraman, K., and Bard, J. (1991). A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research*, 18(8):635–643.

Festa, P. and Resende, M. (2002). GRASP: An annotated bibliography. In Ribeiro, C. and Hansen, P., editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers.

Fisher, M. L. (1981). The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27:1–18.

Frangioni, A. (2005). About lagrangian methods in integer optimization. *Annals of Operations Research*, 139:163–193.

García-López, F., Melián-Batista, B., Moreno-Pérez, J., and Moreno-Vega, J. M. (2002). The parallel variable neighborhood search for the p-median problem*. *Journal of Heuristics*, 8(3):375–388.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co., New York, USA.

Geoffrion, A. M. (1974). Lagrangean relaxation for integer programming. *Math. Programming Study*, 2:82–114.

Glover, F. (2000). Multi-start and starategic oscilation methods - principles to exploit adaptive memory. In Laguna, M. and Gonzáles-Velardes, J. L., editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in ComputerScience and operations Research*, pages 1–24. Kluwer Academic Publishers.

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

Gómez Ravetti, M. (2003). Problemas de seqüenciamento com máquinas paralelas e tempos de preparação dependentes da seqüência. Master's thesis, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais.

Gómez Ravetti, M., Nakamura, F. G., Meneses, C. N., Resende, M. G., Mateus, G. R., and Pardalos, P. M. (2006). A hybrid heuristic for the permutational flow shop problem. In *19th International Symposium on Mathematical Programming*, Rio de Janeiro, Brazil.

Gómez Ravetti, M., Rocha, P. L., Mateus, G. R., and Pardalos, P. M. (2007). A scheduling problem with unrelated parallel machines and sequence dependent setups. *Int. Journal of Operational Research*. In Press.

Graham, R., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling theory: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Hansen, P. and Mladenovic, N. (1999). Variable neighborhood search: Principles and applications. *European journal of operational reasearch*, 130:449–467.

Hansen, P. and Mladenovic, N. (2003). A tutorial on variable neighborhood search. *Le cahiers du GERARD*, G-2003:46.

Hansen, P., Mladenovic, N., and Brimberg, J. (2002). Convergence of variable neighborhood search. *Les Cahiers du GERAD*, G-2002-21.

Held, M. and Karp, R. (1970). The travelling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162.

Held, M. and Karp, R. (1971). The travelling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1:6–25.

Ho, J. C. and Chang, Y.-L. (1995). Minimizing the number of tardy jobs for m parallel machines. *European Journal of Operational Research*, 84(2):343–355.

Jansen, K. and Porkolab, L. (2001). Improved approximation schemes for scheduling unrelated parallel machines. *Mathematics of Operations Research*, 26(2):324–338.

Johnson, D., Aragon, C., and Schevon, L. M. C. (1989). Optimization by simulated annealing: An experimental evaluation; part 1, graph partitioning. *Operations Research*, 37:865–892.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Nav. Res. Log. Quarterly*, 1:61–68.

Kan, A. R. (1976). *Machine scheduling problems: classification, complexity and computations*. The Hague : Nijhoff.

Kim, D.-W., Kim, K.-H., Jang, W., and Chen, F. F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18(3-4):223–231.

Kim, D.-W., Na, D.-G., and Chen, F. F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, 19(1-2):173–181.

Laguna, M. and Martí, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52.

Lawler, E., Lenstra, J. K., Kan, A. R., and Shmoys, D. (1993). Sequencing and scheduling: algorithms and complexity. In Graves, S., Kan, A. R., and Zipkin, P., editors, *Handbooks in Operations Research and Management Science 4, Logistics of Production and Inventory*, pages 445–524. North Holland.

Lee, C.-Y. and Pinedo, M. (2002). Optimization and heuristics of scheduling. In Pardalos, P. M. and Resende, M. G. C., editors, *Handbook of Applied Optimization*. Oxford University Press, New York.

Lourenco, H., Martin, O., and Stutzle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G., editors, *Handbook of metaheuristics*. Kluwer.

Manne, A. (1960). On the job–shop scheduling problem. *Operations Research*, 8(2):219–223.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30.

Matsuo, H., Sug, C., and Sullivan, R. (1989). A controlled search simulated annealing method for the single machine weighted tardiness problem. *Annals of Operations Research*, 21:85–108.

Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia - Pacific Journal of Operational Research*, 18(2):193–242.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program, C3P Report 826.

Moscato, P. (2002). Optimization and heuristics of scheduling. In Pardalos, P. M. and Resende, M. G. C., editors, *Handbook of Applied Optimization*. Oxford University Press, New York.

Mosheiov, G. (2004). Simultaneus minimization of total completion time and total deviation of job completion times. *European Journal of Operational Research*, 157(2):296–306.

Nagano, M. and Moccellin, J. (2002). A high quality solution constructive heuristic for flow shop sequencing. *Journal of the Operational Research Society*, 53(12):1374–1379.

Nawaz, M., Enscore, E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

Ogbu, F. and Smith, D. (1990). The application of the simulated annealing to the solution of the n/m/cmax flow shop problem. *computers & Operations Research*, 17(3):243–253.

Oğuz, C., Ercan, M., Cheng, T., and Fung, Y. (2003). Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. *European Journal of Operational Research*, 149(2):390–404.

Pereira-Lopes, M. J. and de Carvalho, J. V. (2006). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*. In Press.

Pinedo, M. (1995). *Scheduling Theory, Algorithms and Systems*. Prentice Hall, New Jersey, USA.

Polacek, M., Hartl, R. F., and Doerner, K. (2004). A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10(6):613–627.

Potts, C. N. (1985). A lagrangean based branch and bound algorithm for single machine sequencing. *Management Science*, 31(10):1300–1311.

Queyranne, M. and Schulz, A. S. (1994). Polyhedral approaches to machine scheduling. Technical report, Preprint 408/1994, Department of Mathematics, Technical University of Berlin, Berlin, Germany.

Reeves, C. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13.

Resende, M. and Ribeiro, C. (2003a). GRASP and path-relinking: Recent advances and applications. In Ibaraki, T. and Yoshitomi, Y., editors, *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pages T6–1 – T6–6.

Resende, M. and Ribeiro, C. (2003b). Greedy randomized adaptive search procedures. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers.

Rocha, P. L., Gómez Ravetti, M., and Mateus, G. R. (2004). The metaheuristic GRASP as an upper bound for a branch and bound algorithm in a scheduling problem with sequence-dependent setup times. In *4th EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics*, Nottingham - England. Available on-line at http://webhost.ua.ac.be/eume/workshops/hybrid/A037Revised.pdf.

Rocha, P. L., Gómez Ravetti, M., Mateus, G. R., and Pardalos, P. M. (2007). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*. In Press, http://dx.doi.org/10.1016/j.cor.2006.07.015.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Ruiz, R., Maroto, C., and Alcaraz, J. (2003). New genetic algorithms for the permutational flowshop scheduling problem. *MIC2003: The Fifth Metaheuristics International Conference*, pages 63.1–63.8.

Ruiz, R., Maroto, C., and Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, The International Journal of Management Science*, 34:461–476.

Ruiz, R. and Stutzle, T. (2006). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, Available on-line.

Savelsbergh, M. W. P., Uma, R. N., and Wein, J. (1998). An experimental study of lp-based approximation algorithms for scheduling problems. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 453–462, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Sousa, J. P. and Wolsey, L. A. (1992). A time-indexed formulation of non-preemptive single-machine scheduling problems. *Mathematical Programming*, 54:353–367.

Stafford, E. F. (1988). On the development of a mixed-integer linear programming model for the flowshop sequencing problem. *Journal of the Operational Research Society*, 39(12):1163–1174.

Stutzle, T. (1998). Applying iterated local search to the permutation flow shop problem. Technical report, TU Darmstadt, AIDA-98-04, FG Intellektik.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

van den Akker, J., Hurkens, C., and Savelsbergh, M. (1998). Time-indexed formulations for machine scheduling problems: Column generation. citeseer.ist.psu.edu/vandenakker98timeindexed.html.

van den Akker, J., van Hoesel, C., and Savelsbergh, M. (1999). A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85:541–572.

Wagner, H. W. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistic Quarterly*, 6(2):131–140.