MATEUS ROCHA DE PAULA

HEURÍSTICAS PARA A MINIMIZAÇÃO DOS ATRASOS EM SEQUENCIAMENTO DE MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

Belo Horizonte
31 de agosto de 2008

MATEUS ROCHA DE PAULA
ORIENTADOR: GERALDO ROBSON MATEUS

HEURÍSTICAS PARA A MINIMIZAÇÃO DOS ATRASOS EM SEQUENCIAMENTO DE MÁQUINAS PARALELAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte
31 de agosto de 2008

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Heurísticas para a Minimização dos Atrasos em Sequenciamento de Máquinas Paralelas com Tempos de Preparação Dependentes da Sequência

MATEUS ROCHA DE PAULA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. GERALDO ROBSON MATEUS - Orientador
Departamento de Ciência da Computação - UFMG

PROF. GILBERTO DE MIRANDA JÚNIOR
Departamento de Engenharia de Produção - UFMG

PROF. MARCONE JAMILSON FREITAS SOUZA
Departamento de Ciência da Computação - UFOP

PROF. SEBASTIÁN ALBERTO URRUTIA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 12 de dezembro de 2008.

Agradecimentos

Gostaria de agradecer ao meu pai, José Vicente, por me proporcionar todas as boas oportunidades que tive e todo o necessário para chegar até aqui, fazendo de tudo para me tornar um homem melhor. A minha mãe e irmã, France e Rafa, pelo amor incondicional (mesmo quando eu tornava isso praticamente impossível!), garantindo que nossa casa fosse mais do que um teto: um lar. A Flávia, por me tornar completo e feliz, mesmo nos momentos mais difíceis. As minhas companheiras leais, Shilly e Minnie, que enfrentaram intermináveis horas de estudos e incontáveis linhas de código ao meu lado. Elas já devem estar mais perto de provar se P=NP do que eu. Não se surpreenda ao ver essas duas cachorrinhas recebendo um prêmio Nobel qualquer dia desses!

Gostaria também de agradecer ao meu orientador, Robson, que desempenhou um papel muito importante para mim nos últimos anos, sendo não somente um dos melhores mestres que já tive, mas também um excelente chefe e exemplo. Agradeço também ao meu co-orientador Martín, que tem me ajudado em toda a minha carreira acadêmica, tentando sempre desenvolver o meu melhor, e se tornando mais do que um tutor: um amigo.

Finalmente, gostaria ainda de agradecer a todos os meus amigos e colegas. Felizmente, são demais listar, e todos queridos demais para enumerar. Quando se trabalha mais com máquinas do que com pessoas (e, muitas vezes, até com pessoas através de máquinas), são eles que nos mantém humanos e honestos.

Resumo

Considere o problema de sequenciar um conjunto de tarefas, a serem processadas exatamente uma vez em qualquer máquina de um conjunto de máquinas não-relacionadas, sem preempção. Cada tarefa tem uma data de entrega, um peso e, para cada máquina, além de um tempo de processamento, um tempo de preparação dependente da sequência. Em todo este trabalho, o objetivo é minimizar a soma dos atrasos ponderados das tarefas, mas outros objetivos, como a minimização do makespan e da soma das folgas também são discutidos.

Inicialmente, este trabalho propõe e analisa implementações eficientes de diversas heurísticas baseadas em buscas locais para abordar o problema. Fatores como o desenho e detalhes de implementação dos algoritmos são discutidos. As heurísticas propostas são então comparadas com outras implementações bem sucedidas, para mostrar suas vantagens em termos de qualidade e tempo de computação, principalmente para instâncias de grande porte.

Para medir a qualidade das soluções, os valores de função objetivo das soluções obtidas são comparados com limites inferiores do problema. Para obter tais limites, um algoritmo *Non-Delayed Relax-and-Cut* é desenvolvido a partir de uma relaxação lagrangeana de uma formulação indexada no tempo. Para obter-se soluções aproximadas para o problema, a relaxação lagrangeana apresentada também é utilizada para desenvolver uma heurística lagrangeana.

Para alcançar um máximo desempenho e economia de memória, viabilizando o tratamento de instâncias de grande porte do problema, os algoritmos desenvolvidos não utilizam pacotes genéricos de otimização.

Foram obtidas, em tempo razoável, boas soluções para instâncias com até seis máquinas e 200 tarefas, e limites inferiores para instâncias com até seis máquinas e 80 tarefas. Os limites inferiores foram particularmente bons para instâncias fáceis, provando a otimalidade de algumas soluções e provendo *gaps* estreitos para outras. Para as instâncias mais difíceis, os limites inferiores obtidos não foram tão bons, mas ainda significativos.

Abstract

Consider the problem of scheduling a set of jobs to be processed exactly once, on any machine of a set of unrelated parallel machines, without preemption. Each job has a due date, weight, and, for each machine, an associated processing time and sequence-dependent setup time. Throughout this work, the objective function considered is to minimize the total weighted tardiness of the jobs, but other objectives, such as the minimization of the makespan and the minimization of the total slackness, are also discussed.

Initially, this work proposes and analyses efficient implementations of several local search based heuristics to tackle the problem. Aspects such as the algorithms' design and implementation aspects are discussed. Then, the proposed heuristics are compared with other successful implementations, to highlight their advantages in terms of quality and computation time, specially for large instances.

In order to measure the quality of the proposed solutions, their objective function values are compared to lower bounds of the problem. These bounds are obtained by a Non-Delayed Relax-and-Cut algorithm, based on a lagrangean relaxation of a time indexed formulation of the problem. It is also used to develop a lagrangean heuristic, to obtain approximate solutions.

To achieve maximum performance and memory saving, thus allowing to tackle large instances, the developed algorithms do not rely on third party solvers.

Good solutions for instances with up to six machines and 200 jobs, and lower bounds for instances with up to six machines and 80 jobs, were obtained within reasonable time. The obtained lower bounds were particularly good for easy instances, proving the optimality of some solutions and providing tight gaps for others. For more difficult instances, the obtained lower bounds were not so good but still significant.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Problemas de sequenciamento podem ser encontrados nas mais diversas áreas da ciência. Em biologia por exemplo, a bioinformática vem sendo cada vez mais estudada. Em ciência da computação, diversos problemas de recuperação de informação e arquitetura de computadores podem ser modelados como problemas de sequenciamento.

No contexto industrial, os problemas de sequenciamento muitas vezes surgem no processo de planejamento da produção. O programa de produção é feito de acordo com um horizonte de planejamento. Decisões para o longo prazo têm características estratégicas e são tomadas pela alta administração. No médio prazo, conhecido como nível táctico, o objetivo é planejar a produção de forma a minimizar seu custo, atendendo à demanda e aos prazos definidos para o longo prazo. Os problemas de sequenciamento frequentemente aparecem no nível operacional, que considera a produção de um dia, semana ou um mês; onde um dos principais objetivos é definir uma sequência de produção que atenda a um ou mais objetivos.

Decisões operacionais afetam diretamente as tácticas que, por sua vez, têm grande impacto nas decisões estratégicas. Um bom planejamento da produção é fator chave para alcançar uma utilização adequada dos recursos disponíveis e atender bem o objetivo desejado. Assim, é importante que se busque bons métodos para obter bons planejamentos em todos os níveis. Esta pesquisa trata problemas residentes no nível operacional.

Os problemas de sequenciamento estudados neste trabalho consistem num único estágio, composto por máquinas paralelas não relacionadas, com tempos de preparação dependentes da máquina e da sequência, tempos de processamento dependentes da máquina e datas de entrega e pesos para os produtos.

Estes problemas são conhecidos por serem de difícil solução. O problema de sequenciamento com tempos de preparação dependentes da sequência em uma única máquina é NP-Difícil, quando o objetivo é minimizar a soma dos atrasos ponderados (Veja Pinedo (1995) e Du e Leung (1990)). Também é possível provar que minimizar o *makespan* em duas máquinas idênticas é NP-Difícil (veja Garey e Johnson (1997) e Lenstra et al. (1977)). Usando uma redução ao Problema de Emparelhamento em 3 Dimensões, Crama e Spieksma (1993) provaram que, mesmo quando todas as tarefas têm a mesma duração, o problema de minimizar o custo total de processamento, que é função da data de início de processamento (ex: soma dos atrasos ponderados ou a soma das folgas), é NP-Difícil.

Assim sendo, o problema mais complexo, de minimizar a soma dos atrasos ponderados, o makespan ou a soma das folgas em múltiplas máquinas não-relacionadas, com datas de entrega e tempos de preparação dependentes da máquina e da sequência, também é NP-Difícil, já que é uma generalização de um dos casos anteriores.

1.1 Motivação

Considere uma fábrica com um conjunto de tarefas que devem ser processadas, sem preempção, em qualquer máquina de um conjunto de máquinas não relacionadas. Cada tarefa tem diferentes tempos de processamentos para cada máquina, e tempos de preparação dependentes não somente da máquina, mas também da sequência.

O problema tratado nesta pesquisa é baseado no caso real de uma fábrica de tijolos refratários. Ela produz mais de 8000 produtos, todos com a mesma sequência de produção que consiste em 5 estágios: medida de material, mistura, conformação, tratamento de calor e empacotamento. Para cada um destes estágios, a fábrica dispõe de mais de uma máquina, e a maioria dos produtos pode ser processada por qualquer uma delas num determinado estágio.

Neste cenário, por ser intimamente relacionado à qualidade do produto final, o estágio de conformação é a parte mais crítica da produção. Além disso, o plano de produção da fábrica é feito de acordo com o planejamento do estágio de conformação. Assim que este é definido, o planejamento dos outros estágios é ajustado de tal forma que os objetivos definidos sejam atendidos. Por esta razão, a maior parte do esforço do planejamento é investido no estágio de conformação, que é o foco deste trabalho.

O estágio de conformação da fábrica estudada utiliza sete máquinas: quatro idênticas, que precisam de trabalho manual, duas completamente automáticas e uma última designada a somente um tipo de tijolo. O planejamento da produção é feito considerando um problema de sequenciamento com máquinas não relacionadas para as seis primeiras máquinas, e um problema de sequenciamento em uma única máquina para a sétima, ambos considerando tempos de preparação dependentes da sequência e datas de entrega. As instâncias de interesse prático para estes problemas, considerando o caso real descrito, têm por volta de seis máquinas e 150 tarefas. Um melhor detalhamento do caso real pode ser encontrado em Ravetti (2007).

1.2 Problemas de Sequenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Sequência

Considere um conjunto de máquinas não-relacionadas $\mathscr{M} = \{1,2,..,|M|\}$, e um conjunto de tarefas $\mathscr{J} = \{1,2,..,|J|\}$.

Existem vários tipos de problemas de sequenciamento, que se enquadram às mais diversas aplicações reais. O caso mais específico é o de sequenciamento em uma única máquina, onde a sequência de processamento de todas as tarefas deve ser definido em uma determinada máquina.

Quando se dispõe de mais de uma máquina operando em paralelo, que realizam a mesma função, os problemas são modelados como Problemas de Sequenciamento com Máquinas Paralelas. Neste

modelo, qualquer máquina pode processar qualquer tarefa, e toda tarefa deve ser processada exatamente uma vez. As máquinas consideradas podem ser idênticas, uniformes ou não-relacionadas. No caso das máquinas idênticas, todas as máquinas demoram o mesmo tempo para processar qualquer tarefa. No caso de máquinas uniformes, o tempo de processamento depende somente da máquina. Já no caso de máquinas não relacionadas, o tempo de processamento depende da tarefa e da máquina.

Se as tarefas têm de ser processadas por mais de uma máquina, modelos de *shop* são utilizados. Em problemas *Flow-Shop*, todas as tarefas seguem a mesma sequência de máquinas. Isto é, eles têm de ser processados primeiro por uma máquina 1, depois por uma máquina 2 e assim em diante. Se, em cada estágio, dispõe-se de um conjunto de máquinas que podem ser usadas, ao invés de uma única (ex: uma tarefa tem de ser processada por uma máquina do conjunto 1, depois por uma máquina do conjunto 2, e assim em diante), o problema é conhecido como *Flow-Shop* flexível. Se cada tarefa tem uma sequência específica de máquinas a ser visitada, o modelo é conhecido como *Job-Shop*. Finalmente, se as máquinas precisam ser processadas por todas as máquinas, em qualquer ordem, o modelo é conhecido como *Open-Shop*.

Existem ainda diversas características que podem ser associadas às tarefas, como datas de disponibilidade, tempos de preparação, preempção, restrições de precedência, quebras de máquinas, restrições de elegibilidade, permutações, bloqueios, recirculação, e muitos outros.

São também muitos os objetivos que podem ser considerados na aplicação. Alguns dos mais comuns são a minimização do *makespan*, da soma dos atrasos ponderados e da soma das folgas.

Para maiores informações sobre problemas de sequenciamento, e abordagens clássicas, veja Pinedo (1995), Lee e Pinedo (2002), Blazewicz et al. (1996) e Brucker (2004).

Os problemas de sequenciamento com máquinas paralelas abordados neste trabalho consistem em determinar uma máquina para cada tarefa, e sequenciar o conjunto de tarefas designadas para cada máquina, considerando uma determinada função objetivo. Cada tarefa $j \in \mathscr{J}$ tem exatamente uma operação associada, que pode ser realizada por qualquer máquina $m \in \mathscr{M}$. Não é permitido preempção, isto é, uma vez que o processamento de uma tarefa começa, ele não pode ser interrompido.

No problema realista tratado, são associados pesos positivos w_j a cada tarefa $j \in \mathscr{J}$. Para processar uma tarefa j imediatamente após uma tarefa j' numa máquina m, é necessário um tempo de preparação $s_{j'jm}$ dependente tanto da sequência das tarefas $j' \prec j$, quanto da máquina $m \in \mathscr{M}$, onde as tarefas são processadas. É necessário ainda um tempo de processamento p_{jm}, que depende somente da máquina $m \in \mathscr{M}$, onde a tarefa j é processada.

Defina recursivamente a data de término de uma tarefa j como $C_j = C_{j'} + p_{jm} + s_{j'jm}$, onde $C_{j'}$ é zero se a tarefa j é a primeira a ser processada, ou a data de término da tarefa anterior $j' \prec j$, caso contrário. O atraso τ_j de cada tarefa é calculado como $\tau_j = max(C_j - d_j, 0)$, onde d_j é a data de entrega da tarefa j. Seja ainda o *makespan* C_{max}, a data de término da última tarefa a terminar seu processamento $C_{max} = max_{j \in \mathscr{J}}\{C_j\}$. Todo o processamento deve ser feito dentro do horizonte de planejamento $T > C_{max}$. Neste trabalho, $T = \infty$ sempre que não especificado. Assume-se que todas as tarefas e máquinas encontram-se disponíveis no tempo-base t=0.

Os diferentes objetivos, considerados separadamente nesta pesquisa, são a minimização da soma dos atrasos ponderados (TWT), $min \sum_{j=1}^{n} w_j \tau_j$; a minimização do *makespan* (MKS), $min\ C_{max}$, e a minimização da soma das folgas (TS), $min \sum_{j=1}^{m} (C_{max} - C_j)$. Estes problemas também são conhecidos

como $R|s_{jj'm},\tilde{d}_j|\sum w_j\tau_j$, $R|s_{jj'm},\tilde{d}_j|C_{max}$ e $R|s_{jj'm},\tilde{d}_j|\sum_{j=1}^{m}(C_{max}-C_j)$, respectivamente. Recorra a Brucker (2004) para maiores informações sobre notação de problemas de sequenciamento.

Sequenciamentos são geralmente representados por gráficos de Gantt, que podem ser orientados por tarefas ou máquinas (figuras A.1 (a) e (b)). Por simplicidade, neste trabalho são utilizados ainda gráficos de Gantt simplificados, que consideram somente a ordem das tarefas, ao invés das posições ocupadas por elas no tempo (figura A.1 (c)). Como normalmente $|\mathscr{M}| \lll |\mathscr{J}|$, os gráficos de Gantt simplificados, orientados por máquina, serão preferidos por serem mais compactos e proverem uma noção mais intuitiva do sequenciamento como um todo. Não serão considerados períodos ociosos em máquinas, o que é conveniente para problemas de sequenciamento com máquinas paralelas sem datas de disponibilidade, uma vez que uma solução com períodos ociosos em máquinas pode ser automaticamente melhorada adiantando-se tarefas até que não existam mais períodos ociosos em nenhuma máquina.

1.3 Organização do Texto

Três heurísticas baseadas em buscas locais são propostas no capítulo 2. As duas primeiras são baseadas em duas metaheurísticas muito conhecidas: o *Greed Randomized Adaptive Search Procedure* (GRASP) e o *Variable Neighborhood Search* (VNS). A terceira é um algoritmo híbrido. Os métodos construtivos utilizados por estas heurísticas para gerar soluções iniciais são detalhadas na seção 2.1. As buscas locais utilizadas são detalhadas na seção 2.2.2. Alguns aspectos de implementação são discutidos com maior profundidade na seção 2.2.1. Na seção 2.3, as implementações propostas são testadas, e os resultados obtidos são comparados com outros resultados encontrados na literatura. Conclusões parciais são então expostas na seção 2.4.

Para medir a qualidade das soluções obtidas, três métodos para obter limites inferiores para os problemas são detalhados no capítulo 3. O primeiro método é a utilização de relaxações lineares de três formulações matemáticas. Na seção 3.1, a relaxação linear de uma formulação indexada no tempo, proposta por Ravetti (2007), é comparada com as relaxações lineares de duas outras formulações, baseadas nos trabalhos de Wagner (1959) e Manne (1960). O segundo método é a relaxação lagrangeana da formulação indexada no tempo apresentada. Este método foi sugerido por Ravetti (2007), mas em seu trabalho foram apresentados somente testes preliminares, utilizando pacotes genéricos de otimização, para problemas pequenos. Uma vez que tal método se mostrou promissor, uma implementação específica, que não depende de pacotes genéricos de otimização, é proposta na seção 3.2. Para atacar problemas de convergência do método proposto por Ravetti (2007), um algoritmo *Non-Delayed Relax-and-Cut* é proposto na seção 3.4. Uma heurística lagrangeana, também baseada na relaxação lagrangeana de Ravetti (2007), é proposta na seção 3.5. Finalmente, na seção 3.6, os algoritmos desenvolvidos são extensivamente testados, e as conclusões são apresentadas na seção 3.7.

O capítulo 4 conclui este trabalho e sugere alguns trabalhos futuros sobre o problema tratado.

Capítulo 2

Heurísticas Desenvolvidas

Para tratar instâncias de grande porte de Problemas de Sequenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Sequência, foram propostas e apresentadas três heurísticas baseadas em buscas locais: uma baseada na meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) (seção 2.2.4), uma baseada na metaheurística *Variable Neighborhood Search* (seção 2.2.5) e uma heurística GRASP híbrida (seção 2.2.6), que utiliza uma heurística VNS como busca local.

2.1 Heurísticas Construtivas

Para gerar soluções iniciais, um algoritmo completamente aleatório e duas heurísticas construtivas são utilizadas: o *Earliest Due Dates* (EDD) e o *Nawaz-Enscore-Ham* (NEH). Tais métodos são descritos na seção A.2.1.

É possível encontrar outras heurísticas mais elaboradas na literatura como, por exemplo, as propostas por Hoon Lee e Pinedo (1997) e Pfund et al. (2008). Entretanto, tais heurísticas dependem de diversos parâmetros que devem ser obtidos experimentalmente. O propósito das heurísticas construtivas aqui é somente gerar soluções iniciais, que espera-se que sejam melhoradas muito rapidamente utilizando buscas locais. Por isso, optou-se pelas alternativas mais simples.

2.1.1 Algoritmo Aleatório

O algoritmo aleatório proposto visa somente criar uma solução viável rapidamente, e injetar diversidade em um outro método que o utilize. Espera-se que este seja o método construtivo mais rápido dentre os propostos, mas também o que a qualidade das soluções obtidas seja ruim. Este método, apresentado em maior detalhe na seção A.2.1.1, simplesmente insere tarefas, escolhidas aleatoriamente, no final da fila de processamento das máquinas, também escolhidas aleatoriamente.

2.1.2 EDD

Uma implementação da heurística *Earliest Due Dates* (EDD) é também apresentada. Este método insere tarefas, numa ordem determinada (geralmente ordem crescente de datas de entrega), no final

da fila de processamento de uma máquina, de tal forma que a função objetivo aumente o mínimo possível. Espera-se que este método seja mais lento que o método aleatório, mas obtenha soluções de melhor qualidade. Veja a seção A.2.1.2 para maiores detalhes sobre a implementação proposta.

2.1.3 NEH

Visando obter soluções iniciais de melhor qualidade, uma implementação da heurística *Nawaz-Enscore-Ham* (NEH) é também proposta. O NEH amplia as possibilidades de escolha de posicionamentos de uma nova tarefa no sequenciamento parcial do EDD, considerando, não somente a última posição da fila de processamento, mas todos os posicionamentos viáveis em todas as máquinas, que acarreta no menor aumento da função objetivo possível. É esperado que este seja o método construtivo mais lento dentre os propostos, mas que obtém as soluções iniciais de melhor qualidade. Veja a seção A.2.1.3 para maiores detalhes sobre a implementação proposta.

2.2 Heurísticas Baseadas em Buscas Locais

Aspectos de implementação dos movimentos em soluções, como a avaliação inteligente de soluções são discutidos em maior detalhe nas seções A.2.2.1 e A.2.2.1. Uma vez que testes preliminares mostraram que buscas primeiro-aprimorante (que aceitam imediatamente soluções melhores a medida que são encontradas) são ligeiramente mais rápidas que as melhor-aprimorantes (que aceitam somente a melhor solução encontrada em toda a vizinhança de cada solução verificada), estas são preferidas neste trabalho.

2.2.1 Movimentos e Vizinhanças Utilizadas

Três vizinhanças são consideradas: uma baseada na permutação de pares de tarefas (*swaps*), uma baseada na movimentação de tarefas e uma baseada na união das duas vizinhanças anteriores. Acredita-se que a utilização da união de duas vizinhanças seja uma boa idéia pois um mínimo local desta vizinhança resultante é garantidamente um mínimo local das duas partes. Veja a seção A.2.2.1 para maiores informações.

2.2.2 Buscas Locais

Implementações eficientes de três buscas locais, uma para cada vizinhança considerada, são detalhadas na seção A.2.2.2. Neste método, todas as soluções vizinhas da solução atual são avaliadas de forma inteligente e, ao encontrar uma solução melhor, a solução atual é atualizada. Este processo continua até que não se encontre uma solução vizinha melhor que a atual.

2.2.3 Soluções Iniciais

Para gerar soluções iniciais, as implementações propostas utilizam os métodos construtivos descritos na seção 2.1. A ordem de injeção das tarefas considerada, quando aplicável, é a ordem crescente de datas de entrega.

2.2.4 GRASP

A primeira heurística proposta é baseada na metaheurística GRASP. Em sua forma básica, a metaheurística GRASP propõe que, no início de cada iteração, uma nova solução seja criada utilizando um método construtivo. Esta solução deve então ser refinada utilizando uma busca local. Caso uma solução melhor que a atual seja encontrada, esta é atualizada. Este processo continua até que um critério de parada seja satisfeito. Neste trabalho são considerados dois critérios de parada: tempo limite e número de iterações sem melhoria da melhor solução encontrada. Este método é interessante por prover boa diversidade de soluções iniciais. A implementação proposta segue os moldes da heurística proposta por Rocha et al. (2008), visando melhorar a qualidade das soluções obtidas utilizando buscas locais mais agressivas no lugar do procedimento de *Path-Relinking*, utilizado para intensificar a busca local. Para um melhor detalhamento da implementação proposta veja a seção A.2.2.4.

2.2.5 VNS

A segunda heurística proposta, detalhada na seção A.2.2.5, é baseada na metaheurística VNS. O VNS, em sua estrutura básica é bem similar ao GRASP. A principal diferença entre as duas metaheurísticas, é que o VNS propõe que, no início de cada iteração, a solução atual sofra uma perturbação aleatória ao invés de construir uma nova solução. Isto permite que o VNS tenha memória de curto prazo, ao possível custo de perda de diversidade de soluções iniciais. A implementação proposta segue os moldes da heurística proposta por de Paula et al. (2007), visando também melhorar a qualidade das soluções obtidas, aproveitando ainda o ganho em velocidade de convergência observado pelo autor.

2.2.6 Heurística Híbrida

A terceira heurística proposta, é um algoritmo híbrido baseado também na metaheurística GRASP, mas que utiliza iterações de uma heurística VNS no lugar de sua busca local. Espera-se que este método aproveite os aspectos interessantes de ambas as metaheurísticas consideradas: a boa diversidade de soluções iniciais introduzida pelo esquema do GRASP e a memória de curto prazo introduzida pelo VNS. Espera-se, entretanto, que a velocidade de convergência do método seja inferior a do VNS básico. A implementação proposta é detalhada na seção A.2.2.6.

2.3 Experimentos e Resultados - Parte 1

Finalmente, as heurísticas construtivas e as buscas locais são extensivamente testadas e comparadas na seção A.2.3, a fim de determinar quais são mais vantajosas para o conjunto de instâncias testado (veja o apêndice B). A eleita melhor heurística desenvolvida é então comparada com resultados encontrados na literatura. A velocidade de convergência dos três métodos é também testada, afim de determinar qual método deve alcançar uma boa solução mais rápido.

2.4 Conclusões Parciais

Para o conjunto de instâncias testado (veja o apêndice B), o NEH se mostrou vantajoso sobre os outros métodos construtivos. Da mesma forma, a busca local baseada na união das duas vizinhanças consideradas se mostrou superior às outras. Todas as heurísticas propostas se mostraram superiores às heurísticas encontradas na literatura, encontrando soluções de melhor qualidade para a grande maioria das instâncias. O VNS, entretanto, apresentou a convergência mais rápida. Já que uma busca local bastante intensa e poderosa pode ser utilizada, o uso de outras técnicas de intensificação, como o *Path-Relinking* utilizado por Rocha et al. (2008), pôde ser dispensado, levando a uma heurística poderosa e simples, que depende de poucos parâmetros.

Veja a seção A.2.4, para uma argumentação mais detalhada sobre as conclusões parciais sobre as heurísticas baseadas em buscas locais.

Capítulo 3

Medindo a Qualidade das Soluções Obtidas

3.1 Formulações Matemáticas

Na seção A.3.1, uma formulação indexada no tempo, proposta por Ravetti (2007), é utilizada visando obter limites inferiores para o problema. Tais limites inferiores seriam úteis para medir a qualidade das soluções obtidas com as heurísticas desenvolvidas. É constatado que os limites inferiores obtidos com a relaxação linear de tal modelo são significativamente melhores que aqueles obtidos usando relaxações lineares de outros modelos, ao custo de uma enorme quantidade de memória, o que limita o tamanho das instâncias que esta formulação pode resolver.

3.2 Relaxação Lagrangeana

3.2.1 Relaxando Restrições de Precedência

Como alternativa à relaxação linear da formulação indexada no tempo, uma relaxação lagrangeana que dualiza suas restrições de precedência (conforme sugerida por Ravetti (2007)) é desenvolvida e apresentada na seção A.3.2. O problema relaxado tem a propriedade da integralidade e, portanto, os limites obtidos podem ser somente tão bons quanto os obtidos pela relaxação linear. Entretanto, é possível resolver o problema relaxado de forma eficiente, com um algoritmo guloso, que não depende de pacotes genéricos de otimização (ex: CPLEX), o que permite uma economia de memória ainda maior.

3.2.2 Relaxando Restrições de Atribuição

Uma relaxação lagrangeana alternativa poderia dualizar as restrições de atribuição ao invés das de precedência, como mostrado na seção A.3.2.2. Por dualizar bem menos restrições, tal relaxação poderia obter limites inferiores melhores. Entretanto, a memória necessária para resolver tal relaxação seria relativamente próxima à memória necessária para resolver o modelo original, limitando o tama-

nho das instâncias que podem ser tratadas utilizando tal método. Como o foco deste trabalho é em instâncias de grande porte, a primeira relaxação lagrangeana é preferida.

3.3 O Método do Subgradiente

Para resolver o problema dual lagrangeano associado, é utilizado o método do subgradiente (vide seção A.3.3). Em cada iteração deste método, uma relaxação lagrangeana do problema é resolvido para um vetor de multiplicadores lagrangeanos. Se o limite obtido for melhor que o atual, este é atualizado. De qualquer forma, no final de cada iteração, um vetor subgradiente é calculado, e um passo determinado. Finalmente, o vetor de multiplicadores lagrangeanos é atualizado, realizando um passo na direção do vetor subgradiente. Este processo é repetido até que uma condição de parada seja alcançada. A implementação proposta pára quando o algoritmo estoura um limite de tempo, ou tamanho do passo se torna excessivamente pequeno. O vetor de multiplicadores lagrangeanos é inicializado com um vetor cujas componentes são todas iguais a zero.

3.4 Non-Delayed Relax-and-Cut

Testes preliminares mostraram que, devido ao enorme número de restrições dualizadas, que leva a um vetor subgradiente com muitas componentes, o passo do algoritmo se torna muito pequeno, comprometendo a convergência do método. Para tratar tal problema um algoritmo *Non-Delayed Relax-and-Cut*, como proposto por Lucena (2005), é desenvolvido. Este algoritmo é uma adaptação do método do subgradiente. Ele propõe que somente um pequeno conjunto de restrições violadas, escolhidas de acordo com um critério maximal, sejam dualizadas a cada iteração, de forma que o passo seja menos comprometido pelo tamanho da norma do vetor subgradiente. Veja a seção A.3.4 para um maior detalhamento da implementação proposta.

3.5 Heurística Lagrangeana

Uma vez que testes preliminares mostraram que frequentemente os limites inferiores obtidos para instâncias fáceis provavam a otimalidade de soluções obtidas com as heurísticas desenvolvidas, uma heurística lagrangeana, que busca tornar viáveis as soluções obtidas pela relaxação lagrangeana sem comprometer muito o valor da função objetivo, é proposta na seção A.3.5. Tal heurística utiliza os multiplicadores de lagrange para guiar a construção de uma solução viável, que é então aprimorada por uma busca local.

3.6 Experimentos e Resultados - Parte 2

Finalmente, na seção A.3.6, os algoritmos desenvolvidos são extensivamente testados, a fim de determinar qual método é mais vantajoso para obter limites inferiores para o problema, e para determinar se a heurística lagrangeana é vantajosa em relação às heurísticas baseadas em buscas locais desenvol-

vidas. Os limites inferiores e superiores obtidos são então comparados para analisar a qualidade das soluções viáveis obtidas pelas heurísticas.

3.7 Conclusões Parciais

Para as instâncias testadas (veja o apêndice B), a relaxação linear do modelo indexado no tempo obteve limites muito melhores que os limites de relaxação linear dos outros modelos testados. O algoritmo *Non-Delayed Relax-and-Cut* se mostrou ligeiramente superior ao Método do Subgradiente tradicional. Os limites inferiores obtidos, foram particularmente bons para instâncias com datas de entrega folgadas, e não tão bons para instâncias com datas de entrega apertadas. Entretanto, *gaps* válidos e significativos puderam ser obtidos em todos os casos. Foi possível tratar instâncias com até seis máquinas e 180 tarefas. A heurística lagrangeana, auxiliada por uma busca local, pôde obter soluções competitivas com as heurísticas baseadas em buscas locais, o que reafirma o valor das buscas locais desenvolvidas. As soluções obtidas pela heurística lagrangeana, sem uma busca local, se mostraram também interessantes para instâncias de menor porte. Já para as instâncias de maior porte, o *gap* observado cresce rapidamente a medida que o número de tarefas aumenta.

Veja a seção A.3.7, para uma argumentação mais detalhada sobre as conclusões parciais sobre os métodos utilizados para obter limites inferiores para o problema, e sobre a heurística lagrangeana.

Capítulo 4

Conclusões

Este trabalho, tratou de Problemas de Sequenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Sequência.

Para obter soluções para o problema, foram desenvolvidas três heurísticas: uma baseada em GRASP, uma em VNS e uma híbrida.

As soluções encontradas pelas heurísticas foram pelo menos tão boas quanto as melhores conhecidas, obtidas com métodos previamente desenvolvidos, para todas as instâncias consideradas. Os algoritmos propostos foram muitas vezes mais rápidos que os anteriormente desenvolvidos, obtendo em um tempo razoável, boas soluções para instâncias com até seis máquinas e 300 tarefas. Para o caso especial de sequenciamento em uma única máquina, os algoritmos desenvolvidos conseguiram, em tempo razoável, obter soluções melhores para até 95% das melhores soluções encontradas na literatura.

A utilização de avaliações inteligentes de soluções levou a uma grande melhora de desempenho, deixando as buscas locais bem mais baratas. Dessa forma, a busca local pode ser realizada completamente, garantindo mínimos locais, diferentemente de como era feita em trabalhos anteriores.

A união das vizinhanças consideradas também se mostrou bastante vantajosa, levando a uma busca local que pode obter soluções melhores que as baseadas em somente uma vizinhança, num tempo competitivo.

Com a significativa melhora de desempenho, alcançada com a utilização de avaliação delta parcial das soluções; e qualidade, obtida com a utilização de uma busca local melhor, a utilização de técnicas de intensificação, como *Path-Relinking*, e redução, como o uso de buscas locais truncadas, se tornou desnecessária, levando ainda a um algoritmo de estrutura simplificada e que depende de poucos parâmetros.

O sucesso das implementações apresentadas foi principalmente devido ao uso de uma busca local poderosa, favorecida pela diversidade das soluções analisadas proporcionada pelo desenho da heurística como um todo. A utilização de uma estrutura baseada em GRASP foi importante para prover tal diversificação de soluções iniciais. Já a utilização da estrutura do VNS ajudou a melhorar a qualidade das soluções, introduzindo alguma memória de curto prazo ao algoritmo.

Para medir a qualidade das soluções obtidas, relaxações lineares e de lagrange foram utilizadas para obter limites inferiores para o problema. Para tal, foram desenvolvidos uma versão modificada

do Método do Subgradiente e um algoritmo *Non-Delayed Relax-and-Cut*. A relaxação lagrangeana apresentada também foi utilizada para desenvolver uma heurística lagrangeana.

Dentre as formulações testadas, a relaxação linear da formulação indexada no tempo apresentada obtém os melhores limites, ao custo de uma grande quantidade de memória, não conseguindo obter limites inferiores para instâncias grandes do problema.

Os limites obtidos para instâncias pequenas, com a relaxação linear da formulação indexada no tempo, foram particularmente bons para instâncias fáceis, com datas de entrega folgadas, e não tão bons para instâncias mais complicadas, com datas de entrega apertadas e sequenciamentos mais congestionados. Ainda assim, eles foram significativos e bem melhores que zero, um limite inferior óbvio, que era frequentemente o melhor limite inferior encontrado pela relaxação linear das outras formulações testadas.

Uma vez que a relaxação lagrangeana proposta tem a propriedade da integralidade, os limites obtidos pelo problema dual lagrangeano foram no máximo iguais àqueles obtidos pela relaxação linear da formulação indexada no tempo, muitas vezes iguais ou bem próximos. Como esperado, devido a problemas de convergência dos métodos utilizados para resolver o dual lagrangeano, os algoritmos desenvolvidos foram muitas vezes mais lentos que a relaxação linear, e apresentaram limites um pouco piores. Entretanto, os algoritmos desenvolvidos utilizaram substancialmente menos memória, e puderam obter limites válidos inclusive para instâncias grandes do problema, com até seis máquinas e 180 tarefas.

A heurística lagrangeana desenvolvida conseguiu obter bons resultados, mas ao custo de um grande esforço computacional. Assim, ela pode ainda ser interessante para propor boas soluções para instâncias relativamente grandes, com uma medida formal de sua qualidade.

Dessa forma, o objetivo de medir a qualidade das instâncias foi parcialmente atingido, uma vez que só foi possível obter uma boa medida para as instâncias mais fáceis e não muito grandes, muitas vezes ao custo de um grande esforço computacional, significativamente maior que o necessário para obter a solução avaliada.

4.1 Trabalhos Futuros

Existe ainda muito espaço para pesquisa sobre Problemas de Sequenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Sequência.

Uma vez que a união de duas vizinhanças levou a uma outra mais bem sucedida, pode ser interessante experimentar outras combinações de duas ou mais outras vizinhanças. Além disso, podem ser ainda aplicadas outras técnicas para tornar a busca ainda mais rápida, como reduções de vizinhança. Outros métodos para gerar soluções iniciais, mais elaborados, também podem ser utilizados a fim de aumentar o desempenho dos algoritmos.

Já que os limites inferiores obtidos para instâncias difíceis, utilizando a relaxação lagrangeana proposta, não foram muito bons, várias técnicas podem ser utilizadas para melhorar tais limites, como a pesquisa e uso de outras desigualdades válidas num arcabouço *Relax-and-Cut* clássico, e procedimentos de *lifting*. A obtenção de métodos eficientes, que utilizem pouca memória, para resolver a relaxação lagrangeana com as restrições de atribuição dualizadas também pode ser bastante

interessante.

O método do subgradiente, utilizado para resolver o dual lagrangeano, apresentou uma convergência bastante lenta, principalmente para instâncias difíceis. Assim sendo, experimentar outros métodos para resolver o dual lagrangeano podem também revelar alternativas interessantes. Da mesma forma, outras heurísticas lagrangeanas, mais elaboradas e eficientes, podem ser desenvolvidas.

Finalmente, pode ser interessante utilizar os algoritmos desenvolvidos num esquema de *Branch-and-Cut* para resolver instâncias de menor porte.

Appendix A

Detalhamento

A.1 Introduction

Scheduling problems can be found on several areas of science. In biology, for example, bioinformatics problems model several applications to medicine and genetics. In computer science, problems of information retrieval and computer architecture can also be modeled as scheduling problems.

In the industrial context, scheduling problems often arise in the Manufacturing Resource Planning process. The production programming is made according to a planning horizon. Long term decisions have strategic characteristics and are taken by high administration. The mid-term is known as the tactical level, where the objective is to plan the production in order to minimize manufacturing costs subject to meet the demand and due dates defined on the strategic level. Scheduling problems often arise on the operational level, considering the production of a day, a week or a month; where one of the main objectives is to define a processing sequence while pursuing a specific objective.

Operational decisions affect directly the tactical ones. The latter, in turn, has a great impact on strategic decisions. A good production plan is a key factor to provide an adequate usage of the available resources and to fulfill the desired objective. Thus, it is important to seek good methods to obtain good plans for all levels. This research deals with problems resident on the operational level.

Scheduling problems with parallel machines and sequence dependent setup times, as studied in this work, are known to be difficult to solve. The single machine scheduling problem with sequence-dependent setups is NP-hard (see Pinedo (1995) and Du e Leung (1990)). It has also been proven, for the parallel machine case, that the problem of minimizing the makespan with two identical machines is NP-Hard (see Garey e Johnson (1997) and Lenstra et al. (1977)). Using a reduction from the 3-Dimensional Matching Problem (see Garey e Johnson (1997)), Crama e Spieksma (1993) proved that, even when all jobs have equal lengths, the single machine scheduling problem, considering the minimization of the total processing cost (a function of the start time, such as the weighted tardiness or slackness), is strongly NP-Hard. Therefore, the more complex case of minimizing the total weighted tardiness, makespan or total slackness on a scheduling problem with m unrelated parallel machines, due dates, and machine and sequence dependent setup times, is also strongly NP-Hard since it is a generalization of one of the previous cases.

A.1.1 Motivation

Consider a factory with a set of jobs that have to be processed, without preemption, on any machine of a set of unrelated parallel machines. Each job may have different processing times for each machine, and sequence-dependent setup times, also for each machine.

The problem studied in this work is based on a real case of a particular plant of a refractory brick factory. It produced more than 8000 products, all of them with the same production sequence consisting in 5 stages: material input measurement, blending, conformation, heat treatment and packaging. In each of these stages, the plant has more than one machine, and most of the products can be processed by any machine at a particular stage.

In this scenario, the conformation stage is the most critical part of the production, and is closely related to the quality of the final product. Moreover, the production plan of the plant is made according to the conformation stage's planning. After the conformation stage's planning is defined, the other stages' planning are adjusted to meet its objectives. For this reason, most of the planning effort is invested on this stage. This work will focus on the conformation stage, which is modelled a Scheduling Problem with Unrelated Parallel Machines.

The conformation stage of the studied plant uses seven machines: four identical, which require manual work, two fully automatic and the last one designed for a specific kind of bricks. The production plan is made considering the unrelated parallel machines case with the first six machines and a single machine case for the seventh, both considering sequence-dependent setup times and due dates. A more detailed explanation can be found on Ravetti (2007).

A.1.2 Scheduling Problems with Parallel Machines and Sequence-Dependent Setup Times

Consider a set of unrelated machines $\mathscr{M} = \{1,2,..,|M|\}$ and a set of jobs $\mathscr{J} = \{1,2,..,|J|\}$.

There are several kinds of scheduling problems to suit the most diverse applications in the real world. The most specific case, is the single machine scheduling problem, where a set of jobs has to be scheduled on a machine, that is, the problem is to define the job processing order on that machine.

When there are more than one machine available at any time, working in parallel, the problem is modeled as a scheduling problem with parallel machines. In this model, any machine can process any job, and every job has to be processed by exactly one machine. These machines may be either identical, uniform or unrelated. On the identical machines case, all machines have the same processing time for any job. The processing time on uniform machines depends only on the machine, while the processing time on unrelated machines depends both on the job and machine.

If the jobs may have to be processed by one or more machines, "shop" models are used. On Flow Shop problems, every job has the same machine path. That is, they have to be processed first by machine 1, then machine 2 and so on. If there is a set of machines that can process the jobs instead of a single one at each stage, the problem is known as Flexible Flow-Shop. If every job has its own specific sequence of machines to visit, then the model is known as Job Shop. Finally, if all jobs have to be processed by all machines, but does not have to respect any specific sequence at all, the model is known as Open Shop.

There are also several characteristics that may be associated with jobs, such as release dates, sequence-dependent setup times, preemption, precedence restrictions, machine breaks, eligibility restrictions, permutations, blocks, recirculation and many others.

There are also many objectives that may be pursued, depending on the application. Some of the most common objectives are to minimize the makespan, the total weighted tardiness, the total slackness and the maximum lateness.

For a review on scheduling problems, and classical approaches for them, refer to Pinedo (1995), Lee e Pinedo (2002), Blazewicz et al. (1996) and Brucker (2004).

The scheduling problems with parallel machines tackled in this work consist in assigning a machine for each job, and sequencing the set of jobs assigned to each machine considering a given objective function. Every job $j \in \mathscr{J}$ has exactly one associated operation, which can be performed by any machine $m \in \mathscr{M}$. Preemption is not allowed.

In the realistic problem studied in this work, positive weights w_j are associated to each job $j \in \mathscr{J}$. In order to process a job j immediately after a job j' on machine m, it is required a positive setup time $s_{j'jm}$, that depends both on the sequence of jobs $j' \prec j$ [1] and on the machine $m \in \mathscr{M}$ where they are processed; and a positive processing time p_{jm}, that depend only on machine $m \in \mathscr{M}$ where job j is to be processed.

Define the completion time of job j recursively as $C_j = C_{j'} + p_{jm} + s_{j'jm}$, where $C_{j'}$ is zero if job j is the first job scheduled or the completion time of the previous job $j' \prec j$ otherwise. The tardiness τ_j of each job is calculated as $\tau_j = max(C_j - d_j, 0)$, where d_j is the due date of job j. Let C_{max} be the completion time of the last job to finish processing ($C_{max} = max_{j \in \mathscr{J}}\{C_j\}$). All processing must be done within a stipulated time horizon $T > C_{max}$. In this work, $T = \infty$, unless explicitly told otherwise. It is assumed that all jobs and machines are available at the base-time t=0.

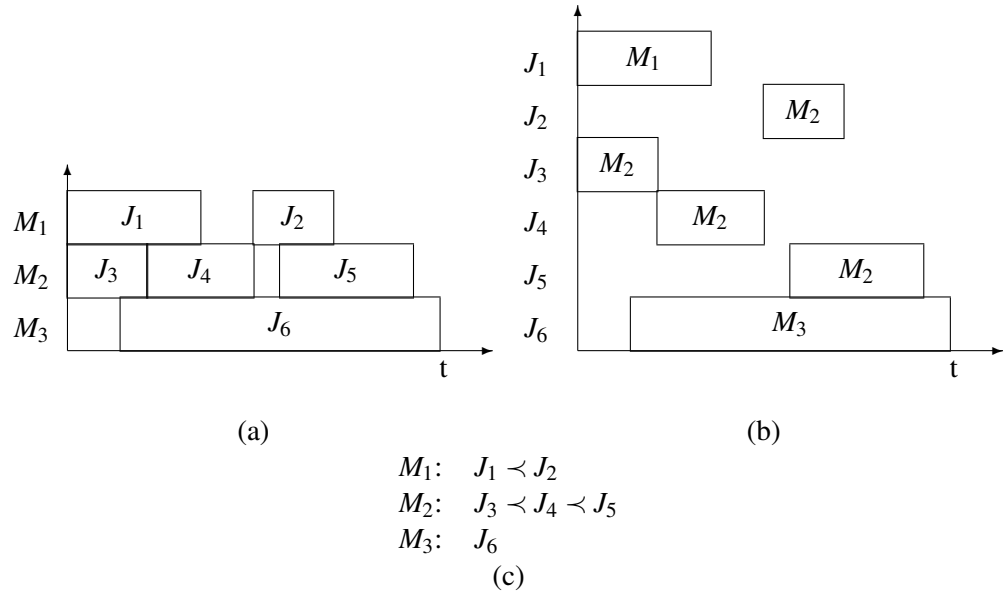
The different goals, considered separately in this research, are the minimization of the total weighted tardiness (TWT), $min \sum_{j=1}^{n} w_j \tau_j$; the minimization of the makespan (MKS), $min\ C_{max}$, and the minimization of the total slackness (TS), $min \sum_{j=1}^{m} (C_{max} - C_j)$.

Summarizing, the problems studied in this work are Scheduling Problems with Unrelated Parallel Machines, with Machine and Sequence-Dependent Setup Times and Due Dates. The main objective is to minimize the Total Weighted Tardiness, but other objectives, such as the minimization of the Makespan and Total Slackness, are discussed. These problems are also known as $R|s_{jj'm}, \tilde{d}_j| \sum w_j \tau_j$, $R|s_{jj'm}, \tilde{d}_j|C_{max}$ and $R|s_{jj'm}, \tilde{d}_j| \sum_{j=1}^{m} (C_{max} - C_j)$, respectively. Refer to Brucker (2004) for more information on notation for scheduling problems.

Schedules are often represented by Gantt charts, which may be job or machine oriented (see figure A.1 (a) and (b)). In this work, a simplified version of the machine oriented Gantt chart, which considers only the scheduling order instead of time positions, will also be used (see figure A.1 (c)). Since usually $|\mathscr{M}| \lll |\mathscr{J}|$, machine oriented simplified Gantt charts will be preferred because they are more compact, and provide a more intuitive view of the whole schedule. Idle time periods are not considered. That is convenient for scheduling problems with parallel machines without release dates, since solutions with idle time periods can be easily improved by anticipating jobs that start after an idle period until all machine are always working (or being worked on, if setup times are considered)

[1] A job $j \prec j'$ is scheduled immediatly before another job j'.

Figure A.1: Example of machines (a) and job oriented Gantt charts (b) on job sequences.



M_1: $J_1 \prec J_2$
M_2: $J_3 \prec J_4 \prec J_5$
M_3: J_6

(c)

at any time before its makespan.

The following notation is used throughout the text:

- j: A particular job in $\mathscr{J}$.
- m: A particular machine in $\mathscr{M}$.
- T: The planning horizon. Consider that a planning horizon of size T have $T+1$ discrete units of time, that is, jobs can be scheduled at times $t = 0, ..., T$.
- t: A particular time period, $0 \leq t \leq T$.
- $|\mathscr{S}|$: The cardinality of set $\mathscr{S}$. That is, the number of jobs or machines, if $\mathscr{S} = \mathscr{J}$ or $\mathscr{S} = \mathscr{M}$, respectively.
- p_{jm}: Processing time of job j on machine m.
- $s_{jj'm}$: Setup time needed to process job j' immediately after job j on machine m.
- $\delta_{jj'm}$: $p_{jm} + s_{jj'm}$
- r_{jm}: The release date of job $j \in \mathscr{J}$ for machine $m \in \mathscr{M}$. That is, the processing of job j can only start on machine m at time $t = r_{jm}$ or later.
- d_j: The due date of job j. The processing of job j has to finish until time $t = d_j$. Otherwise a penalty proportional to the job's weight and its tardiness, $w_j * \tau_j$, will be considered in the objective function.
- w_j: The weight of job j.
- C_j: The completion time of job j.
- C_m: The completion time of the last job processed by machine m.
- C_{max}: The greatest completion time of all jobs, $max_{j \in \mathscr{J}}\{C_j\}$.
- τ_j: The tardiness of job j. $\tau_j = max\{0, C_j - d_j\}$.
- M: a very large constant.

A.1.3 Text Organization

On chapter A.2, three local-search based heuristics are proposed. The first two are based on two very well known metaheuristics: GRASP and VNS. The third is an hybrid approach. The constructive methods used in these heuristics, to generate initial solutions are detailed on section A.2.1. The local searches used are detailed on section A.2.2.2. Implementation aspects are discussed in greater detail on section A.2.2.1. On section A.2.3 these implementations are tested and the obtained results are compared to other results found on the literature. Partial conclusions are then found on section A.2.4.

In order to measure the quality of the obtained solutions, on chapter A.3, three methods to obtain lower bounds for the problem are detailed. The first methods are the linear programming relaxations of three Integer Programming formulations of the problem. On section A.3.1, the linear programming relaxation of a time-indexed formulation, proposed by Ravetti (2007), is compared with the linear programming relaxations of two other formulations, based on Wagner (1959)'s and Manne (1960)'s works. The second, is a lagrangean relaxation of the same time-indexed formulation. This method was previously suggested in Ravetti (2007), but only preliminary results for small problems, obtained with an implementation based on third party optimization packages, were presented. Since Ravetti (2007)'s preliminary results seemed quite interesting, an specific implementation, that does not rely on any third party optimization package, of the suggested method is developed on section A.3.2.1. To address convergence issues of Ravetti (2007)'s method, a novel Non-Delayed Relax-and-Cut algorithm is developed and presented on section A.3.4. On section A.3.5, a novel Lagrangean Heuristic, also based on Ravetti (2007)'s lagrangean relaxation, is proposed. Finally, on section A.3.6 the developed algorithms are extensively tested, and partial conclusions are summarized on section A.3.7.

Chapter A.4 concludes this work and suggests future research on this problem.

A.2 Heuristics Developed

Several papers on the literature deal with scheduling problems with parallel machines, optimizing over several different objectives and considering several problems characteristics.

Ibarra e Kim (1976) considered the minimization of the makespan on scheduling problems with unrelated parallel machines, without setup times or due dates, using constructive heuristics that still remain a basis of comparison to current research methodologies designed to solve the same problem. Martello et al. (1997) tackled the same problem using an approximation algorithm that uses lower bounds obtained by a lagrangean relaxation. Horowitz e Sahni (1977), De e Morton (1980), Davis e Jaffe (1981), Serna e Xhafa (2002), Jansen e Porkolab (1999) and Mokotoff e Chrétienne (2002) developed different polynomial time approximation schemes for the same problem. Glass et al. (1994) evaluated the performance of several local search algorithms, namely, a gradient search, simulated annealing, tabu search, genetic algorithm and genetic descent algorithms. Srivastava (1998) developed a hashed tabu search for the same problem.

An also very common heuristic approach found on the literature is to use a two-stage algorithm, where a linear program is solved on the first stage, and then a heuristic to schedule the infeasible or unscheduled jobs is ran. Potts (1985), Hariri e Potts (1991) and Lenstra et al. (1990) developed such

algorithms.

Suresh e Chaudhuri (1994) tackled the unrelated parallel machines problem, minimizing the maximum tardiness without setup times, with three heuristics. They used an improvement heuristic that exploited pairwise job swaps, obtaining good results specially for instances with loose due dates. Musier e Evans (1989) also used a pairwise job swap local search to improve solutions, obtained randomly, for the scheduling problem with identical parallel machines, minimizing the total tardiness.

Adamopoulos e Pappis (1998) considered the problem of scheduling on unrelated parallel machines, still without considering setup times, minimizing the total earliness and tardiness, using a constructive heuristic which assigns jobs to machines deterministically, and then improves the resulting solutions solving scheduling problems with a single machine. Bank e Werner (2001) considered the same problem with release times, and tackled it with several constructive heuristics, neighbourhood search and simulated annealing. For the neighborhood search, they recommended the neighbour generation by a shift of one job from a machine to the set of another.

A problem that has also been very studied is that in which there is a set of jobs to be processed on a set of parallel machines, and the loading of a job on a machine (the setup time period) is performed by a single server. This setup time can not be performed while a machine is processing a job. On the other hand, the machine can process a job without the server being present after the job is loaded on the machine. Moreover, simultaneous requests of the server results in machine idle time.

Kravchenko e Werner (1997) studied the case with identical parallel machines, constant setup times and a single server, minimizing the makespan. They proved it to be NP-Hard and presented polynomially solvable cases. Kravchenko e Werner (2001) considered the same problem, only considering unit setup times and the minimization of the total completion time. They presented a heuristic with an absolute error bound.

Koulamas (1996) proposed an efficient beam-search heuristic to minimize the machine idle time on two parallel machines with sequence independent setup times, also with a single server. Abdekhodaee e Wirth (2002) tackled the same problem, minimizing the makespan under a specific assumption of alternating job processing, using an Integer Programming formulation, and presented polynomially solvable cases. Abdekhodaee et al. (2004) addressed the special case of equal processing and setup times of the same problem using several heuristics. Abdekhodaee et al. (2006) tackled the general case using greedy heuristics and a genetic algorithm. Hall et al. (2000) and Brucker et al. (2002) provided several complexity results for special cases of the same problem, proving NP-Hardness for some objective functions, NP-Hardness in the strong sense for others; and presented polynomially solvable cases. Glass et al. (2000) presented complexity results for the same problem, but with dedicated machines, and proposed approximation algorithms.

Heady e Zhu (1998) addressed the case with identical parallel machines and sequence-dependent setup times, where some machines are not able to process some jobs, minimizing the sum of earliness and tardiness costs. They proposed an Integer Programming formulation and heuristics. Ventura (2000) presented a mathematical programming formulation and a simulated annealing heuristic for the same problem, and showed polynomially solvable cases.

Mendes et al. (2002) e Gendreau et al. (2001) tackled the same problem, minimizing the makespan, using tabu search and a memetic algorithm. Gendreau et al. (2001) also proposed lower bounds for

the problem and a divide-and-merge heuristic. Tahar et al. (2006) addressed the same problem, also minimizing the makespan, but considering job splitting, using a linear programming based heuristic.

Hurink et al. (2001) also considered the problem with identical parallel machines, minimizing the makespan with precedence constraints, providing complexity results when considering the problem in two parts: partitioning and sequencing. They concluded that the problem can not be solved considering decisions based on one of these parts and solving the remaining problem afterwards.

Weng et al. (2001) considered the problem with unrelated parallel machines and sequence-dependent setup times, minimizing the total weighted completion times, using seven simple heuristics. Park et al. (2000) used a neural network to obtain values for parameters in a priority rule to minimize the total weighted tardiness on the case with identical machines and sequence-dependent setup times.

Zhu e Heady (2000) solved small instances of the scheduling problem with unrelated parallel machines and sequence-dependent setup times, minimizing the total weighted earliness plus the total weighted tardiness, using an Integer Programming formulation. Sivrikaya-SerifogIu e Ulusoy (1999) tackled the same problem on uniform parallel machines using genetic algorithms. Balakrishnan et al. (1999) solved small instances of the same problem using a Mixed-Integer Programming formulation.

Anglani et al. (2005) considered the case with identical parallel machines and sequence-dependent setup times, where the job processing times are uncertain, using a fuzzy mathematical programming approach and an Integer Linear Programming model.

Approximation algorithms for the batch scheduling with sequence independent setup times were developed by Brucker et al. (1998). Lenstra et al. (1990) and Gairing et al. (2007) proposed approximation algorithms for the case with unrelated parallel machines and no setup times. Ghirardi e Potts (2005) proposed a Recovery Beam Search approach, also for the case with unrelated parallel machines and no setup times, obtaining good solutions for instances with up to 50 machines and 1000 jobs within reasonable time when minimizing the makespan. Hoon Lee e Pinedo (1997) proposed a three phase algorithm, consisting in statistical instance evaluation, an adapted ATCS (Apparent Tardiness Costs with Setups) constructive heuristic and a Simulated-Annealing-based heuristic also for the, with identical parallel machines and sequence-dependent setup times, obtaining good solutions for instances with up to 2 machines and 360 jobs within reasonable time. This heuristic was later extended by Pfund et al. (2008) to allow non-ready jobs to be scheduled. Armentano e Yamashita (2000) proposed a Tabu Search algorithm for the case with identical machines and no setup times, obtaining good solutions within reasonable time for up to 150 jobs and 10 machines, when minimizing the mean tardiness. Logendran et al. (2007) tackled the case with sequence-dependent setup times using Tabu Search, obtaining good solutions within reasonable time for instances with up to 15 machines and 60 jobs, when minimizing the total weighted tardiness.

For a introductory survey on the state of the art of scheduling problems with parallel machines up to 1990, see Cheng e Shin (1990). For a survey on Scheduling Problems with a common due date, see Gordon et al. (2002). For a survey on Scheduling Problems with Unrelated Parallel Machines, without setup times, preemptions or any other side-conditions, see Pfund et al. (2004). For a survey on Scheduling problems with setup times or costs, see Allahverdi et al. (2008).

Most of the studies on scheduling problems consider a single machine and the objective of minimizing the makespan. Most of the papers that address the parallel machine case consider either

identical or uniform machines. Compared to other areas of deterministic scheduling, there are still few studies on unrelated parallel machine problems, and even less considering due dates and the objective of minimizing the total weighted tardiness (see Pfund et al. (2004)). There are even less papers on the minimization of the total weighted tardiness on unrelated parallel machines, considering due dates and sequence-dependent setup times.

Instances of practical interest, considering the real case, usually have at least 100 jobs and 6 different machines. The exact approaches known up to the time of this research can solve instances with up to six machines and 25 jobs. Since none of them is able to solve large instances within reasonable computational time, it is interesting to seek heuristic approaches able to obtain good quality solutions faster. In the real case, the company takes up to 3 days to define the whole month's sequencing.

Ravetti et al. (2007) proposed a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic with Path-Relinking, used later by Rocha et al. (2008) as an upper bound for a Branch-and-Bound algorithm. This heuristic considered only a job-swap neighborhood, used the Earliest Due Dates (EDD) heuristic to obtain initial solutions, and direct complete evaluation of the movements. Since a new solution was obtained from an existing one by complete reconstruction, and accepted if better, local searching was very expensive and had to be stopped before guaranteeing a local minimum. The algorithm stopped arbitrarily after 1000 iterations. It could obtain reasonably good solutions for instances with up to 150 jobs and 6 machines within reasonable time.

de Paula et al. (2007) improved Ravetti et al. (2007)'s heuristic implementing a job-insertion neighborhood, different local search procedures and the NEH heuristic (see section A.2.1) for the parallel machines problem. They also proposed a Variable Neighbourhood Search (VNS) algorithm using both swap and insertion neighbourhoods and multiple local searches, which made the algorithm significantly faster. However, since the implementation still used direct complete evaluation of new solutions, local searching was also still very expensive and had to be truncated. The algorithm also stopped arbitrarily after 1000 iterations. It could obtain good solutions for instances with up to 200 jobs and six machines within reasonable time. The solutions obtained for small instances using the VNS implementation were usually worse than those obtained using GRASP.

Cicirello (2003) provided a set of benchmark instances for the single machine case, and tackled the problem using several constructive, local search and hybrid heuristics. The obtained results were improved later (Cicirello e Smith (2005)) using randomized Heuristic-Biased Stochastic Sampling, and then improved even more using Simulated Annealing (Cicirello (2006)). Anghinolfia e Paolucci (2007) also tackled the single machine case using Particle Swarm algorithms, improving all of Cicirello's results.

This chapter describes efficient GRASP, VNS and hybrid heuristics that use partial delta evaluation of the perturbations: they evaluate only the part that differs new solutions from the current, aborting evaluation as soon as it can be deduced that it will not be better. The result is a great improvement of the heuristics performance.

A.2.1 Constructive Heuristics

In this work, a pure random algorithm and two constructive heuristics are considered to quickly obtain initial solutions: the Earliest-Due-Dates (EDD) heuristic and the Nawaz-Enscore-Ham (NEH)

heuristic.

A.2.1.1 Random algorithm

In $O(|\mathscr{J}|)$, the lgorithm 1 simply creates a new solution by randomly selecting and inserting unscheduled jobs at the end of the scheduling queue of a random available machine. The aim of this heuristic is only to introduce initial solution diversity to the algorithms, without considering quality.

Algorithm 1: Random Solution Construction

Input: A sorted list $\mathscr{J}$ of jobs
Output: A feasible random solution
for $j = 1$ *to* $|\mathscr{J}|$ **do**
    Choose a machine m at random; Schedule $\mathscr{J}[j]$ at the end of m's processing queue;
return *Solution*

A.2.1.2 EDD

The Earliest-Due-Dates heuristic was first proposed for the permutational Flow-Shop Scheduling Problem. Ravetti (2007) used it to generate initial solutions for his GRASP heuristic for the Scheduling problem with Parallel Machines and Sequence-Dependent Setup Times.

The EDD heuristic attempts to obtain solutions of better quality by inserting jobs sequentially at the end of the most interesting machine's scheduling queue. The proposed implementation considers that the machine that will lead to the least objective function value increase is the most interesting one. It's important to notice that different job insertion sequences may lead to different solutions, thus some diversification can still be achieved.

This heuristic runs in $O(|\mathscr{J}||\mathscr{M}|)$ time. Since usually $|\mathscr{M}| \lll |\mathscr{J}|$, it tends to be reasonably fast.

Algorithm 2: *EDD*

Input: A sorted list $\mathscr{J}$ of jobs
Output: A feasible solution
for $j = 1$ *to* $|\mathscr{J}|$ **do**
    $BestMachine \leftarrow 0$;
    $BestObjective \leftarrow \infty$;
    foreach *Machine* m **do**
        $Objective \leftarrow$ objective function of the partial solution assuming that j is scheduled at the end of m's processing queue;
        if $Objective < BestObjective$ **then**
            $BestMachine \leftarrow m$;
            $BestObjective \leftarrow Objective$;
    Schedule $\mathscr{J}[j]$ at the end of $BestMachine$'s processing queue;
return *Solution*

A.2.1.3 NEH

The NEH heuristic was first proposed for the Permutational Flow-Shop Scheduling Problem by Nawaz et al. (1983), and then used by de Paula et al. (2007) for the Parallel Machines Scheduling Problem. Algorithm 3 presents de Paula et al. (2007)'s version.

It attempts to obtain solutions of even better quality by inserting jobs sequentially at the most interesting position of the partial schedule. The proposed implementation considers that the position, considering all available positions of all machines, that will lead to the least objective function value increase is the most interesting one.

Different job insertion sequences also lead to different solutions, although it is expected that the impact of the job insertion order will be smaller on solutions obtained by NEH than on those obtained by EDD, since more partial solutions are evaluated. For the same reason, it is expected that the solutions obtained using this method will be better than those obtained by EDD.

The NEH heuristic runs in $O(|\mathscr{M}||\mathscr{J}|^2)$ time.

Algorithm 3: *NEH*

Input: A sorted list $\mathscr{J}$ of jobs
Output: A feasible solution

```
1  foreach j = 1 to |J| do
2      BestMachine ← 0;
3      BestPosition ← 0;
4      BestObjective ← ∞;
5      foreach Machine m do
6          foreach Position p available at m do
7              Objective ← objective function of the partial solution assuming that j is scheduled
               at position p of m's processing queue;
8              if Objective < BestObjective then
9                  BestMachine ← m;
10                 BestPosition ← p;
11                 BestObjective ← Objective;
12     Schedule J[j] at position BestPosition of BestMachine's processing queue;
13 return Solution
```

A.2.2 Local Search Based Heuristics

A.2.2.1 Solution Movements and Neighbourhoods

The efficient implementation of local search procedures is the key for the success of several local search based algorithms. Two implementation issues are determinant in the efficiency of a local search procedure: the number of neighbour solutions evaluated and the fast computation of the neighbour solutions' costs (see Lin e Kernighan (1973), Glover e Laguna (1997) and Ribeiro et al. (2008), for examples on the Travelling Salesman Problem, Single Machine Scheduling and Car-Sequencing Problems, respectively). In this section, efficient implementations for the local search procedures used in this work are described.

Feasible solutions of scheduling problems with parallel machines are basically processing queues for all machines in which all jobs are scheduled exactly once. However, several other items have to be considered to compute the objective function value of the solution. The completion times, for example, are common to all three objective functions considered. They are needed to compute the tardinesses for the TWT, the makespan or the TS. Since only a small portion of the solution is changed after a movement, other values calculated can also be saved to avoid recalculation. The proposed implementation saves the jobs' completion times and penalties. The machine and position in where the jobs are currently scheduled and the machines' sequencing as an array of queues are also kept. The two positioning methods are used to determine where a certain job is, or what job is in a certain position of a certain machine, in constant time.

Swap and insertion moves (and neighbourhoods) were used in several other successful approaches to scheduling problems, particularly those based on metaheuristics. Such moves and neighbourhoods are found in the Tabu Search algorithms developed by Logendran et al. (2007) and Ümit Bilge et al. (2004), in the mutations of the Genetic Algorithm developed by Abdekhodaee et al. (2006), in the Simulated Annealing developed by Cicirello (2006), in the improvement heuristic proposed by Rabadi et al. (2006), in the GRASP algorithms proposed by Rocha et al. (2004) and de Paula et al. (2007), in the VNS algorithm proposed by de Paula et al. (2007) and in many others.

Let a n-swap be the neighborhood defined by n sequential (different) job swaps. A job swap between two jobs consists of swapping their positions on the scheduling. For example, if a job j is at position p of machine m and job j' is at position p' of machine m', after a *1-swap(j,j')*, job j would be at position p' of machine m' and job j' would be at position p of machine m.

Let a n-insertion be the neighborhood defined by n sequential (different) job insertions. A job insertion consists of moving a job from one position on the scheduling to another. For example, if job j is at position p of machine m, after a *1-insertion(j,p',m')*, job j would be at position p' of machine m'. All jobs from position $p+1$ of machine m to the last position of the same machine would also be moved one position behind on the same machine, and all jobs from position p' of machine m' to the last position of the same machine would also be moved one position ahead on the same machine.

In this work job *swap* and *insertion* are used as synonyms for *1-swap* and *1-insertion*, respectively. It is noteworthy that the proposed implementations perform both forward and backward operations (e.g.: a 1-insertion may move a job from a position prior or after the current) and both on different or the same machines (e.g.: swap one job from one machine A and with another from another machine $B \neq A$, or swap two jobs resident on the same machine), whenever applicable.

Partial Movement Evaluation After a movement, it is not necessary to update the information of all jobs, but only of those affected by it. Particularly, for any movement, the jobs resident on machines other than those involved in the movement won't be affected, and thus don't have to be updated. Moreover, for each job insertion or swap, it is possible to calculate, in constant time, exactly two completion time difference constants for exactly two disjoint job sequence intervals that include all the jobs affected by the movement. Therefore, in order to update the jobs' completion time, it is sufficient to add the respective completion time difference constant to the completion times of the jobs comprised on both job sequence intervals defined. Figures A.2 (a)-(e) illustrate the affected jobs on

Figure A.2: Job movements considered in this work. The movement is illustrated by the curved arrows. The job(s) moved are circled. The braces delimiter the jobs by the movement with completion time difference constant K_n, detailed on table A.1. The affected jobs index are those of the scheduling before the movement.

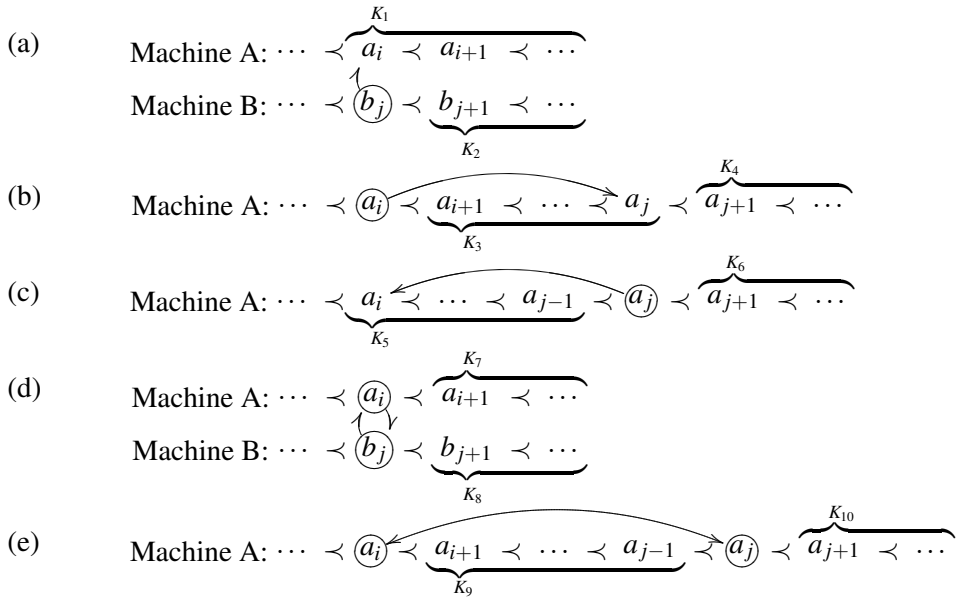


Table A.1: Completion time difference constants. Since all values used are based on the schedule before the movement, they are already known and can be easily deduced from figures A.2 (a)-(e).

K_1:	$p_{b_jA} - s_{a_{i-1}a_iA} + s_{a_{i-1}b_jA} + s_{b_ja_iA}$
K_2:	$-p_{b_jB} - s_{b_{j-1}b_jB} - s_{b_jb_{j+1}} + s_{b_{j-1}b_{j+1}B}$
K_3:	$-p_{a_iA} - s_{a_{i-1}a_iA} - s_{a_ia_{i+1}} + s_{a_{i-1}a_{i+1}A}$
K_4:	$K_3 + p_{a_iA} - s_{a_ja_{j+1}A} + s_{a_ja_iA} + s_{a_ia_{j+1}A}$
K_5:	$p_{a_jA} - s_{a_{i-1}a_iA} + s_{a_{i-1}a_jA} + s_{a_ja_iA}$
K_6:	$K_5 - p_{a_jA} - s_{a_{j-1}a_jA} - s_{a_ja_{j+1}A} + s_{a_{j-1}a_{j+1}A}$
K_7:	$-p_{a_iA} - s_{a_{i-1}a_iA} - s_{a_ia_{i+1}A} + p_{b_jA} + s_{a_{i-1}b_jA} + s_{b_ja_{i+1}A}$
K_8:	$-p_{b_jB} - s_{b_{j-1}b_jB} - s_{b_jb_{j+1}B} + p_{a_iB} + s_{b_{j-1}a_iB} + s_{a_ib_{j+1}B}$
K_9:	$-p_{a_iA} - s_{a_{i-1}a_iA} - s_{a_ia_{i+1}A} + p_{a_jA} + s_{a_{i-1}a_jA} + s_{a_ja_{i+1}A}$
K_{10}:	$K_9 - p_{a_jA} - s_{a_{j-1}a_jA} - s_{a_ja_{j+1}A} + p_{a_iA} + s_{a_{j-1}a_iA} + s_{a_ia_{j+1}A}$

each movement considered, assuming that the jobs to be moved are first removed from their original positions simultaneously, and then inserted at the desired positions, in order of start times. The braces indicate the two mentioned job sequence intervals and their respective completion time difference constants Kn, summarized on Table A.1.

Delta Movement Evaluation One of the key factors to improve performance on local search based heuristics is to quickly determine whether a movement will improve the solution or not. In some situations, this can be done by following some simple observations on the completion times.

First, note that if a completion time difference constant is positive, all jobs comprised in its associated sequence interval will be delayed, that is, their completion times will increase. Similarly, if a completion time difference constant is negative, all jobs comprised in its associated sequence interval will have their completion times reduced. Analysing differences between solutions to determine

which is better is a technique called *delta evaluation*.

When trying to minimize the makespan, the new solution can only be better than the current if at least one of the machines involved on a movement is one of the bottlenecks of the solution, that is, the machine with the highest makespan, and its makespan decreases. Different solutions with the same objective function values are discarded to avoid cycling. If the bottleneck machine's makespan does not change or increases, the new solution certainly will not be better than the current. Thus it is not necessary to update the affected jobs' information (completion times, penalties, etc.). If a machine, other than the bottleneck, has its makespan increased, then it might become the new bottleneck, resulting in a worse solution. It is important to notice that a machine's makespan value difference equals its associated completion time difference constant when the movement involves jobs on different machines, and the sum of the two completion time difference constants involved when the movement involves only one machine.

From the TS definition,

$$\sum_{m \in |\mathscr{M}|} (C_{max} - C_m) = \tag{A.1}$$

$$\sum_{m \in |\mathscr{M}|} C_{max} - \sum_{m \in |\mathscr{M}|} C_m = \tag{A.2}$$

$$|\mathscr{M}| * C_{max} - \sum_{m \in |\mathscr{M}|} C_m = \tag{A.3}$$

$$(|\mathscr{M}| - 1) * C_{max} - \sum_{m \in |\mathscr{M}|, m \neq bottleneck} C_m \tag{A.4}$$

the TS difference equals $|\mathscr{M}| * (C^{new}_{bottleneck} - C^{old}_{bottleneck}) - Ka - Kb$ where Ka and Kb are the movement's associated completion time difference constants. Therefore, when minimizing the TS, if a movement involves the (only) bottleneck machine of the solution and increases its makespan, the new solution will certainly be worse than the current one, and no more calculation is needed. Otherwise it will still be necessary to update all jobs' information to calculate the new TS and determine whether the solution is improving or not.

Finally, when minimizing the total tardiness (with unit or uniform weights), if both associated completion time difference constants are positive, all jobs affected by the movement will be delayed, increasing the weighted tardiness and, consequently, resulting in a worse solution. In this case it is also safe to abort the movement evaluation.

A.2.2.2 Local Searching

The proposed implementation uses first-improvement local searches based on job insertions, job swaps, and their union. Considering the union of two neighbourhoods is usually a good idea since it is guaranteed that its local minima are local minima of both neighbourhoods.

Any swap can be implemented with two insertions. This alternative, however, would need twice the operations needed by two job insertions. When considering the union of the job insertion and swap neighbourhoods, the only intersection between them are the movements between adjacent jobs.

The Insertion Search is more interesting when the current solution is not balanced, that is, there are some machines with high makespan and others with low makespan; while the Swap Search is

usually best to improve balanced solutions.

The three local searches are detailed by Algorithms 4, 5 and 6.

Algorithm 4: *Insertion Search*

```
1  while The objective function value improves do
3      foreach job j do
4          foreach machine m do
5              foreach position p of machine m, p ≠ current position of j and m = the current
               machine in which j is scheduled do
6                  if insertion(j,p,m) improves the objective function value then
7                      insertion(j,p,m);
8                      goto line 3;
```

Algorithm 5: *Swap Search*

```
1  while The objective function value improves do
3      foreach job j do
4          foreach job j' = j+1 to n do
5              if swap(j,j') improves the objective function value then
6                  swap(j,j');
7                  goto line 3;
```

Algorithm 6: *Swap ∪ Insertion Search*

```
1  while The objective function value improves do
3      foreach job j do
4          foreach job j' = j+1 to n do
5              if swap(j,j') improves the objective function value then
6                  swap(j,j');
7                  goto line 3;
8          foreach machine m do
9              foreach position p of machine m, p is not adjacent to the position of j do
10                 if insertion(t,p,m) improves the objective function value then
11                     insertion(t,p,m);
12                     goto line 3;
```

Preliminary tests have shown that first-improvement local search implementations were slightly faster than best-improvement and produced solutions with the same average quality.

Two simple observations make it possible to reduce the number of solution evaluations needed, which may be very significant in practice.

The first observation is that a movement that does not change any job positions obviously won't improve the current solution. Next, job swaps are symmetric, that is $swap(a,b,m) = swap(b,a,m)$.

Thus, if $swap(a,b,m)$ did not improve the current solution, a $swap(b,a,m)$ certainly also will not. Next, any job insertion between adjacent jobs is equivalent to a job swap between the same jobs. Thus, job insertions involving adjacent jobs are also symmetric.

Let $m = |\mathscr{M}|$ and $n = |\mathscr{N}|$. Considering these observations, it is only necessary to evaluate up to $\sum_{i=1}^{n}(m+n-1-1) = \sum_{i=1}^{n} m + \sum_{i=1}^{n} n - \sum_{i=1}^{n} 2 = mn+n^2-2n$ insertions, instead of $n*(m+n-1) = mn+n^2-n$; and up to $\sum_{i=1}^{n}(n-i)-n = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i - n = n^2 - \frac{n*(n+1)}{2} = \frac{n^2+n}{2} - n = \frac{n^2-n}{2}$ swaps instead of n^2.

Because of the sizes of the neighbourhoods and the proposed reductions, Swap Searching is faster than Insertion Searching in practice. Therefore, it is possible to exploit the first-improvement structure of the Swap $\cup$ Insertion Search to make it potentially faster by Swap-Searching before Insertion-Searching. Hopefully, the Swap Search will find good solutions quickly, leaving few improvements to be done by Insertion-Searching, as most of the insertions would quickly be discarded using partial evaluation.

Note that the local search stage is the most complex part of a local search based heuristic. Thus, fast movements and solution evaluations, which compose the local search, are a key factor to the success of the heuristic. Techniques like partial and delta solution evaluation, described on sections A.2.2.1 and A.2.2.1, respectively; are then very interesting from the local search point of view. If these techniques, or others, cannot be applied, there are other techniques that can speed up the local search, such as neighborhood reductions (see Hoos e Stützle (2004)) and truncated local searches.

For example, since preliminary testing showed that good results were obtained on the first local search iteration, or improved very little after the first iteration; and because it was not possible to use delta and partial solution evaluation to minimize the multi-objective function considered, Ravetti (2007), Rocha et al. (2008) and de Paula et al. (2007) used truncated local searches to reduce the computation time .

A.2.2.3 Initial Solutions

All proposed heuristics use the same basic scheme detailed by Algorithm 7. Any method capable of generating a feasible solution would be sufficient for the first part of our algorithms. However, as shown by Johnson et al. (1989) and Matsuo et al. (1989), a good initial solution can reduce considerably the computation time.

An initial solution is always provided, despite the generation of new ones at the beginning of each iteration or not (like in GRASP. See section A.2.2.4). It is expected that a good initial solution will lead to better final solutions faster, since the first iterations of the heuristic will quickly discard more solutions that are worse than the initial using partial evaluation.

The NEH heuristic is used to provide initial solutions for all the algorithms, since it is expected to provide better results than the random algorithm and the EDD heuristic. The jobs are sequentially inserted in increasing order of due dates, in order to prioritize jobs closer to their deadline. Preliminary tests showed that the first local search iterations quickly improve the solutions substantially. Therefore, although other sorting criteria could lead to even better initial solutions, they are not expected to have a great impact on the final solution or the computation time.

More elaborate constructive heuristics, such as that proposed by Hoon Lee e Pinedo (1997) and

Pfund et al. (2008), could also lead to better initial solutions. However, these heuristics often require parameters that have to be statistically estimated, or experimentally determined, to work well for a target class of instances. Since such statistical evaluation of the target instance could be rather complex, and the performance gain is not expected to be that significant (since it is expected that the computational effort is dominated by the local search), simpler procedures are preferred.

Algorithm 7: Basic Heuristic

```
1 JobList ← Sort(all jobs);                    /* Increasing order of due dates */
2 BestSolution ← ConstructiveHeuristic(JobList);   /* NEH constructive heuristic */
3 ImprovingHeuristicIterations(BestSolution);        /* GRASP or VNS iterations */
4 return BestSolution
```

Path-Relinking

Several techniques can be used to improve heuristics, such as the usage of candidate lists to guide the initial solution creation, composition and reduction of neighbourhoods to intensify the local search or focus it on specific families of neighbour solutions, diversification methods and introduction of long and short term memory to the search process.

The Path-Relinking technique is often used to intensify and introduce long term memory to the local search. It attempts to find better solutions searching intermediate solutions between two good known solutions. That is, it attempts to transform one good solution on another (the “relinking" part) using solution movements (such as those described on section A.2.2.1), and evaluates the solutions obtained after each movement (which composes the “path"). These are potentially good solutions because variations of good solutions are evaluated, trying to keep any advantageous structure shared by both extreme solutions. See Glover e Kochenberger (2002) and Resende e Ribeiro (2003) for more information on Path-Relinking.

Note that if the path considered has less than five solutions, including the extreme solutions, the solutions evaluated will certainly be direct neighbours of the extreme solution (can be obtained applying exactly one movement on the extreme solution). In this case, if the extreme solutions are local minima, then no improvement can be achieved. This was not an issue for Rocha et al. (2008)'s implementation, since the local search procedures were truncated, and the proposed solutions were not guaranteed to be local minima. For that reason, the Path-Relinking procedure could be even more successful on their implementation than in others, which used complete local searching.

Since Rocha et al. (2008) used truncated local searches, several solutions were not evaluated. To compensate for that, a Path-Relinking procedure was used mainly to intensify the local search, and was critical to the solution quality. Algorithm 8 details the process.

A.2.2.4 GRASP

The Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic, proposed by Feo e Resende (1995), is a metaheuristic for combinatorial problems. It is essentially a multi-start local search approach: every iteration start by creating a new solution which is then improved by a local search

Algorithm 8: Path-Relinking Procedure for Scheduling Problems with Parallel Machines

Input: A source solution s and a destiny solution d.

```
1  c ← s /* Create a copy of the source solution to work on.  All moves and
       swaps are performed on the current state of c.                    */
2  foreach m ∈ M do
       /* Stage 1:  Swaps                                                 */
3      foreach Position p of m, considering c do
4          Let j^c be the job at position p of m on solution c and j^d be the job at position p of m on
           solution d.;
5          if j^c ≠ j^d then
6              c ← swap(j^s, j^d);
7              if c is better than BestSolution then
8                  BestSolution ← c;
9              if c is better than the worst solution on the pool then
10                 Update the pool;
       /* Stage 2:  Insertions                                            */
11     foreach Position p of m that exists in c but not in d do
12         Let j^d be the job at position p of m on solution d.;
13         insert(j^d, p, m);
14         if c is better than BestSolution then
15             BestSolution ← c;
16         if c is better than the worst solution on the pool then
17             Update the pool;
```

procedure. See Festa e Resende (2002) and Glover e Kochenberger (2002) for more information on GRASP.

The proposed GRASP starts by sorting the job list in increasing order of due dates, and obtaining an initial solution using the NEH constructive heuristic. Then, GRASP iterations are ran until no improvement is made on the last x iterations, where x is a defined constant. On every GRASP iteration, the job list is shuffled and a new current solution is obtained using any constructive heuristic presented on section A.2.1. The current solution is then refined using any local search detailed on section A.2.2.2, and the best solution is updated if the current solution is better. Algorithms 7 and 9 depict the proposed GRASP heuristic.

A.2.2.5 VNS

Variable Neighbourhood Search (VNS) is a modern metaheuristic proposed by Hansen e Mladenovic (1999) that uses systematic changes on the neighborhood to tackle combinatorial problems. It has been successfully used to optimize over several problems. It works very much like GRASP, but instead of creating a new solution at the beginning of each iteration, the current solution is "shaken", that is, modified with random perturbations according to the current neighborhood. The current neighborhood is controlled systematically based on the previous solution's improvement.

Algorithm 9: GRASP iterations

```
   Input: BestSolution
1  Waste ← 0;
2  Iterations ← 0;
3  JobList ← all jobs;
4  CurrentSolution ← BestSolution;
5  while Waste < Waste Limit or execution time does not exceed time limit do
6      Iterations++;
7      RandomShuffle(JobList);
8      CurrentSolution ← ConstructiveHeuristic(JobList);
9      LocalSearch(CurrentSolution);                    /* Or VNS iterations */
10     if CurrentSolution < BestSolution then
11         BestSolution ← CurrentSolution;
12         Waste ← 0;
           /* Optional Path-Relinking procedure                                */
13     else Waste++;
```

de Paula et al. (2007) uses various truncated local searches, some of them fast and some more intense, to deal with the use of expensive evaluations. Because of the repeated use of fast (and thus, less intense) local searches on small instances, the solutions obtained on those tend to be poor. In order to always prioritize solution quality, and because the proposed implementation present much faster solution evaluation procedures, only the best local search developed is used.

The proposed VNS starts by setting the current neighborhood to zero (the first neighborhood defined), sorting the job list in increasing order of due dates, and obtaining an initial solution using the NEH constructive heuristic. Then, VNS iterations are ran until no improvement is made on the last x iterations, where x is a defined constant. At the beginning of each VNS iteration, a shake procedure is applied on the current solution by making an unconditional k-insertion, where k is the current neighborhood, that is, k 1-insertions are performed randomly, even it makes the solution worse. The current solution is then refined using any local search detailed on section A.2.2.2, and the best solution is updated if the current solution is better. If the best solution is not improved at the end of any iteration, the next defined neighborhood is used (k is incremented). The first neighborhood is used (k is reset to zero) otherwise. Algorithms 7 and 10 present the proposed VNS heuristic.

It is expected that, if the first neighbourhoods used are successful enough, and the perturbation is faster than creating a new solution from scratch, this heuristic will be faster than the GRASP heuristic. Moreover, this metaheuristic introduces some short term memory, since much of the structure of the current solution (which is expected to be good) is always maintained. However, the part of the solution that is kept is not chosen by any criteria at all. Therefore, although some memory is kept, there is no guarantee about its quality.

It is noteworthy that, since a k-insertion is faster than a $k+(n>0)$-insertion, the first neighborhoods yield faster shake procedures, which grow slower as the algorithm approaches the last defined neighborhood. Thus, it is interesting that the algorithm spend more time on the initial neighborhoods. Preliminary tests showed that even with $k=1$ there's a good chance that the local search will be ef-

fective, since it changes the machine's load, allowing for a quite wide range of swaps, and even other insertions, to be improving. Preliminary tests also showed that using increasingly larger neighborhoods are a better choice than repeating them, as done in Iterated Local Search (ILS) schemes. ILS only differs from the basic VNS in that the same neighborhood is always used on the shake procedure (see Glover e Kochenberger (2002) for more information on ILS).

For more information on VNS refer to Hansen e Mladenovic (1999), Hansen e Mladenovic (2002), Hansen et al. (2002) and Hansen e Mladenovic (2003).

Algorithm 10: VNS Iterations

```
   Input: BestSolution
1  Waste ← 0;
2  Iterations ← 0;
3  k ← 1;
4  while Waste < Waste Limit or execution time does not exceed time limit do
5      Iterations++;
6      for i ← 0 to k do
           /* Shake procedure                                          */
7          insertion(random job, random machine m, random position in m);
8      LocalSearch(Current Solution);
9      if Current Solution < Best Solution then
10         Best Solution ← Current Solution;
11         Waste ← 0;
12         k ← 1;
13     else
14         Waste++;
15         k++;
```

A.2.2.6 Hybrid GRASP+VNS

It is well known that using short and/or long term memory on randomized algorithms could lead to better solutions (Glover e Kochenberger (2002)).

The GRASP framework generates a new solution at the beginning of each iterations and, in its basic form, does not provide any memory at all. The VNS framework, however, only shakes the current solution at the beginning of each iteration. That is, it only changes a part of the current solution (which is expected to be reasonably good after a few iterations), leaving some part of the current solution structure intact. That provides some short term memory, although the preserved part of the current solution structure is randomly chosen.

Since the multi-start characteristic of GRASP could favour the solutions diversity, and VNS could introduce some kind of short-term memory to the algorithm, it might be a good idea to use a VNS-like heuristic as local search within a GRASP framework. Moreover, this approach has been successfully used to tackle other combinatorial problems (see Ribeiro et al. (2008), Festa et al. (2002)). Therefore, the third proposed algorithm is a hybrid GRASP+VNS heuristic. It is basically the very same GRASP

Table A.2: Experiment 1: Average solutions of GRASP using Swap ∪ Insertion as local search and each constructive heuristics: random × EDD × NEH. Computation time fixed in 60 seconds. Instances with six machines and 200 jobs.

Obj. Funct.	**random**	**edd**	**neh**
Instances with loose due dates			
SS	**0.00**	**0.00**	**0.00**
MKS	2054.74	2047.47	**2047.28**
SWT	58.36	**58.35**	**58.35**
Instances with tight due dates			
SS	**0.00**	**0.00**	**0.00**
MKS	2043.57	**2036.11**	**2036.11**
SWT	108470.86	108473.73	**108048.38**

heuristic detailed by Algorithms 7 and 9, using the VNS heuristic, detailed by Algorithm 10, as local search.

A.2.3 Experiments and Results - Part 1

All algorithms were implemented in C++, compiled with the Intel compiler (icpc) using the *-fast* flag only, and ran on a Intel Core 2 Quad 2.5GHz machine with 4GB RAM. All implementations are single-threaded.

Two experiments were conducted to determine the most effective constructive heuristic and local search, and one last experiment to compare the obtained results with other results found on the literature.

In order to determine which constructive heuristic to use, the first experiment tested each constructive heuristic considered within a GRASP algorithm. Solutions for all instances described in section B with 200 jobs were obtained using a GRASP heuristic. The local search used is the Swap ∪ Insertion Search, which was arbitrarily chosen. Each constructive heuristic described on section A.2.1 was tested. Ten different random seeds were used for each heuristic and instance. The best solution found after 60 seconds of CPU time was considered. The time limit was also arbitrarily chosen.

Analysing the average results presented on Table A.2, it is possible to see that the algorithms based on the NEH constructive heuristic obtained solutions at least as good as the others. Moreover, the solutions obtained by the algorithms based on the NEH constructive heuristic are never worse than those obtained by the other algorithms, considering the same execution time. Therefore, the NEH constructive heuristic was used to generate initial solutions in all algorithms on the next experiments.

It is worth noting that the results obtained on this experiment, considering the minimization of the TS, were optimal because they reached an obvious lower bound for this objective function (zero). See tables C.3-C.4). Since these results cannot be improved, this objective function will not be considered on the next experiments.

The second experiment seeks to determine which is the best proposed algorithm: a pure VNS implementation, or a GRASP implementation using NEH as constructive heuristic and a local search procedure. The decision to always use the NEH constructive heuristic within the GRASP implemen-

Table A.3: Experiment 2: Solutions of the VNS, GRASP, and Hybrid heuristics. Computation time fixed in 60 and 600 seconds. Using the NEH constructive heuristic and the swap ∪ Insertion local search wherever applicable. Results for the GRASP heuristic using the insertion and swap local searches are not shown because they were always worse than those presented on this table. Instances with six machines and 200 jobs.

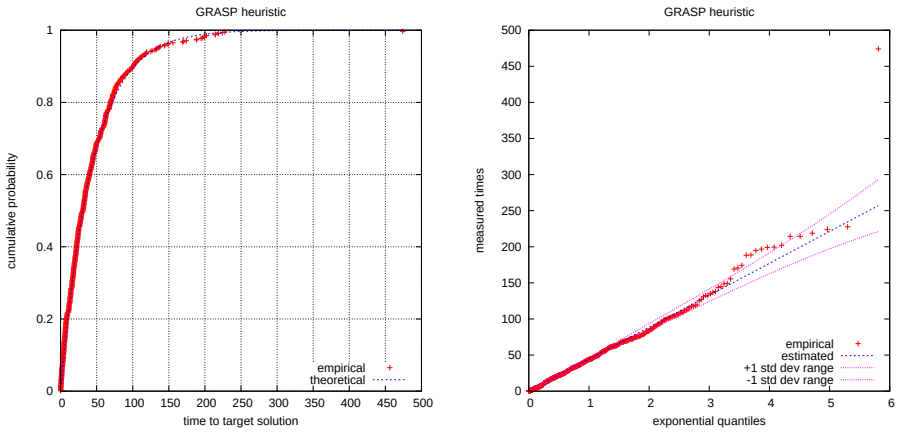
Obj. Funct.	VNS	GRASP	Hybrid
Average solutions after 60 seconds			
Instances with loose due dates			
MKS	**1983.35**	2041.45	1983.38
SWT	94.06	**92.25**	92.55
Instances with tight due dates			
MKS	1973.93	2035.27	**1972.82**
SWT	**99810.28**	103573.48	103802.50
Average solutions after 600 seconds			
Instances with loose due dates			
MKS	**1971.26**	2028.99	1973.41
SWT	92.465	**92.25**	**92.25**
Instances with tight due dates			
MKS	**1961.515**	2021.515	1963.095
SWT	99090.925	102643.395	**98920.7**

tation is based on the previous experiment's results. Solutions for all instances described on section B with 200 jobs were obtained using a GRASP heuristic. Three GRASP heuristics using each local search described on section A.2.2.2 were tested. The VNS heuristic and the hybrid heuristic detailed on section A.2.2.6 were also tested, using the Swap ∪ Insertion Search. Ten different random seeds were used for each heuristic and instance. The best solutions found after 60 and 600 seconds are considered. The time limits were also arbitrarily chosen. The VNS heuristic, when used as local search by the hybrid heuristic, arbitrarily stops after 100 iterations without improvement. The VNS maximum number of movements for the shake procedure arbitrarily equals the maximum number of iterations without improvement.

Analysing the average solutions of the tested instances, presented on Table A.3, it is possible to notice that the pure VNS heuristic tends to provide good results when a small computation time is used. However, the hybrid heuristic found better results, since the the stopping criterion was more tolerant, and the algorithm ran for more time.

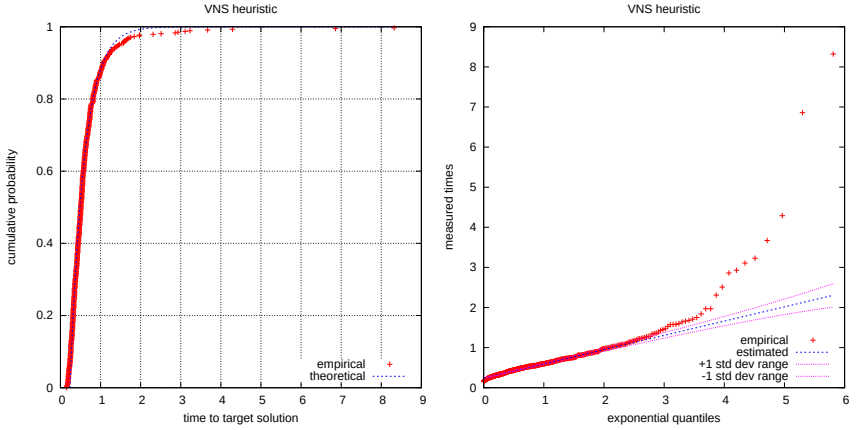
The results obtained using any of the proposed heuristics improved all those found by de Paula et al. (2007) both in quality and computation time (see tables C.7-C.12). The proposed VNS algorithm finds good quality solutions for both small and large instances, while de Paula et al. (2007)'s implementation finds solutions of poor quality for small instances.

The aim of the third experiment was to analyse the convergence of the proposed methods. Each algorithm (GRASP, VNS and Hybrid) was ran 500 times with different random seed on an arbitrary instance with six machines, 100 jobs and tight due dates. The stopping criterion considered was a target objective value, that is, the algorithm stopped when it reached a solution at least as good as the target solution. The target solution was the best solution found by the Hybrid algorithm after 1 hour



(a) Theoretical vs empirical TTTplot (b) Theoretical vs empirical QQplot

Figure A.3: GRASP heuristic

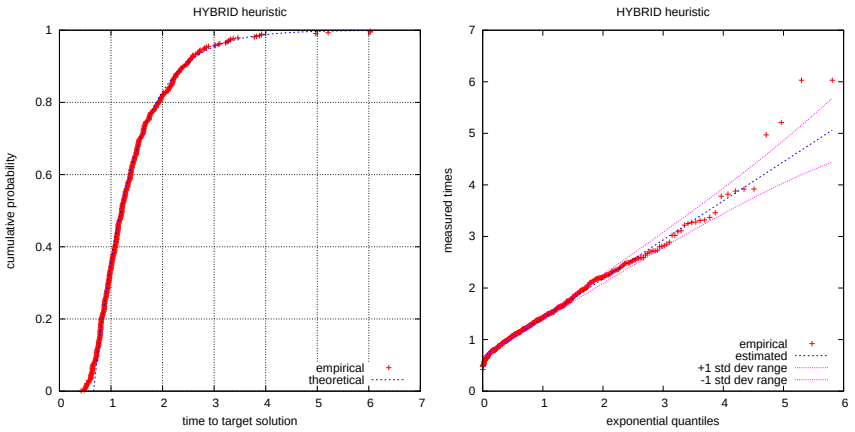


(a) Theoretical vs empirical TTTplot (b) Theoretical vs empirical QQplot

Figure A.4: VNS heuristic

of CPU time. Time-to-Target (TTT) plots are a very convenient and interesting methodology to plot measured CPU times that are assumed to fit a shifted exponential distribution. This is often the case in local search based heuristics for combinatorial optimization. TTTplots plots the expected Time-to-Target plots (TTTplots) and Quantile-Quantile plots (QQplots) were made. For more information about this methodology refer to Aiex et al. (2005).

The obtained TTTplots (figures A.3-A.5), show that the VNS heuristic has the best convergence, and the GRASP heuristic the worst. That suggests that the short term memory introduced by the VNS algorithm was advantageous, even though the part of the solution structure that is kept over the iterations is not chosen, but randomly determined.



(a) Theoretical vs empirical TTTplot (b) Theoretical vs empirical QQplot

Figure A.5: Hybrid heuristic

Finally, the fourth experiment compares the best proposed implementations' results with those obtained by Cicirello (2003), Cicirello e Smith (2005), Cicirello (2006) and Anghinolfia e Paolucci (2007), for the single machine case, using Cicirello (2003)'s instances. This set of instances comprises 10 instances, generated each with a different random seed with 60 jobs and only one machine, of each of 12 problem sets covering a spectrum from loosely to tightly constrained problem instances, totalling 120 instances. The proposed implementation's settings used are the same as those used on the previous experiment. These were the only compatible benchmark instances found on the literature for comparison. The only objective function considered here is to minimize the total weighted tardinesses on a single machine, which is a special case of the problem considered in this work.

Table A.4: Experiment 3: Average solutions of the best proposed implementations minimizing the sum of weighted tardinesses on Cicirello (2003)'s instances. Cicirello (2003), Cicirello e Smith (2005) and Cicirello (2006)'s results are not shown on this table because Anghinolfia e Paolucci (2007)'s results are at least as good as them.

		VNS		Hybrid	
Inst	Anghinolfia-2007	solution	time (s)	solution	time
1	**513**	610.70	1.77	537.30	384.99
2	5082	4910.50	1.81	**4500.30**	309.54
3	1609	1424.90	2.04	**1185.40**	334.11
4	6146	6352.60	1.77	**5952.80**	329.68
5	**4263**	4727.10	1.47	4281.10	373.45
6	**5788**	6783.70	1.90	6269.20	353.00
7	3514	3329.80	2.11	**2984.40**	329.27
8	129	112.20	2.09	**83.00**	369.92
9	6094	5838.30	1.47	**5527.70**	293.20
10	1895	1542.10	2.11	**1455.00**	330.33
11	3649	2404.70	1.38	**1654.40**	327.58

		VNS		Hybrid	
Inst	Anghinolfia-2007	solution	time (s)	solution	time
12	**0**	**0.00**	0.37	**0.00**	45.69
13	4430	3815.40	1.37	**2905.30**	245.26
14	2749	1912.70	1.55	**1394.80**	272.12
15	1245	560.90	1.68	**109.30**	245.02
16	4127	3797.60	1.82	**3210.00**	282.50
17	75	24.30	1.25	**0.00**	142.99
18	971	721.10	1.62	**211.30**	331.88
19	**0**	**0.00**	0.78	**0.00**	83.98
20	2545	1979.80	1.75	**1419.20**	258.46
21	**0**	**0.00**	0.40	**0.00**	44.77
22	**0**	**0.00**	0.41	**0.00**	45.02
23	**0**	**0.00**	0.25	**0.00**	25.83
24	1043	707.70	2.15	**200.80**	395.41
25	**0**	**0.00**	0.69	**0.00**	55.61
26	**0**	**0.00**	0.35	**0.00**	40.51
27	**0**	**0.00**	1.04	**0.00**	87.30
28	**0**	**0.00**	0.88	**0.00**	77.24
29	**0**	**0.00**	0.43	**0.00**	38.53
30	**0**	223.30	2.20	37.70	388.81
31	**0**	**0.00**	0.26	**0.00**	30.42
32	**0**	**0.00**	0.39	**0.00**	47.35
33	**0**	**0.00**	0.44	**0.00**	41.88
34	**0**	**0.00**	0.32	**0.00**	40.39
35	**0**	**0.00**	0.33	**0.00**	32.86
36	**0**	**0.00**	0.39	**0.00**	36.68
37	186	60.40	1.80	**0.00**	167.88
38	**0**	**0.00**	0.28	**0.00**	34.90
39	**0**	**0.00**	0.34	**0.00**	48.44
40	**0**	**0.00**	0.31	**0.00**	30.06
41	**69102**	70504.60	2.17	69144.70	363.19
42	57487	56070.40	1.90	**53739.90**	393.93
43	145883	145050.20	2.13	**142194.50**	456.88
44	35331	33991.00	2.46	**32374.40**	353.06
45	58935	60004.40	1.90	**58593.10**	408.61
46	34805	33526.20	2.23	**32877.20**	365.79
47	73378	72606.80	2.05	**71507.50**	373.86
48	64612	64934.20	2.42	**63503.30**	429.60
49	77771	74886.70	2.01	**73448.00**	399.48
50	31810	30833.50	2.35	**29801.00**	480.49
51	49907	45425.70	1.92	**41284.60**	444.02
52	94175	89624.60	2.00	**83594.70**	359.18
53	86891	81116.40	2.00	**74823.20**	390.29
54	118809	118397.30	1.82	**112258.10**	393.02
55	68649	65268.80	1.57	**59047.30**	246.71
56	75490	69890.70	1.99	**64991.80**	437.74
57	64575	52917.40	1.53	**50116.90**	407.87
58	45680	43645.70	1.59	**39289.90**	363.79
59	52001	52385.00	1.74	**48068.30**	406.51
60	62828	57496.30	1.98	**49822.50**	315.20
61	75916	73203.00	2.51	**72328.00**	293.19
62	44769	43129.00	3.16	**41848.00**	294.65
63	75317	77159.20	2.22	**75101.80**	327.81

		VNS		Hybrid	
Inst	Anghinolfia-2007	solution	time (s)	solution	time
64	**92572**	95491.30	2.61	94211.00	475.53
65	126696	124108.80	2.82	**122238.50**	411.09
66	**59685**	64288.50	3.68	63045.00	381.28
67	29390	29694.70	2.58	**27678.00**	388.76
68	22120	22583.20	2.48	**20860.30**	455.71
69	**64632**	67197.50	2.35	65540.90	379.45
70	75102	75535.60	2.02	**74748.50**	305.47
71	145771	147125.40	1.90	**136669.60**	492.74
72	43994	41765.30	2.56	**34401.10**	363.86
73	28785	27878.30	2.25	**25090.00**	466.08
74	30734	30367.40	1.71	**26711.00**	384.61
75	21602	17752.10	2.09	**14798.00**	401.68
76	53899	49096.70	2.31	**42575.20**	422.85
77	31937	28494.20	2.04	**24217.90**	454.07
78	19660	20842.30	1.91	**17090.20**	391.54
79	114999	117950.40	2.65	**110752.30**	379.19
80	18157	11663.50	1.62	**8376.90**	339.86
81	383485	378651.50	1.75	**375025.00**	348.20
82	409544	406211.70	2.97	**403219.30**	404.66
83	458787	460793.10	1.78	**457256.70**	381.95
84	329670	331975.50	2.18	**328566.90**	412.11
85	554766	547150.90	2.41	**543084.40**	442.18
86	361417	361896.80	2.53	**358575.40**	360.27
87	398551	396405.50	2.60	**393363.80**	366.93
88	433519	430159.80	2.21	**426887.70**	568.22
89	410092	406544.40	2.36	**401983.40**	352.85
90	401653	401846.70	2.58	**396968.00**	283.83
91	340040	327781.60	1.93	**320854.70**	324.90
92	361152	358570.80	1.76	**352897.00**	400.92
93	406728	397828.60	1.79	**389922.30**	342.28
94	332983	338274.90	1.54	**330052.00**	303.07
95	517170	522522.20	1.93	**504032.20**	378.84
96	459321	455588.80	2.09	**441689.10**	448.21
97	410889	403635.60	2.38	**393077.00**	300.44
98	522630	521896.80	1.92	**510349.50**	346.69
99	365149	357782.10	2.29	**347851.10**	402.55
100	432714	420086.90	1.59	**411594.90**	325.01
101	352990	348746.20	1.87	**345977.40**	329.78
102	**493069**	497453.40	2.78	494190.00	464.46
103	378602	378020.60	2.14	**375503.60**	422.01
104	357963	354445.60	1.72	**351875.40**	304.89
105	450806	447807.60	2.05	**443593.10**	426.11
106	455093	447627.50	2.22	**442937.80**	339.00
107	352867	354955.40	2.28	**351353.70**	266.55
108	460793	461459.80	2.19	**458301.20**	483.26
109	413004	409815.20	2.22	**407317.60**	321.35
110	**418769**	423404.40	2.37	419508.00	358.06
111	342752	333950.90	2.05	**321041.30**	382.99
112	369237	367397.20	2.08	**357332.10**	398.97
113	260176	244123.70	2.56	**238880.00**	429.06
114	464136	460009.00	1.95	**444033.10**	331.05
115	457782	442796.30	1.98	**430512.90**	334.82

		VNS		Hybrid	
Inst	Anghinolfia-2007	solution	time (s)	solution	time
116	527459	528069.30	2.48	**519377.60**	366.19
117	503199	500347.60	1.62	**488023.00**	313.96
118	350729	346494.60	1.84	**340541.90**	333.06
119	573046	567913.30	2.06	**556798.50**	378.55
120	396183	386826.50	1.41	**379847.00**	314.03

The obtained results are better than Cicirello's in all instances. As seen on table A.4, the proposed hybrid heuristic is best for 111 of the 120 instances (92.5% of Cicirello (2003)'s instances), while Anghinolfia e Paolucci (2007)'s algorithms are best for the rest. The Hybrid heuristic produced the best results. The pure VNS implementation, however, performed much faster and could also obtain better results than Anghinolfia e Paolucci (2007)'s for many instances. Note that Cicirello (2003)'s instances consider the special case of single machine only, while the proposed algorithms are designed to tackle the more general case of unrelated parallel machines. Therefore, only a subset of the proposed features are explored in this experiment. Benchmark instances for the problem with unrelated parallel machines, for a more interesting comparison, were not found on the literature, up to the time of this research.

A.2.4 Partial Conclusions

In this part, efficient implementations of three heuristics for the Scheduling Problem with Parallel Machines and Sequence-Dependent Setup Times are proposed: a GRASP based heuristic, a VNS based heuristic and a hybrid GRASP heuristic that uses a VNS heuristic as local search. A local search based on the union of two very used neighbourhoods is also proposed. The proposed algorithms are simple and require few parameters.

Analysing the results of a series of experiments, it is clear that the proposed implementations perform much faster than those of previous work for easy instances, and at least as fast, for difficult instances. The solutions found are at least as good as the best known solutions for all instances considered from previous work. The proposed algorithms could also improve the best known solutions for 92.5% of the instances found on the literature for the single machine case, using only a subset of the proposed features.

The union of the two neighbourhoods, job swaps and insertions, led to a very powerful local search, which could be implemented efficiently using partial delta evaluations, providing solutions of great quality in a short time. Therefore, the use of other techniques exploited on the literature, such as Path Relinking and truncated local searches, was no longer necessary.

Further work includes a better study of sorting criteria to obtain initial solutions and their impact on the final solution; the parallelization of the implementations, the study of interesting neighborhood reductions and other diversification/intensification methods.

It would also be interesting to experiment with other hybrid algorithms, such as a VNS algorithm that repeats some of the neighborhoods, behaving partially like ILS; or a VNS that changes neighborhoods only with a certain probability, behaving partially like Simulated Annealing. Both these approaches might help to improve the overall performance, keeping the algorithm more time on the first neighborhoods, which yeild faster shake procedures.

Finally, there's also much room for research on local searches. Larger neighborhoods, such *k*-insertions could lead to interesting local searches, for example. Using instance data, such as the weights, processing and setup times, and due dates to guide the local search could also be particularly interesting.

A.3 Measuring the Quality of the Obtained Solutions

A.3.1 Mixed-Integer Programming Formulations

Rocha et al. (2008) proposed two Mixed-Integer Programming (MIP) formulations to tackle Scheduling Problems with Parallel Machines and Sequence Dependent Setup Times: one based on Manne (1960) and another based on Wagner (1959). Ravetti (2007) proposed also a Time-Indexed formulation that is explored in this chapter. For more information on time-indexed formulations for scheduling problems, refer also to Sousa e Wolsey (1992), Queyranne e Schulz (1994), Akker et al. (2000) and Akker et al. (1999). Rocha et al. (2008) proposed a Branch-and-Bound algorithm based on constraint programming techniques, that used polynomial methods to quickly obtain lower-bounds. Ravetti (2007)'s approaches could solve instances with up to 16 jobs and 6 machines, and Rocha et al. (2008) could solve instances with up to 25 jobs and 6 machines, considering the minimization of the makespan plus the total weighted tardiness.

While the formulation based on Wagner (1959)'s work, detailed on figure A.6, uses discrete positions on machines, the formulation based on Manne (1960)'s work, detailed on figure A.7, is based on job precedence. The time-indexed formulation, detailed on figure A.8, uses discrete time positions in a similar way to the formulation based on Wagner (1959)'s work, but instead of using positions, which assume variable durations, it uses time units with a fixed minimum duration (Ravetti (2007)). The studied plant works with a 30 minute time unit on its Gantt charts.

The time-indexed formulation requires a planning horizon to be specified, that is, an upper bound on the optimal solution's makespan. Thus, the proposed implementations of the proposed algorithms based on the time indexed formulation use the makespan of a known feasible solution as planning horizon, even though there's no guarantee that the optimal solution's makespan will fit on it. It is known that this formulation cannot solve large instances of the problem which are the focus of this work. Thus, this method to estimate the planning horizon is still convenient, since the aim of the algorithms that require this parameter is to obtain lower bounds, and lower bounds obtained with a subestimated time horizon would still be valid. Moreover, since the size of the time-indexed formulation grows substantially as the planning horizon grows, it is crucial to keep it conveniently low.

In order to have a good measure of a primal solution's quality, a possible approach is to obtain a lower bound for the same problem and calculate their respective gap. Good lower bounds are also very important to perform an effective bounding in Branch-and-Bound schemes.

In this section, lower bounds for Scheduling Problems with Parallel Machines and Sequence-Dependent Setup times are obtained using classical techniques, to determine how good are the solutions obtained on chapter A.2.

Figure A.6: Formulation based on Wagner (1959)'s work for the Scheduling Problem with Parallel Machines and Sequence-Dependent Setup-Times, considering the minimization of the Total Weighted Tardinesses

$$minimize \sum_{j \in \mathscr{J}} w_j \tau_j \tag{A.5a}$$

subject to

$$\sum_{m \in \mathscr{M}} \sum_{l=0}^{|\mathscr{J}|-1} x_{jm}^l = 1 \qquad \forall j \in \mathscr{J} \tag{A.5b}$$

$$\sum_{j \in \mathscr{J}} x_{jm}^l \leq 1 \qquad \forall m \in \mathscr{M}, \forall l = 0..|\mathscr{J}|-1 \tag{A.5c}$$

$$\sum_{j \in \mathscr{J}} x_{jm}^l \leq \sum_{j \in \mathscr{J}} x_{jm}^{(l-1)} \qquad \forall m \in \mathscr{M}, \forall l = 1..|\mathscr{J}|-1 \tag{A.5d}$$

$$\tau_j \geq t_m^l + p_{jm} - d_j - M(1 - x_{jm}^l) \qquad \forall j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall l = 0..|\mathscr{J}|-1 \tag{A.5e}$$

$$t_m^0 = 0 \qquad \forall m \in \mathscr{M} \tag{A.5f}$$

$$y_m^{(l-1)} \geq s_{jj'm} - M(2 - x_{jm}^{(l-1)} - x_{j'm}^l) \qquad \forall j, j' \neq j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall l = 1..|\mathscr{J}|-1 \tag{A.5g}$$

$$t_m^l \geq t_m^{(l-1)} + \sum_{j \in \mathscr{J}} (x_{jm}^{(l-1)} p_{jm}) + y_m^{(l-1)} \qquad \forall m \in \mathscr{M}, \forall l = 1..|\mathscr{J}|-1 \tag{A.5h}$$

$$x_{jm}^l \in \{0,1\} \qquad \forall j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall l = 0..|\mathscr{J}|-1 \tag{A.5i}$$

$$y_m^l \in \{0,1\} \qquad \forall j, j' \neq j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall l = 0..|\mathscr{J}|-1 \tag{A.5j}$$

$$t_m^l \geq 0 \qquad \forall m \in \mathscr{M}, \forall l = 0..|\mathscr{J}|-1 \tag{A.5k}$$

$$\tau_j \geq 0 \qquad \forall j \in \mathscr{J} \tag{A.5l}$$

l: A specific position on a machine's schedule. There can be at most $|\mathscr{J}|$ positions, which would be the case of all jobs being scheduled on a single machine.

Decision Variables:
x_{jm}^l: 1 if job j is scheduled at position l of machine m, and 0 otherwise.
y_m^l: Setup time needed in order to process the job at position l of machine m after the job at position $l+1$ of the same machine.
t_m^l: Start time of the job scheduled at position l of machine m.

Objective Function and Restrictions:
(A.5a): The objective function is to minimize the sum of weighted tardinesses.
(A.5b): Each job must be scheduled at exactly one position and one machine.
(A.5c): There can be at most one job scheduled at each position of each machine.
(A.5d): If there's a job scheduled at position l, there must be a job scheduled at position $l-1$.
(A.5e): The jobs' completion time equals its due date plus its tardiness.
(A.5f): All jobs scheduled at the first position of any machine start at time $t = 0$.
(A.5g): If both jobs j and j' are respectively processed at positions $l-1$ and l of machine m, the setup time needed to process the job at position l after the job at position $l-1$ of machine m is greater or equal to $s_{jj'm}$.
(A.5h): The start time of the job scheduled of position l of machine m is greater than or equal to the start time of the job scheduled at the previous positions plus the setup time needed to process the job at position l after position $l-1$ of machine m plus the processing time of the job scheduled at position $l-1$ of machine m.
(A.5i) and (A.5j): The decision variables x and y are binary.
(A.5k) and (A.5l): The processing start times and tardinesses are positive.

Figure A.7: Formulation based on Manne (1960)'s work for the Scheduling Problem with Parallel Machines and Sequence-Dependent Setup-Times, considering the minimization of the Total Weighted Tardinesses.

$$minimize \sum_{j \in \mathscr{J}} w_j \tau_j \tag{A.6a}$$

subject to

$$\sum_{m \in \mathscr{M}} x_{jm} = 1 \qquad \forall j \in \mathscr{J} \tag{A.6b}$$

$$d_j + \tau_j \geq t_j + p_{jm} - M(1 - x_{jm}) \qquad \forall j \in \mathscr{J}, \forall m \in \mathscr{M} \tag{A.6c}$$

$$M(1 - x_{jm}) + M(1 - x_{j'm}) + M(1 - y_{jj'm}) + t_{j'} \geq t_j + p_{jm} + s_{jj'm} \qquad \forall j < j', (j, j') \in \mathscr{J}, \forall m \in \mathscr{M} \tag{A.6d}$$

$$M(1 - x_{jm}) + M(1 - x_{j'm}) + M y_{jj'm} + t_j \geq t_{j'} + p_{j'm} + s_{j'jm} \qquad \forall j < j', (j, j') \in \mathscr{J}, \forall m \in \mathscr{M} \tag{A.6e}$$

$$\tau_j \geq 0 \qquad \forall j \in \mathscr{J} \tag{A.6f}$$

$$t_j \geq 0 \qquad \forall j \in \mathscr{J} \tag{A.6g}$$

$$x_{jm} \in \{0, 1\} \qquad \forall j \in \mathscr{J}, \forall m \in \mathscr{M} \tag{A.6h}$$

$$y_{jj'm} \in \{0, 1\} \qquad \forall j < j', (j, j') \in \mathscr{J}, \forall m \in \mathscr{M} \tag{A.6i}$$

Decision Variables:
x_{jm}: 1 if job j is scheduled on machine m, and 0 otherwise.
$y_{jj'm}$: 1 if jobs j and j' are scheduled on machine m and j is processed immediately before j'. 0 otherwise.
t_j: Processing start time of the job j.

Objective Function and Restrictions:
(A.6a): The objective function is to minimize the sum of weighted tardinesses.
(A.6b): Each job must be processed by exactly one machine.
(A.6c): The jobs' completion time equals its due date plus its tardiness.
(A.6d): The processing of job j' can only start after job j is done and machine m set up for its processing, if both jobs are processed on machine m and job j' is processed after job j.
(A.6e): The processing of job j can only start after job j' is done and machine m set up for its processing, if both jobs are processed on machine m and job j' is not processed after job j.
(A.6f) and (A.6g): The start times and tardinesses are positive.
(A.6h) and (A.6i): The decision variables x and y are binary.

No special methods for obtaining lower bounds for the minimization of the TS because the proposed heuristics almost always reached the optimal solution, with a null objective function value, which is an obvious lower bound for this objective function.

One alternative to obtain lower bounds is to use relaxations of the problem, such as the Linear Programming relaxation of any of the formulations presented on section A.3.1.

A.3.2 Lagrangean Relaxation

The obtained Linear Programming Relaxation bounds of the formulations based on Wagner's and Manne's works are typically too weak to be of practical interest (see subsection A.3.6). Thus, a better alternative would be to use the Linear Programming Relaxation of the time-indexed formulation. However, it requires a very large amount of memory, since it requires a very large number of variables and constraints, which limits the size of the instances that can be tackled with this method. A classical approach to this problem, is to consider the Lagrangean Relaxation of the formulation, relaxing

Figure A.8: Time-Indexed Formulation for the Scheduling Problem with Parallel Machines and Sequence-Dependent Setup-Times proposed by Ravetti (2007), considering the minimization of the Total Weighted Tardinesses. The time horizon considered (T) is the makespan of a known feasible solution.

$$minimize \sum_{j \in \mathscr{J}} w_j \tau_j \tag{A.7a}$$

subject to

$$\sum_{m \in \mathscr{M}} \sum_{t=0}^{T-p_{jm}} x_{jmt} = 1 \qquad \forall j \in \mathscr{J} \tag{A.7b}$$

$$x_{jmt} + \sum_{u=t}^{\min\{t+p_{jm}+s_{j'jm}-1, T-p_{j'm}\}} x_{j'mu} \leq 1 \qquad \forall j, j' \neq j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall 0 \leq t \leq T-p_{jm} \tag{A.7c}$$

$$\tau_j \geq \sum_{m \in \mathscr{M}} \sum_{t=0}^{T-p_{jm}} (t+p_{jm}) x_{jmt} - d_j \qquad \forall j \in \mathscr{J} \tag{A.7d}$$

$$\tau_j \geq 0 \qquad \forall j \in \mathscr{J} \tag{A.7e}$$

$$x_{jmt} \in \{0,1\} \qquad \forall j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall 0 \leq t \leq T-p_{jm} \tag{A.7f}$$

Decision Variables:
x_{jmt}: 1 if job j is scheduled at position t of machine m, and 0 otherwise.

Objective Function and Restrictions:
(A.7a): The objective function is to minimize the sum of weighted tardinesses.
(A.7b): Each job must be scheduled at one time index of exactly one machine.
(A.7c): If job j is scheduled at time period t, no other job j' can be scheduled on the next $p_{jm}+s_{jj'm}$ periods, which are reserved to the processing of job j and proper machine setup.
(A.7d): The tardiness of job j is greater than or equals to the difference between its start time plus its processing time and its due date.
(A.7e): The tardinesses must be positive.
(A.7f): The decision variable x is binary.

enough constraints to reduce the problem to a more reasonable size.

Consider an Integer Linear problem

$$z = min_x\{c'x : Ax \leq b, x \in X\} \tag{A.8}$$

where $Ax \leq b$ are complicating constraints, and assume that the problem $z = min_x\{c'x : x \in X\}$ is *easy* to solve. A Lagrangean Relaxation of the former problem is obtained by relaxing the set of complicating constraints and adding them to the objective function along with associated non-negative Lagrangean Multipliers λ, to penalize the violation of such constraints. The resulting problem is a family of (Lagrangean) Relaxations of the original problem (see Wolsey (1998)), one problem for each existing vector of Lagrangean Multipliers λ.

$$z(\lambda) = min_x\{c'x + \lambda(b-Ax) : x \in X\} \tag{A.9}$$

Figure A.9: Lagrangean Relaxation of the Time-Indexed Formulation proposed by Ravetti (2007), considering the minimization of the Total Weighted Tardinesses, and relaxing the precedence constraints.

$$minimize_{\lambda} \sum_{j\in\mathscr{J}} w_j\tau_j + \sum_{j\in\mathscr{J}} \sum_{\substack{j'\in\mathscr{J}\\ j'\neq j}} \sum_{m\in\mathscr{M}} \sum_{t=0}^{T-p_{jm}} \lambda_{jj'mt}(x_{jmt} + \sum_{u=t}^{\min\{t+p_{jm}+s_{jj'm}-1,T-p_{j'm}\}} x_{j'mu} - 1) \quad \text{(A.11a)}$$

subject to

$$\sum_{m\in\mathscr{M}} \sum_{t=0}^{T-p_{jm}} x_{jmt} = 1 \qquad \forall j \in \mathscr{J} \quad \text{(A.11b)}$$

$$\tau_j \geq \sum_{m\in\mathscr{M}} \sum_{t=0}^{T-p_{jm}} (t+p_{jm})x_{jmt} - d_j \qquad \forall j \in \mathscr{J} \quad \text{(A.11c)}$$

$$\tau_j \geq 0 \qquad \forall j \in \mathscr{J} \quad \text{(A.11d)}$$

$$x_{jmt} \in \{0,1\} \qquad \forall j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall 0 \leq t \leq T - p_{jm} \quad \text{(A.11e)}$$

In order to obtain the best possible lower bound with this approach, it is necessary to solve the Lagrangean Dual Problem, which consists in determining the best lagrangean multipliers for the following problem:

$$Z_d = max_{\lambda}\{z(\lambda), \lambda \geq 0\} \quad \text{(A.10)}$$

It is possible to show that the bounds obtained by the lagrangean dual, solved to optimality, are at least as good the linear programming bounds (Wolsey (1998)). Moreover, if the relaxed problem has the integrality property, the bounds obtained are exactly equal to the linear programming bounds (Wolsey (1998)).

Consider a lagrangean relaxation $min\{c'x - \lambda(Ax-b) : Cx \leq d, x \in X\}$ of the original problem $min\{c'x : Ax \leq b, Cx \leq d, x \in X\}$, where X contains sign and integrality restrictions on $x \in X$. Such a relaxation is said to hold the integrality Property if the convex hull $Co\{x \in X : Cx \leq d\} = \{x : Cx \leq d\}$, that is, the extreme points of $\{x : Cx \leq d\}$ are in X (see Wolsey (1998), Guignard (1998) and Guignard (2003)).

For more information about Lagrangean Relaxations see Held e Karp (1970), Held e Karp (1971), Fisher (1981), Wolsey (1998), Guignard (1998) and Guignard (2003).

A.3.2.1 Relaxing the Precedence Constraints

One possible lagrangean relaxation is achieved by relaxing the precedence constraints A.7c. Figure A.9 details such Lagrangean Relaxation.

Note that, once a schedule is defined, the tardinesses can be easily calculated. Since the job, machine and time index are fixed for each cost, and the Lagrangean Multipliers are known prior its calculation, it is possible to pre-calculate the costs of each job scheduling using equation A.16 from

figure A.10.

To calculate the cost of scheduling job j at time index t in machine m, assume that $x_{jmt} = 1$. For clarity, consider the objective function of the relaxed problem A.11a-A.11e rewritten as expressions A.12-A.15:

$$minimize_{\lambda} \sum_{j \in \mathscr{J}} w_j \tau_j \qquad + \qquad \text{(A.12)}$$

$$\sum_{j \in \mathscr{J}} \sum_{\substack{j' \in \mathscr{J} \\ j' \neq j}} \sum_{m \in \mathscr{M}} \sum_{t=0}^{T-p_{jm}} \lambda_{jj'mt} x_{jmt} \qquad + \qquad \text{(A.13)}$$

$$\sum_{j \in \mathscr{J}} \sum_{\substack{j' \in \mathscr{J} \\ j' \neq j}} \sum_{m \in \mathscr{M}} \sum_{t=0}^{T-p_{jm}} \lambda_{jj'mt} \sum_{u=t}^{\min\{t+p_{jm}+s_{jj'm}-1, T-p_{j'm}\}} x_{j'mu} \qquad - \qquad \text{(A.14)}$$

$$\sum_{j \in \mathscr{J}} \sum_{\substack{j' \in \mathscr{J} \\ j' \neq j}} \sum_{m \in \mathscr{M}} \sum_{t=0}^{T-p_{jm}} \lambda_{jj'mt} \qquad \text{(A.15)}$$

From expressions A.12-A.15, three main factors have to be considered in order to calculate the cost of scheduling a job j at a certain time index t of a machine m (setting $x_{jmt} = 1$): the resulting tardiness (expression A.12), the lagrangean multipliers associated directly with such scheduling of j (expression A.13) and the lagrangean multipliers indirectly associated with it (expression A.14). Since the λ values are input data for the relaxed problem, expression A.15 is constant.

The weighted tardinesses can be easily calculated, since it involves only fixed variables and constants: the time period t, the processing time p_{jm} of job j at machine m, and its due date d_j. These values, considering all costs associated with a scheduling that satisfies constraints A.11b, sum up to expression A.12.

Since x_{jmt} is set to 1, the lagrangean multipliers associated with jobs that the job scheduling affect are directly determined by expression A.13. These multipliers, considering all costs associated with a schedule that satisfies constraints A.11b, also sum up to A.13.

Finally, from A.14, and also because $x_{jmt} = 1$; each job $j' \neq j$ will add $\lambda_{j'jmu}$ to the objective function for each time index $u \leq t$ that would make its processing time overlap the processing time of j, even if $x_{jmu} = 0$. That is, for each $u = \{t - p_{j'm-s_{j'jm}}\}...t$. Note that time indexes $u > t$ could also make the processing time of j' overlap the processing time of j, but the associated multiplier would already be considered in the same way on the symmetrical situation (the current j = the old j' and the current j' = the old j). These multipliers, considering all costs associated with a schedule that satisfies constraints A.11b, also sum up to A.14.

Using these costs, it is possible to rewrite figure A.9 as figure A.10. Note that constraints A.11c and A.11d model $max\{0, t + p_{jm} - d_j\}$. Thus, if only positive tardinesses are considered on the β costs, these constraints may be discarded, since their purpose is only to calculate the tardinesses and consider only the positive ones on the objective function (so that jobs that finish processing early are not awarded).

Figure A.10: Pre-processed Lagrangean Relaxation of a Time-Indexed Formulation proposed by Ravetti (2007), considering the minimization of the Total Weighted Tardinesses, and relaxing the precedence constraints, using β costs.

$$\beta_{jmt} = \max\{0, t + p_{jm} - d_j\} w_j + \sum_{\substack{j' \in \mathscr{J} \\ j' \neq j}} (\lambda_{jj'mt} + \sum_{u=t-p_{j'm}-s_{j'jm}+1}^{t} \lambda_{j'jmu}) \tag{A.16}$$

$$minimize_{\beta,\lambda} \sum_{j \in \mathscr{J}} \sum_{m \in \mathscr{M}} \sum_{t=0}^{T-p_{jm}} (\beta_{jmt} x_{jmt} - \sum_{\substack{j' \in \mathscr{J} \\ j' \neq j}} \lambda_{jj'mt}) \tag{A.17a}$$

subject to

$$\sum_{m \in \mathscr{M}} \sum_{t=0}^{T-p_{jm}} x_{jmt} = 1 \qquad \forall j \in \mathscr{J} \tag{A.17b}$$

The resulting subproblem is merely a multiple choice problem, that can be solved by inspection. Since an optimal solution is achived simply by choosing the smallest β (a job assignment) value for each job, it has the integrality property (see Ross e Soland (1975), Guignard (1998), Guignard (2003)), so constraints A.7f may also be discarded. For the same reason, it follows that the lower bounds obtained by such relaxation are exactly the same as those obtained by the linear programming relaxation (see Guignard (1998), Guignard (2003) and Wolsey (1998)), and might require more computation time to be obtained, depending on the convergence of the methods used to solve the Lagrangean Dual. However, the subproblem can be easily solved in $O(|\mathscr{M}||\mathscr{J}|T)$ time by inspection (see algorithm 11), and require much less memory than a linear programming relaxation would because all data can be processed directly, and there's no need to store loads of variables and constraints, necessary to create the model, in memory. Therefore, using this approach, it should be possible to obtain valid lower bounds for larger problems than using a linear programming approach.

A.3.2.2 Relaxing the Assignment Constraints

An alternative lagrangean relaxation of the time-indexed formulation is achieved by dualizing the assignment constraints A.7b (see figure A.11). Algorithm 12 could also be used to solve the associated lagrangean dual problem, but allowing λ to be also negative, since an equality constraint is dualized.

Such relaxation could be interesting because less constraints are discarded, thus obtaining stronger bounds. However, it is not trivial to implement an efficient procedure to solve the relaxation without linear programming approaches. Thus, a much larger amount of memory is required to solve the lagrangean dual, since not only a time indexed schedule have to be kept in memory, but also even more variables and constraints, than it would be needed if the precedence constraints were dualized. Since the aim of this work is to propose methods to tackle large instances of the problem, memory would quickly become a problem as the instance size increased. Thus, the lagrangean relaxation with precedence constraints dualized is chosen.

Algorithm 11: Greedy algorithm to solve the lagrangean relaxed problem detailed on figure A.10.

Input: β and $\sum\lambda = \sum_{j\in\mathscr{J}}\sum_{j'\in\mathscr{J},j'\neq j}\sum_{m\in\mathscr{M}}\sum_{t=0}^{T-p_{jm}}\lambda_{jj'mt}$, which can be calculated at the same time as β

Output: A lower bound and a (possibly infeasible) schedule *rpSol* and a lower bound *LB*

```
1  LB ← 0;
2  foreach j ∈ 𝒥 do
3      bestβ ← ∞;
4      bestMachine ← ∞;
5      bestStartTime ← ∞;
6      foreach m ∈ ℳ do
7          foreach 0 ≤ t ≤ T − p_jm do
8              if β_jmt < bestβ then
9                  bestβ ← β_jmt;
10                 bestMachine ← m;
11                 bestStartTime ← t;
12     LB ← LB + bestβ;
13     Schedule(j, rpSol, bestMachine, bestStartTime);
14 return LB − ∑λ;
15 return rpSol
```

Figure A.11: Lagrangean Relaxation of the Time-Indexed Formulation proposed by Ravetti (2007), considering the minimization of the Total Weighted Tardinesses, and relaxing the assignment constraints.

$$minimize_{\lambda} \sum_{j\in\mathscr{J}} w_j\tau_j + \sum_{j\in\mathscr{J}}\lambda_j\Big(\sum_{m\in\mathscr{M}}\sum_{t=0}^{T-p_{jm}} x_{jmt} - 1\Big) \tag{A.18a}$$

subject to

$$x_{jmt} + \sum_{u=t}^{\min\{t+p_{jm}+s_{j'jm}-1,\,T-p_{j'm}\}} x_{j'mu} \leq 1 \qquad \forall j,j' \neq j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall 0 \leq t \leq T-p_{jm} \tag{A.18b}$$

$$\tau_j \geq \sum_{m\in\mathscr{M}}\sum_{t=0}^{T-p_{jm}} (t+p_{jm})x_{jmt} - d_j \qquad \forall j \in \mathscr{J} \tag{A.18c}$$

$$\tau_j \geq 0 \qquad \forall j \in \mathscr{J} \tag{A.18d}$$

$$x_{jmt} \in \{0,1\} \qquad \forall j \in \mathscr{J}, \forall m \in \mathscr{M}, \forall 0 \leq t \leq T-p_{jm} \tag{A.18e}$$

A very similar situation is discussed by Guignard (2003) and Ross e Soland (1975) for the Generalized Assignment Problem, in which, instead of precedence constraints, one has knapsack constraints.

A.3.3 The Subgradient Method

It is possible to show that the Lagrangean Dual is a piecewise-linear convex problem (Wolsey (1998)). A simple method to solve the Lagrangean Dual without using a linear programming system is the Subgradient Method (SM). Algorithm 12 details a modified SM for the Lagrangean Relaxation proposed on section A.3.2.1.

Let LB be a lower bound, UB be an upper bound, SG be the subgradient vector and $step$ be the step taken by the SM. At the beginning of the SM, $\pi = 2$, and at the end of each iteration it is multiplied by a positive **factor** < 1.

$$SG^k_{jj'mt} = x^k_{jmt} + \sum_{u=t}^{\min\{t+p_{jm}+s_{j'jm}-1,T-p_{j'm}\}} x^k_{j'mu} - 1 \tag{A.19}$$

$$step^k = \frac{\pi * (\rho * UB - LB)}{||SG||^2} \tag{A.20}$$

$$\lambda^{k+1} = max\{0, \lambda^k + Step * SG^k\} \tag{A.21}$$

The subgradients SG, for the Lagrangean Relaxation relaxing the Precedence Constraints, are calculated using equation A.19 (Wolsey (1998)), and the relaxed problems are solved using algorithm 11.

In the proposed implementation, the input constants are arbitrarily set with values **factor**=0.5, **Tolerance**=0.001 and **MaxWaste**=50.

A.3.4 Non-Delayed Relax-and-Cut

With the lagrangean relaxation of the precedence constraints of the time-indexed formulation, a huge number of constraints are relaxed, leading to a huge number of nonzero entries in SG of algorithm 12 too. Thus, from equations A.19 and A.20, the subgradient norm $||SG||^2$ value becomes enormous, resulting in a very small step and bad convergence.

Relax-and-Cut algorithms, as defined in Lucena (2005), are algorithms that attempt to improve Lagrangean bounds by dynamically strengthening relaxations with the introduction of valid constraints, possibly selected from families with exponentially many of them, where strengthening constraints may or may not be explicitly dualized. This definition includes a wide range of algorithms that may differ in several aspects. The way strengthening constraints are treated is particularly interesting.

As for any Lagrangean Relaxation algorithm, regular Relax-and-Cut algorithms, like those developed by Gavish (1985) and Escudero et al. (1994); start with a relaxation of a given model where a set of complicating constraints is dualized, while the remaining constraints are kept. The algorithm

Algorithm 12: Subgradient Method for the Time-Indexed formulation of Scheduling Problems with Parallel Machines and Sequence-Dependent Setup Times. Let LB be a lower bound, UB be an upper bound, SG be the subgradient vector and x^k, x and $\bar{x}$ the optimal solution of the relaxed problem solved at iteration k, the solution of the lagrangean dual, and a feasible solution respectively. **Tolerance, MaxWaste** and **factor** are constants such that $0 <$**Tolerance**≤ 1, $0 <$**factor**≤ 1 and **MaxWaste** is a positive integer. Note that it is necessary that $0 < \pi \leq 2$ (Wolsey (1998)).

$k \leftarrow 0$;
$\lambda^0 \leftarrow 0$;
$LB \leftarrow 0$;
$UB \leftarrow$ Primal Bound ; `/* Ex: GRASP+VNS, detailed on section A.2.2.6 */`
$\pi \leftarrow 2$;
$Waste \leftarrow 0$;
while $\pi <$ ***Tolerance*** **do**
  $(x^k, LB^k) \leftarrow Solve(z(\lambda^k))$;
  $LB^k \leftarrow ceil(LB^k)$; `/* All input data is integer, thus any feasible solution is also integer. That allows for integer rounding. */`
  if $LB^k > LB$ **then**
    $LB \leftarrow LB^k$;
    $x \leftarrow x^k$;
    $Waste \leftarrow 0$;
    if *x is not feasible and the lower bound improved 10%* **then**
      `/* optional step */`
      $\bar{x} \leftarrow$ Run a heuristic to obtain feasibility;
      $UB^k \leftarrow$ Objective function of $\bar{x}$;
    else
      $UB^k \leftarrow$ Objective function of x^k;
    if $UB^k < UB$ **then** $UB \leftarrow UB^k$;
    if $LB = UB$ **then return** LB; `/* Optimal Solution Found */`
  else
    $Waste \leftarrow Waste + 1$;
    if $Waste =$ ***MaxWaste*** **then**
      $\pi \leftarrow \pi *$ **factor**;
      $Waste \leftarrow 0$;
  $SG^k \leftarrow$ (Equation A.19);
  if $SG^k = 0$ **then return** LB $Step^k \leftarrow$ (Equation A.20);
  $\lambda^{k+1} \leftarrow$ (Equation A.21);
  $k \leftarrow k + 1$;
return LB;
return x

then proceeds by solving the corresponding Lagrangean Dual Problem. Valid constraints that violate the Lagrangean Dual Problem solution are then identified and either dualized or kept. Either way, a new Lagrangean Dual Problem is formulated and solved. This procedure continues until a stopping criterion is reached.

Since the described method suggests that only *some* of the available strengthening constraints are dualized, it could be interesting to reduce the number of nonzero entries on the subgradient vector used by the Subgradient Method. However, since preliminary testing showed that the convergence of the proposed subgradient method can be quite slow, solving potentially many Lagrangean Dual Problems could be rather expensive.

Alternatively, Lucena (2005) proposed simple modifications to the SM to handle a large number of relaxed constraints. The idea behind the proposed scheme, called *Non-Delayed Relax-and-Cut*, is to dualize constraints *on-the-fly*, that is, as they become violated. The main difference between the *Non-Delayed Relax-and-Cut* and regular Relax-and-Cut schemes is that the dualization of violated constraints is not delayed untill the Lagrangean Dual is solved. That is, strengthening constraints are dualized after each Lagrangean *Relaxation* Problem is solved, instead of after each Lagrangean *Dual* Problem is solved.

At any iteration k of the SM, the relaxed constraints can be classified into three sets: those violated by x^k, those that have nonzero lagrangean multipliers λ^k associated with them, and the remaining constraints. Throughout this text these constraints are referred as *Currently Violated Active Set* (CA^k), *Previously Violated Active Set* (PA^k) and *Currently Inactive Set* (CI^k). Note that a constraint may be both on CA^k and PA^k simultaneously.

If no jobs are scheduled in the time interval considered by a constraint from equation A.7c, its associated subgradient value is negative (-1). If only one job is scheduled on that interval, the associated subgradient value is null. In these cases, the respective constraints are not violated. Violated constraints have more than one job scheduled on the time interval it considers, and thus have positive (≥ 1) subgradient values. It follows that constraints in CI^k do not contribute to the lagrangean costs at the current iteration. However, they play a decisive role in determining the step size on iteration k, because their squared value is considered. Moreover, from equation A.21, if the current lagrangean multiplier associated with such constraint is null, it will remain null at the end of the iteration, since the step is a positive real number and the associated subgradient value will be null or negative. Therefore, in order to deal with a great number of relaxed constraints in CI^k, the subgradient values $SG^k_{jj'mt}$ are arbitrarily set to zero whenever $SG^k_{jj'mt} \leq 0$ and $\lambda_{jj'mt} = 0$ (see Beasley (1993) and Lucena (2005)).

The described modification allows to handle problems with a lot of constraints in CI^k. However, the studied problem also introduces a big number of constraints in $CA^k \setminus PA^k$, that is, currently violated constraints with null lagrangean multipliers associated with them. These inequalities will become effectively dualized at the end of the *kth* SM iteration. In order to deal with an exceedingly large number of these constraints, Lucena (2005) proposed that only one of those constraints should be effectively relaxed at each iteration of the SM. Since the dualization of one of the constraints in $CA^k \setminus PA^k$ does not affect directly more than one machine, the proposed implementation relaxes at most one maximal constraint per machine at each iteration of the Subgradient Method, and the other constraints have their subgradient entries arbitrarily set to zero, thus becoming, in effect, constraints

in CI^k (see Lucena (2005)).

In order to choose a maximal constraint to be dualized for a machine $m \in \mathscr{M}$, it is necessary to identify the violated constraints at the current solution. Clearly, violated constraints have positive subgradient entries. Preliminary tests showed a "very violated" and penalized constraint, that is, that starts much sooner than it should and have a large associated λ value, is usually a good choice to be penalized. Thus, following Lucena (2005)'s suggestion, for every machine m, jobs j and j' and a time index t are chosen such that $(t - t' + p_{jm} + s_{jj'm}) * \lambda_{jj'mt}$ is maximal, considering that job j starts its processing at time index t and job j' starts its processing time at time index t'. In this case, $SG_{jj'mt}$ is left unchanged, and all other subgradient entries associated with machine m are set to zero.

It is possible to choose one such maximal constraint in $O(nT^2)$, with a greedy algorithm. Although this algorithm is polynomial, it may not be efficient if a large planning horizon is considered.

A.3.5 Lagrangean Heuristic

Note that, since jobs scheduled on the first time-index of a machine cannot be anticipated, solutions of relaxed problems with precedence constraints dualized tend to distribute the jobs on the first time index of the machines and the lagrangean multipliers tend to separate pairs of jobs as much as possible (penalizing violated constraints, that is, pairs of jobs with overlapping processing time periods). Since the assignment constraints already guarantee that all jobs are scheduled exactly once, when a lower bound is strong, in this case, it should be close to feasibility, except for a few jobs that still have overlapping processing times, because of a machine or sequence misplacement.

Indeed, preliminary tests showed that, for many instances (mainly the easy ones), the obtained lower bounds proved the optimality of known feasible solutions (obtained with the methods described on part A.2) or provided a very small gap.

Since the jobs scheduled with a large associated β (see equation A.16) value have a greater impact on the objective function, a procedure to force the feasibility, without compromising the objective function value much, would be to schedule, in a greedy fashion (i.e.: using one of the constructive heuristics detailed on section A.2.1), the jobs in decreasing order of associated β values (β_{jmt} such that $x_{jmt} = 1$ in the solution of the current relaxed problem). Running a local search, such as those proposed on section A.2.2.2, might also help, since it guarantees the resulting solution to be a local minimum with respect to the associated neighborhood. The proposed implementation uses the swap $\cup$ insertion search (see algorithm 6), thus it guarantees that the obtained solution of both the swap and insertion neighborhoods.

Because the use of the NEH (see section A.2.1.3) constructive heuristic, and the swap$\cup$insertion search (see section A.2.2.2) proved to be advantageous for the tested instances (see A.2.3), they are used within a proposed lagrangean heuristic, depicted by algorithm 13.

In order to minimize the computational overhead introduced by the lagrangean heuristic, the proposed implementation of the subgradient method (algorithm 12) runs algorithm 13 every time the lower bound improves 10% since the last run of the lagrangean heuristic. Since preliminary tests showed that the β values and the relaxed problem solutions change relatively little from one iteration to the next, running the lagrangean heuristic more frequently on the SM would be too costly and would lead to the same solution too often. It is expected that the use of a local search after each run

Algorithm 13: Lagrangean Heuristic

```
1 list ← sort all jobs in decreasing order of associated β values ; /* β_jmt such that x_jmt = 1
  in the solution of the current relaxed problem */
2 solution ← NEH(list);
  /* Optional                                                                          */
3 LocalSearch(solution);          /* E.g.: Swap ∪ Insertion search (see section
  A.2.2.2) */
4 return solution
```

of the lagrangean heuristic, instead of running the lagrangean heuristic more frequently, would lead to better final solutions.

A.3.6 Experiments and Results - Part 2

The lagrangean relaxations of the time indexed formulation, relax-and-cut algorithm and lagrangrean heuristic were implemented in C++ and compiled with the Intel C++ compiler (icpc) version 10.1.017, using the *-fast* flag only. The models of formulations based on Wagner (1959)'s and Manne (1960)'s works were implemented in C using the Ilog CPLEX Callable Library. The Time-Indexed formulation was implemented in C++ using the Ilog Concert Technology 2.4. All models were compiled with the GNU GCC version 4.2.3, using the *-O3 -g0* flags; and solved by Ilog CPLEX 10.2. All experiments were performed on a Intel Core 2 Quad 2.5GHz machine with 4GB RAM. All implementations and external processes (CPLEX, when applicable) are single-threaded.

In all experiments described in this section the algorithms were initialized with a feasible solution obtained with the GRASP+VNS algorithm depicted on section A.2.2.6.

In the first experiment, the linear relaxation of each model presented on section A.3.1 was solved for sets of 20 different instances with 6 machines and 6, 8, 10, 12, 14 and 16 jobs, totaling 120 instances. Each set of instances has 10 instances with loose due dates and 10 instances with tight due dates ($\theta = 1$ and $\theta = 5$, respectively. See appendix B). Table A.5 depicts the results.

Table A.5: Average Lower bounds obtained with the Linear Programming relaxation of formulations based on Manne (1960)'s (MLP) and Wagner (1959)'s (WLP) works, and with the time-indexed formulation proposed by Ravetti (2007) (TLP), for problems minimizing the sum of weighted tardinesses on instances with loose due dates.

Size (jobs)	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
Instances with loose due dates								
6	39.9	1.87	0	0.01	0	0.01	34.45	0.29
8	51.25	16.86	0	0	0	0.02	39.3	2.78
10	58.4	56.68	0	0.01	0	0.03	43.35	9.83
12	70.55	143.63	0	0.01	0	0.06	60.2	23.95
14	42.75	386.57	0	0.01	0	0.09	36.9	62.34
16	47.45	991.95	0	0.01	0	0.17	48.4	447.52
Instances with tight due dates								
6	217.6	5.5	0	0.01	0	0.02	166.4	0.07
8	433.6	3.02	0	0.01	0	0.02	257.7	0.59
10	586.55	11.09	0	0.01	0	0.04	262.85	1.95

Size (jobs)	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
12	718.6	109.3	0	0.01	0	0.1	252.25	5.88
14	706.2	2522.82	0	0.01	0	0.11	264.25	16.54
16	677.25	3570.84	0	0.02	0	0.27	300	34.97

The average results presented on Table A.5 (see also TablesC.21 and C.22, on appendix C.2, for more detailed results) show that the linear programming relaxation of the time-indexed formulation is substantially superior to the others. In fact, the bounds obtained using the formulations based on Wagner's and Manne's works were the obvious null ones, when considering the minimization of the TWT. However, when considering the minimization of the MKS, the bounds, obtained with the formulation based on Wagner (1959)'s work are much better (see tables C.19 and C.20 on appendix C.2). That suggests that the use of the "big-M" technique compromises the lower bounds, since the formulation based on Wagner (1959)'s work uses less "M" constants, improving the bounds obtained with the formulation based on Manne (1960)'s work for the minimization of the MKS but still obtaining poor lower bounds for the TWT; and the time-indexed formulation does not use the "big-M" technique at all, improving the bounds for the minimization of the TWT. It is worth noting that, although the obtained bounds are better, the CPU time needed to solve the linear relaxation of the Time-Indexed formulation is often bigger than the CPU time needed to solve the linear relaxation of the other models, which is expected since the Time-Indexed formulation yields a much bigger model and data structures.

In the following experiment, lower bounds for the same instances were obtained solving the Lagrangean Dual of the Lagrangean Relaxation with Precedence and Assignment constraints relaxed. Tables C.23 and C.24 depict the results, and the average results are presented on table A.6.

Table A.6: Average Lower bounds for problems minimizing the sum of weighted tardinesses on instances with loose due dates. The bounds were obtained with the linear relaxation of the Time-Indexed formulation proposed by Ravetti (2007) (TILP) and the Time-Indexed Lagrangean Dual with Precedence (TILDP) dualized.

Size (jobs)	TILP		TILDP		
	LB	Time (s)	LB	SM iterations	Time (s)
Instances with loose due dates					
6	34.45	0.35	34.45	25.54	0.10
8	39.30	3.20	38.75	47.00	1.27
10	43.35	9.89	42.05	61.60	23.15
12	60.20	23.97	58.70	7.86	25.79
14	36.90	61.59	36.80	17.50	21.76
16	50.95	447.70	48.45	25.15	133.11
Instances with tight due dates					
6	166.40	0.07	164.05	150.00	0.46
8	257.70	0.59	252.95	262.80	3.93
10	262.85	1.95	257.40	329.95	11.54
12	252.25	5.88	247.45	319.60	34.10
14	264.25	16.54	259.45	317.75	78.54
16	300.00	34.99	293.30	391.60	141.01

The results presented on table C.23 and C.24 show that, as expected, the lagrangean relaxation with assignment constraints relaxed provides the best lower bounds, proving the optimality of a known feasible solution whenever possible, with very few SM iterations. Only results that could be obtained before the timeout of two hours are considered on the average. Very often the lagrangean dual of

the lagrangean relaxation, with the assignment constraints dualized, could only obtain lower bounds for instances that the optimal value is zero, and no lower bound at all for instances with a nonzero optimal solution. Therefore, the average results for this method are not shown on table A.6, since the average values would be compromised by the lack of the information of some instances.

The results presented on table A.6 show that, also as expected, the bounds obtained with the lagrangean relaxation with precedence constraints relaxed matched, or were very close to, the linear programming bounds. However, to solve the relaxed problem, one still depends on third party optimization packages (Ex: CPLEX), as there are no known specific methods to solve it. Thus, since a (still) very large number of variables and constraints have to be stored (in fact not that many less than the original problem needed), memory would quickly become a problem, specially when dealing with large instances, which are the focus on this work. If such resources and time are available, a better alternative would be to solve the problem directly using the time-indexed formulation.

In the next experiment, the Non-Delayed Relax-and-Cut algorithm is compared to the Plain (without the Non-Delayed Relax-and-Cut modifications) Subgradient Method, using sets of large instances, on a total of 80. Each set has 20 different instances with 6 machines and 10, 20, 40 and 80 jobs. Each set of instances has 10 instances with loose due dates and 10 instances with tight due dates ($\theta = 1$ and $\theta = 5$, respectively. See appendix B). Tables C.25 and C.26 depict the results, and the average values are presented on table A.7.

Table A.7: Average lower bounds for problems minimizing the total weighted tardiness on instances with loose due dates obtained with the Plain Subgradient Method (PSM) and Non-Delayed Relax-and-Cut (NDRC).

Inst	PSM		NDRC	
	LB	Time (s)	LB	Time (s)
Instances with loose due dates				
10	30	15.67	31.75	13.06
20	32.75	660.57	33.1	629.23
40	0	1442.075	0	1442.07
80	0	2892.96	0	2892.94
Instances with tight due dates				
10	257.15	7.86	267.4	7.09
20	267.75	597.59	282.35	563.72
40	312.45	4767.11	330.95	4390.85
80	0	7200	0	7200

The results presented on table A.7 suggests that, although the Non-Delayed Relax-and-Cut algorithm fixes the problem of the very small step size, convergence is still slow. In fact, preliminary tests showed that, at first, a lot of iterations of the Non-Delayed Relax-and-Cut algorithm produce very uninteresting bounds, but after some iterations (which may take a lot of time) an interesting direction is found and then convergence is much better. On a few instances, this made a positive and significant difference, but the computation times were very close for most of the instances.

The last experiment analyses the proposed lagrangean heuristic using the 80 large instances of the previous experiment. Tables C.27 and C.28 depict the results while table A.8 presents average values. The time limit is fixed in one hour. The initial solutions used on the subgradient method for the lagrangean heuristics are obtained with VNS. However, only solutions obtained with the lagrangean heuristic itself (improving the initial solution or not) are considered. Thus, all values presented on

the lagrangean heuristic's columns are relative to the best feasible solution found by the lagrangean heuristic, even if the initial solutions, obtained by VNS, is better. The gaps are calculated according to equation A.22, where UB is the solution found by the heuristic (upper bound) and LB is the lower bound obtained using the Non-Delayed Relax-and-Cut algorithm.

$$gap = \frac{UB - LB}{UB} \tag{A.22}$$

Table A.8: Average gaps for problems minimizing the total weighted tardiness on instances with loose due dates obtained with the Lagrangean heuristics with (LHLS) and without a local search step (LH), and the VNS heuristic.

	LH		LHLS		VNS	
Inst	**Gap (%)**	**Time (s)**	**Gap (%)**	**Time (s)**	**Gap (%)**	**Time (s)**
Instances with loose due dates						
10	2.40	12.45	1.87	12.37	2.26	0.02
20	0.85	323.33	0.31	323.25	0.19	0.02
40	3.56	721.68	1.39	721.68	0.15	0.02
80	17.17	1450.72	0.47	1450.68	0.30	0.06
Instances with tight due dates						
10	1.66	5.67	1.43	5.63	1.49	0.02
20	8.49	171.17	7.03	166.36	6.79	0.01
40	31.31	2584.71	21.37	2584.83	20.36	0.04
80	139.70	3600.00	89.09	3600.00	85.96	0.34

The results presented on table A.8 show that the lagrangean heuristic developed obtained good feasible solutions using the solutions obtained by the relaxed problems, some even better than those obtained by the heuristics detailed on chapter A.2. However, since substantially more computation time was needed to obtain lower bounds than to obtain good feasible solutions using the heuristics proposed on part A.2, and the quality of the obtained solutions was about the same, the heuristics proposed on chapter A.2 seemed to be more interesting in practice. The use of a local search to improve the solution obtained by the lagrangean heuristic also seemed to be advantageous, since little more computational effort was spent and the obtained solutions were significantly better. For easy instances, with loose due dates, the use of the lagrangean heuristic, initialized with a solution obtained by one of the methods from part A.2, might be justified by a significant probability that the obtained bound will prove optimality of the solution or at least provide a good primal-dual gap, if this guarantee is really critical.

Note that, in all experiments, the developed algorithms terminated very quickly on many of the easy instances, and some instances with a known solution with null objective. That is because often the first lagrangean relaxation solved by the Subgradient Method finds a lower bound with the same objective function value of the known feasible solution that the algorithm was initialized with, proving its optimality. For the other instances, convergence problems were observed, and the subgradient method could not improve the lower bound, which indicates that a good lower bound is obtained on the first few iterations of the algorithm. Therefore, for practical purposes it might be a good idea to run only a few iterations of the Subgradient Method, or to set a high tolerance as stopping criterion.

A.3.7 Partial Conclusions

In this work, several methods to obtain lower-bounds for Scheduling Problems with Parallel Machines and Sequence-Dependent Setup Times were compared, and new methods were developed to improve these bounds.

The linear programming relaxation of the proposed time-indexed formulation provides much better bounds than the linear programming relaxation of the other formulations tested, at the expense of a great memory usage.

The obtained bounds were particularly good for easy instances, with loose due dates, and not so good for more complicated ones, with tight due dates and more congested schedules. However, the obtained bounds were still significant and much better than the obvious null lower-bound.

To handle large instances, a Lagrangean Relaxation of the presented time-index formulation, as suggested by Ravetti (2007), is presented and analysed. Also, to save memory, and thus allow for bigger problems to be handled, a specific implementation, which avoid linear programming approaches, is detailed. Finally, to tackle convergence problems a Non-Delayed Relax-and-Cut algorithm is also detailed.

Since the proposed lagrangean relaxation of the problem has the integrality property, the bounds obtained by lagrangean dual were at best the same as those obtained using linear relaxations of the problems. As expected, because of the convergence of the methods used to solve the lagrangean dual, the developed algorithm is often slower than linear programming approaches. However, also as expected, the specific lagrangean relaxation algorithm required substantially less memory to run, and thus could obtain valid bounds even for some large instances, with up to 6 machines and 180 jobs. However, the computation time needed to obtain such lower bounds, was exceedingly large, with respect to the time needed to obtain a primal bound. Particularly, it was not possible to solve instances that were both large (in terms of number of jobs and machines) and difficult, because they often needed a large planning horizon to process all its jobs, which increased substantially the number of time indexes considered (and thus the “real" size of the instance, for a time-indexed formulation).

The lagrangean heuristic developed successfully obtained good feasible solutions using the solutions obtained by the relaxed problems, some even better than those obtained by the heuristics detailed on chapter A.2. However, since substantially more computation time was needed to obtain lower bounds than to obtain good feasible solutions using the heuristics proposed on part A.2, and the quality of the obtained solutions was about the same, the heuristics proposed on part A.2 seemed to be more interesting in practice. For easy instances, with loose due dates, the use of the lagrangean heuristic, initialized with a solution obtained by one of the methods from part A.2, might be justified by a significant probability that the obtained bound will prove the optimality of the solution or at least provide a good gap, if this guarantee is really critical.

Further work include the development of a local search procedure for the lagrangean heuristic that uses more information from the relaxed problem solution to guide the search, a deeper study of criteria to choose constraints to be penalized on the non-delayed relax-and-cut algorithm, research on other strong valid cuts to be used on a delayed relax-and-cut or branch-and-cut scheme, and specific methods, that do not require a linear programming solver, to solve the lagrangean relaxation with

assignment restrictions relaxed. Bound strengthening techniques, such as Relaxation Linearization Technique (RLT), could also be used on the time-indexed formulation to improve its bounds, allowing for the proposed Relax-and-Cut algorithm to obtain stronger bounds. The study of Surrogate Constraints could also lead to interesting methods.

A.4 Concluding Remarks

In this work, Scheduling Problems with Parallel Machines and Sequence-Dependent Setup Times were tackled with exact and heuristic approaches.

Instances of practical interest for these problems have around six machines and 150 jobs. Since no known exact approach is able to solve instances of this size, heuristics are an interesting alternative to tackle the problem.

Efficient implementations of three heuristics were detailed: a GRASP based heuristic, a VNS based heuristic and a hybrid GRASP heuristic that uses a VNS heuristic as local search. Several local searches and three constructive algorithms were also proposed.

The solutions by the developed heuristics were at least as good as the best known solutions for all instances tested, and very competitive in terms of computation time. The proposed algorithms also often performed faster than all previously known algorithms, obtaining good solutions for instances with up to six machines and 300 jobs in reasonable time. For the special case of scheduling on a single machine, the developed methods could obtain better solutions for up to 95% of the best known solutions found on the literature.

The use of partial delta evaluation of solution movements, led to a great performance improvement, making the local search procedures cheaper. Therefore, the neighbourhoods could be completely searched, guaranteeing local minima. Previous works had to truncate the local search due to the computational effort it required.

The union of two successful neighbourhoods also seemed to be advantageous, leading to a local search that could quickly obtain better solutions than the local searches based on only one of the considered neighbourhoods could.

Because of the use of a powerful local search and efficient solution movements, the use of intensification and reduction methods (such as Path-Relinking and truncated local searches, respectively) could be avoided, leading to a simpler algorithm design, which required few parameters.

The success of the proposed implementations was mostly due to the use of a powerful local search, and the algorithms design, that provided a good diversity of evaluated solutions. The multi-start characteristic of the GRASP structure was important to provide good initial solution diversity. In turn, the VNS' shake procedure helped to improve the quality of the solutions, introducing some kind of short term memory to the algorithm.

In order to measure the obtained solutions' quality, lower bounds were obtained using linear programming and lagrangean relaxations. To obtain such bounds, a Subgradient Method and a Non-Delayed Relax-and-Cut algorithm were developed. The presented lagrangean relaxation was also used to develop a Lagrangean Heuristic.

The linear programming relaxation of the proposed time-indexed formulation provides much bet-

ter bounds than the linear programming relaxation of the other considered formulations, at the cost of a great memory usage, which limited the size of the instances that could be tackled using this method.

The bounds obtained using the time-indexed formulation were particularly good for easy instances, with loose due dates, and not so good for the more complicated ones, with tight due dates and more congested schedules. However, they were still significant and much better than the obvious null lower-bound, which was frequently found using the linear programming relaxation of the other formulations considered.

As expected, because of the convergence of the methods used to solve the lagrangean dual, the developed algorithm was often slower than linear programming approaches. However, also as expected, the specific lagrangean relaxation algorithm required substantially less memory to run, thus could obtain valid bounds even for large instances, with up to six machines and 180 jobs.

Although the results it obtained were very competitive with respect to those obtained by the local search based heuristics, it was substantially slower and required much more memory, which limited the size of the instances it could tackle. Thus, the use of the lagrangean heuristic might be justified if a formal measure of the quality of the obtained solutions is required, and enough time and resources are available.

The objective of measuring the quality of the obtained feasible solutions was partially achieved, since that was only possible for easy or medium-sized instances, at the cost of a great computational effort, which was often significantly larger than the effort spent to obtain the solutions in first place.

A.4.1 Future Research

There is still plenty of room for future research on Scheduling Problems with Sequence-Dependent Setup Times.

Since the union of two good neighbourhoods led to a more successful local search, it might be interesting to experiment with other combinations of two or more diverse neighbourhoods. Moreover, it would also be interesting to experiment with several other neighbourhood reductions, more elaborate constructive heuristics and other methods to improve the heuristics' performance.

Since the lagrangean dual bounds for the proposed lagrangean relaxation are equal to the linear programming bounds, and the obtained bounds for difficult instances were not so good, it might be a good idea to research more valid inequalities to be dualized either on a standard (delayed) Relax-and-cut or Branch-and-Cut framework, lifting procedures and bound strengthening techniques. Specific methods, that do not rely on third party optimization packages, to solve the lagrangean relaxation of the time-indexed formulation, dualizing the assignment constraints, would also be a good topic for research.

All primal bounds obtained in this work were obtained using heuristics, and only an algorithm based on the Subgradient Method was developed to solve the lagrangean dual. The study of other methods to tackle the lagrangean dual might be interesting, as well as the development of better lagrangean heuristics.

Appendix B

Instance Generation

Random instances are used in this work. They were generated with the same algorithm as Rocha et al. (2008). Let h be the makespan (C_{max}) of a solution obtained by the EDD constructive heuristic, considering the job generation order as the job insertion order (see section A.2.1). The processing times, setup times, weights and due dates values are generated using a discrete uniform distribution on the ranges described in Table B.1.

Table B.1: Range values for the instance generation

Input data	Min	Max
Processing time	5	200
Setup time	25	50
Priority	1	3
Due date	$max_{j \in \mathscr{J}}(p_j)$	$\frac{2*h}{\theta}$

The setup times also satisfy the triangle inequality ($s_{ijm} \leq s_{ikm} + p_k + s_{kjm} \forall i \neq j \neq k \in \mathscr{J}, m \in \mathscr{M}$). This is not necessary for the heuristics developed and time-indexed formulations and relaxations. However, since this feature is required by Rocha et al. (2008)'s Branch-and-Bound algorithm and Wagner (1959) and Manne (1960) based formulations, it is implemented in order to compare all results consistently.

As a rule of thumb, the greater the θ, the harder the instance is to solve, because the due dates are tighter and the scheduling system more congested.

For a fixed number of jobs, 40 instances are generated, 20 using $\theta = 1$ and 20 using $\theta = 5$. All instance information and generator can be found at the *UFMG Scheduling Group* home page [1].

Instances with 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200 and 300 jobs were considered. The number of machines considered, for the parallel machines problem considered, is always fixed in six unrelated machines, to reflect the real case presented on section A.1. This set of instances was also tested by Ravetti (2007), Rocha et al. (2008) and de Paula et al. (2007).

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

Appendix C

Extra Results Listings

C.1 Experiments with Heuristics

Table C.1: Experiment 1: Solutions of grasp (swapUmove) and constructive heuristics: random x edd x neh (fixed time = 200 seconds). Algorithms minimizing the makespan on instances with loose due dates, six machines and 200 jobs.

Inst	random	edd	neh
00	2016.50	1999.60	**1996.30**
01	2034.40	**2018.80**	2024.60
02	2000.20	1980.60	**1980.00**
03	2063.30	2051.40	**2050.30**
04	2042.90	**2024.20**	2024.90
05	**2032.00**	2032.00	2032.00
06	**2032.00**	2032.00	2032.00
07	**2032.00**	2032.00	2032.00
08	**2032.00**	2032.00	2032.00
09	**2032.00**	2032.00	2032.00
10	**2032.00**	2032.00	2032.00
11	**2032.00**	2032.00	2032.00
12	**2032.00**	2032.00	2032.00
13	**2032.00**	2032.00	2032.00
14	**2032.00**	2032.00	2032.00
15	2209.90	2187.70	**2187.40**
16	**2182.00**	2182.00	2182.00
17	2077.50	**2058.70**	2061.80
18	2105.00	2094.80	**2086.40**
19	2043.00	**2031.70**	2031.80
avg	2054.74	**2047.47**	2047.28

Table C.2: Experiment 1: Solutions of grasp (swapUmove) and constructive heuristics: random x edd x neh (fixed time = 200 seconds). Algorithms minimizing the makespan on instances with tight due dates, six machines 200 and jobs.

Inst	random	edd	neh
00	2013.00	1997.50	**1996.80**
01	2036.10	**2020.60**	2020.80
02	1993.80	1979.30	**1978.40**
03	2061.80	2051.20	**2046.60**
04	2038.50	2027.00	**2020.20**
05	**2030.00**	2030.00	2030.00
06	**2030.00**	2030.00	2030.00
07	**2030.00**	2030.00	2030.00

Inst	random	edd	neh
08	**2030.00**	2030.00	2030.00
09	**2030.00**	2030.00	2030.00
10	**2030.00**	2030.00	2030.00
11	**2030.00**	2030.00	2030.00
12	**2030.00**	2030.00	2030.00
13	**2030.00**	2030.00	2030.00
14	**2030.00**	2030.00	2030.00
15	2210.80	**2187.70**	2188.90
16	1988.20	1984.90	**1984.60**
17	2074.70	**2057.90**	2060.70
18	2102.10	**2085.90**	2092.20
19	2052.40	**2030.10**	2032.90
avg	2043.57	**2036.11**	2036.11

Table C.3: Experiment 1: Solutions of grasp (swapUmove) and constructive heuristics: random x edd x neh (fixed time = 200 seconds). Algorithms minimizing the total slackness on instances with loose due dates, six machines and 200 jobs.

Inst	random	edd	neh
00	0.00	0.00	0.00
01	0.00	0.00	0.00
02	0.00	0.00	0.00
03	0.00	0.00	0.00
04	0.00	0.00	0.00
05	0.00	0.00	0.00
06	0.00	0.00	0.00
07	0.00	0.00	0.00
08	0.00	0.00	0.00
09	0.00	0.00	0.00
10	0.00	0.00	0.00
11	0.00	0.00	0.00
12	0.00	0.00	0.00
13	0.00	0.00	0.00
14	0.00	0.00	0.00
15	0.00	0.00	0.00
16	0.00	0.00	0.00
17	0.00	0.00	0.00
18	0.00	0.00	0.00
19	0.00	0.00	0.00
avg	0.00	0.00	0.00

Table C.4: Experiment 1: Solutions of grasp (swapUmove) and constructive heuristics: random x edd x neh (fixed time = 200 seconds). Algorithms minimizing the total slackness on instances with tight due dates, six machines and 200 jobs.

Inst	random	edd	neh
00	0.00	0.00	0.00
01	0.00	0.00	0.00
02	0.00	0.00	0.00
03	0.00	0.00	0.00
04	0.00	0.00	0.00
05	0.00	0.00	0.00
06	0.00	0.00	0.00
07	0.00	0.00	0.00
08	0.00	0.00	0.00
09	0.00	0.00	0.00
10	0.00	0.00	0.00
11	0.00	0.00	0.00
12	0.00	0.00	0.00
13	0.00	0.00	0.00
14	0.00	0.00	0.00
15	0.00	0.00	0.00

Inst	random	edd	neh
16	0.00	0.00	0.00
17	0.00	0.00	0.00
18	0.00	0.00	0.00
19	0.00	0.00	0.00
avg	0.00	0.00	0.00

Table C.5: Experiment 1: Solutions of grasp (swapUmove) and constructive heuristics: random x edd x neh (fixed time = 200 seconds). Algorithms minimizing the total weighted tardiness on instances with loose due dates, six machines 200 and jobs.

Inst	random	edd	neh
00	**10.00**	10.00	10.00
01	**42.00**	42.00	42.00
02	**124.00**	124.00	124.00
03	**51.00**	51.00	51.00
04	**24.00**	24.00	24.00
05	**24.00**	24.00	24.00
06	**24.00**	24.00	24.00
07	**24.00**	24.00	24.00
08	**24.00**	24.00	24.00
09	**24.00**	24.00	24.00
10	**24.00**	24.00	24.00
11	**24.00**	24.00	24.00
12	**24.00**	24.00	24.00
13	**24.00**	24.00	24.00
14	**24.00**	24.00	24.00
15	268.20	**268.00**	268.00
16	**268.00**	268.00	268.00
17	**57.00**	57.00	57.00
18	**75.00**	75.00	75.00
19	**8.00**	8.00	8.00
avg	58.36	**58.35**	58.35

Table C.6: Experiment 1: Solutions of grasp (swapUmove) and constructive heuristics: random x edd x neh (fixed time = 200 seconds). Algorithms minimizing the total weighted tardinesses on instances with tight due dates, six machines and 200 jobs.

Inst	random	edd	neh
00	96565.90	96555.00	**96036.90**
01	88557.90	88699.00	**87573.70**
02	96870.90	96624.70	**95554.10**
03	108046.50	107799.30	**107255.40**
04	112450.00	112663.50	**111942.20**
05	**111663.00**	111663.00	111663.00
06	**111663.00**	111663.00	111663.00
07	**111663.00**	111663.00	111663.00
08	**111663.00**	111663.00	111663.00
09	**111663.00**	111663.00	111663.00
10	**111663.00**	111663.00	111663.00
11	**111663.00**	111663.00	111663.00
12	**111663.00**	111663.00	111663.00
13	**111663.00**	111663.00	111663.00
14	**111663.00**	111663.00	111663.00
15	115464.40	115327.30	**114747.80**
16	109008.90	109054.20	**107769.20**
17	103214.50	103096.80	**102515.70**
18	109697.20	109698.40	**108816.90**
19	112911.10	113326.40	**112125.60**
avg	108470.86	**108473.73**	108048.38

Table C.7: Experiment 2: Solutions of GRASP+swapUmove vs of GRASP+swap vs of GRASP+move vs VNS vs GRASP+VNS (Fixed time = 200 seconds. Fixed constructive heuris-

tic: neh). Algorithms minimizing the makespan on instances with loose due dates, six machines and 200 jobs.

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
00	1939.60	2037.90	2038.00	1996.30	**1936.00**
01	**1961.20**	2068.10	2133.00	2024.60	1961.70
02	**1919.10**	2027.30	2131.00	1980.00	1919.20
03	1983.40	2100.50	2159.00	2050.30	**1983.10**
04	**1960.20**	2071.60	2236.00	2024.60	1963.10
05	**1918.60**	2033.90	2108.00	1988.70	1919.60
06	1999.50	2104.80	2151.00	2060.50	**1997.10**
07	1908.50	1997.90	2065.00	1959.20	**1904.30**
08	**1976.80**	2084.20	2088.00	2042.70	1977.10
09	**1948.30**	2053.40	2058.00	2003.30	1952.70
10	**1909.90**	2017.80	2057.00	1967.10	1911.80
11	**2003.40**	2128.00	2201.00	2082.80	2011.70
12	1965.70	2076.10	2178.00	2026.20	**1961.60**
13	1958.10	2060.20	2169.00	2015.80	**1955.40**
14	2062.10	2168.00	2243.00	2113.50	**2058.70**
15	2131.50	2214.70	2215.00	2187.40	**2126.70**
16	**2126.00**	2126.00	2126.00	2126.00	2126.00
17	2002.30	2105.90	2143.00	2061.80	**1997.20**
18	**2028.80**	2130.70	2180.00	2086.40	2031.00
19	**1964.00**	2054.00	2054.00	2031.80	1973.60
avg	**1983.35**	2083.05	2136.65	2041.45	1983.38

Table C.8: Experiment 2: Solutions of GRASP+swapUmove vs of GRASP+swap vs of GRASP+move vs VNS vs GRASP+VNS (Fixed time = 200 seconds. Fixed constructive heuristic: neh). Algorithms minimizing the makespan on instances with tight due dates, six machines and 200 jobs.

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
00	1936.10	2045.40	2063.00	1996.80	**1934.20**
01	**1960.60**	2067.30	2112.00	2020.80	1962.80
02	1923.40	2005.00	2005.00	1978.40	**1919.80**
03	**1985.20**	2096.90	2154.00	2046.60	1987.20
04	1967.70	2068.60	2103.00	2020.20	**1961.00**
05	1916.90	2032.60	2107.00	1987.40	**1915.60**
06	1997.00	2103.70	2185.00	2060.90	**1995.60**
07	1906.20	1998.40	2042.00	1957.80	**1904.40**
08	1982.20	2083.90	2219.00	2043.60	**1980.50**
09	**1948.40**	2054.30	2068.00	2009.50	1948.50
10	**1911.30**	2015.60	2086.00	1971.00	1912.20
11	**2006.20**	2120.80	2122.00	2080.40	2007.00
12	**1961.70**	2077.40	2112.00	2028.50	1964.00
13	1960.50	2068.20	2080.00	2021.10	**1958.60**
14	2061.30	2151.00	2151.00	2120.00	**2060.00**
15	2129.30	2230.50	2273.00	2190.50	**2127.00**
16	1926.80	2038.80	2122.00	1986.20	**1921.70**
17	**1997.30**	2096.20	2156.00	2060.70	1999.60
18	**2026.60**	2137.10	2154.00	2092.20	2026.70
19	1973.90	2072.80	2153.00	2032.90	**1970.00**
avg	1973.93	2078.23	2123.35	**2035.27**	1972.82

Table C.9: Experiment 2: Solutions of GRASP+swapUmove vs of GRASP+swap vs of GRASP+move vs VNS vs GRASP+VNS (Fixed time = 200 seconds. Fixed constructive heuristic: neh). Algorithms minimizing the total slackness on instances with loose due dates, six machines and 200 jobs.

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
00	0.00	0.00	0.00	0.00	0.00

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
01	0.00	0.00	0.00	0.00	0.00
02	0.00	0.00	0.00	0.00	0.00
03	0.00	0.00	0.00	0.00	0.00
04	0.00	0.00	0.00	0.00	0.00
05	0.00	0.00	0.00	0.00	0.00
06	0.00	0.00	0.00	0.00	0.00
07	0.00	0.00	0.00	0.00	0.00
08	0.00	0.00	0.00	0.00	0.00
09	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00
avg	0.00	0.00	0.00	0.00	0.00

Table C.10: Experiment 2: Solutions of GRASP+swapUmove vs of GRASP+swap vs of GRASP+move vs VNS vs GRASP+VNS (Fixed time = 200 seconds. Fixed constructive heuristic: neh). Algorithms minimizing the total slackness on instances with tight due dates, six machines and 200 jobs.

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
00	0.00	0.00	0.00	0.00	0.00
01	0.00	0.00	0.00	0.00	0.00
02	0.00	0.00	0.00	0.00	0.00
03	0.00	0.00	0.00	0.00	0.00
04	0.00	0.00	0.00	0.00	0.00
05	0.00	0.00	0.00	0.00	0.00
06	0.00	0.00	0.00	0.00	0.00
07	0.00	0.00	0.00	0.00	0.00
08	0.00	0.00	0.00	0.00	0.00
09	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00
avg	0.00	0.00	0.00	0.00	0.00

Table C.11: Experiment 2: Solutions of GRASP+swapUmove vs of GRASP+swap vs of GRASP+move vs VNS vs GRASP+VNS (Fixed time = 200 seconds. Fixed constructive heuristic: neh). Algorithms minimizing the total weighted tardinesses on instances with loose due dates, six machines and 200 jobs.

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
00	**10.00**	10.00	10.00	10.00	10.00
01	**42.00**	42.00	42.00	42.00	42.00
02	**124.00**	124.00	124.00	124.00	124.00
03	**51.00**	51.00	51.00	51.00	51.00
04	**24.00**	24.00	24.00	24.00	24.00
05	**152.00**	152.00	152.00	152.00	152.00

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
06	**75.00**	75.00	75.00	75.00	75.00
07	**124.00**	124.00	124.00	124.00	124.00
08	**27.00**	27.00	27.00	27.00	27.00
09	**43.00**	43.00	52.60	43.00	43.00
10	**12.00**	12.00	12.00	12.00	12.00
11	**40.00**	40.00	40.00	40.00	40.00
12	315.80	314.50	339.00	**314.00**	315.80
13	**57.00**	57.00	57.00	57.00	57.00
14	**74.00**	74.00	74.00	74.00	74.00
15	302.40	271.40	313.40	**268.00**	272.30
16	**268.00**	268.00	268.00	268.00	268.00
17	**57.00**	57.00	60.80	57.00	57.00
18	**75.00**	75.00	75.00	75.00	75.00
19	**8.00**	8.00	8.00	8.00	8.00
avg	94.06	92.45	96.44	**92.25**	92.55

Table C.12: Experiment 2: Solutions of GRASP+swapUmove vs of GRASP+swap vs of GRASP+move vs VNS vs GRASP+VNS (Fixed time = 200 seconds. Fixed constructive heuristic: neh). Algorithms minimizing the total weighted tardinesses on instances with tight due dates, six machines and 200 jobs.

Inst	vns	grasp+swap	grasp+move	grasp+swapUmove	grasp+vns
00	**91635.80**	96577.00	96505.20	96036.90	94764.60
01	**84359.50**	89323.00	88317.80	87573.70	88757.20
02	**91933.40**	97548.00	96289.40	95554.10	96534.90
03	**103469.80**	111459.00	107970.00	107255.40	109993.10
04	**107356.70**	114484.00	112212.10	111942.20	109880.10
05	**91537.90**	96017.00	95773.10	95776.20	95170.60
06	**106222.40**	112524.00	109700.40	109308.50	111765.90
07	**92353.20**	97322.00	96712.60	96247.50	95752.30
08	**99796.60**	105875.00	103920.30	103144.20	104047.20
09	**118123.70**	122700.00	122657.20	122439.10	121848.20
10	**91282.70**	97813.00	95870.60	95602.10	95468.40
11	**99297.90**	103655.00	103111.20	102937.60	102599.90
12	**97108.40**	101992.00	100511.90	100400.30	100490.30
13	**90751.00**	95571.00	94163.60	93995.50	93186.00
14	**103739.30**	109497.00	108233.90	107385.50	107766.10
15	**110684.20**	116490.00	115556.00	114804.60	114022.80
16	**104257.70**	111818.00	108401.80	107769.20	107134.90
17	**98720.40**	107347.00	102401.40	102515.70	104124.10
18	**105033.40**	110495.00	109057.20	108779.50	107712.40
19	**108541.60**	116731.00	112793.30	112001.80	115031.10
avg	**99810.28**	105761.90	104007.95	103573.48	103802.50

Table C.13: Experiment 3: Solutions and computation times of the best of (Experiment 2) vs de Paula et al. (2007)'s results (max grasp iterations waste = 10 , max vns iterations waste = 5 , timeout = inf). Algorithms minimizing the makespan on instances with six machines and loose due dates.

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
				20 jobs				
00	**205.9**	0.03	214.4	**0.01300**	207.5	5.889	249.35	1.3569
01	**209.4**	0.027	210.4	**0.01301**	211	5.867	246.2	1.3777
02	235.9	0.028	238.8	**0.01102**	**231.8**	5.849	273.68	1.3389
03	221.6	0.023	227.2	**0.01303**	**217.5**	5.832	270.75	1.3212
04	**220.3**	0.025	224.7	**0.01304**	224.2	5.866	271.92	1.3866
05	**235.7**	0.026	246.2	**0.01405**	236.7	5.95	297.37	1.389
06	218.2	0.03	225.4	**0.0106**	**217.6**	5.968	272.36	1.4048
07	244.4	0.028	255.9	**0.01107**	**243.2**	5.902	282.62	1.3462
08	**193.9**	0.028	197	**0.01308**	195.2	5.889	234.02	1.3629
09	**211.4**	0.026	212.9	**0.0109**	211.4	5.898	244.14	1.3888

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
10	**204.3**	0.03	214.1	**0.01410**	207.4	5.984	243.44	1.4034
11	**247.3**	0.03	256.1	**0.01311**	249.3	5.86	296.63	1.444
12	197.3	0.022	199.8	**0.01312**	**193.3**	5.903	244.23	1.3853
13	209.1	0.028	223.7	**0.01213**	**207.5**	5.853	263.85	1.4033
14	171.3	0.03	178.9	**0.01114**	**170.5**	5.912	214.35	1.3742
15	178.5	0.029	188.1	**0.0115**	**176**	5.913	216.6	1.3563
16	**218.9**	0.029	228.8	**0.01216**	221.9	5.868	270.79	1.4038
17	**202.2**	0.022	202.9	**0.01117**	202.7	5.91	224.97	1.338
18	228.1	0.029	245.2	**0.01218**	**227.8**	5.916	289.28	1.3796
19	227	0.026	232.8	**0.01219**	**222**	5.836	275.1	1.3676
avg	214.035	0.0273	221.165	**0.01228**	**213.725**	5.89325	259.0825	1.376425
				40 jobs				
00	**433**	0.172	445.7	**0.02500**	457.8	27.3	524.48	6.869
01	**420.3**	0.155	442.2	**0.02301**	446.9	27.586	503.59	7.1046
02	**386.1**	0.161	410.4	**0.02202**	417.2	27.554	479.92	6.8634
03	**440.8**	0.167	455.7	**0.02403**	463.2	27.415	494.22	6.9385
04	**467.7**	0.191	488.2	**0.02404**	492.1	27.477	565.31	6.8017
05	**421.5**	0.173	451.4	**0.02205**	449.8	27.56	513.78	6.819
06	**394.6**	0.147	410.4	**0.02106**	415.5	27.555	491.35	6.8245
07	**467.5**	0.144	485.4	**0.02307**	489.2	27.575	547.32	6.9455
08	**415.7**	0.194	433.2	**0.02508**	432.3	27.522	514.23	7.0262
09	**400**	0.139	420.6	**0.02509**	427.7	27.297	511.77	7.0187
10	**429.3**	0.186	451.9	**0.02510**	456.2	27.568	508.82	7.0178
11	**497**	0.168	524.2	**0.02611**	518.9	27.629	598.79	6.9379
12	**386.7**	0.173	404.2	**0.02712**	407.8	27.668	503.98	7.0938
13	**410.6**	0.167	432.8	**0.02613**	436	27.378	510.9	6.8728
14	**448.2**	0.161	466.4	**0.0214**	470.7	27.395	531.57	6.9795
15	**418.8**	0.14	432.5	**0.02515**	436.9	27.547	499.69	7.0147
16	**441.3**	0.171	462.2	**0.01916**	464.6	27.609	518.56	6.8169
17	**439.8**	0.147	465.1	**0.02517**	464.1	27.668	535.31	7.1338
18	**376.3**	0.165	403.2	**0.02118**	403.7	27.33	462.07	6.905
19	**437.3**	0.151	455.8	**0.0219**	454.9	27.4	524.39	7.068
avg	**426.625**	0.1636	447.075	**0.0236435**	450.275	27.50165	517.0025	6.952565
				60 jobs				
00	**606**	0.542	631.7	**0.0600**	668.4	59.669	772.94	16.5579
01	**570.5**	0.519	594.1	**0.04801**	632.6	65.536	730.96	17.2186
02	**651.5**	0.497	672	**0.05202**	706.3	66	783.83	17.094
03	**600.2**	0.481	621.9	**0.04603**	662.7	66.059	720.57	16.9819
04	**601.2**	0.569	625.2	**0.04604**	654.4	66.042	744.14	17.0942
05	**549.8**	0.481	580.8	**0.05205**	612.2	65.996	668.92	17.1436
06	**615**	0.571	642.9	**0.05706**	673.4	66.121	734.54	17.0041
07	**626.1**	0.496	655.5	**0.06107**	697.7	67.031	775.27	17.1041
08	**643.7**	0.613	675.1	**0.06308**	707.4	67.164	771.74	17.2294
09	**585.9**	0.526	613.8	**0.05809**	648.3	66.449	711.83	16.9989
10	**589.7**	0.542	609.4	**0.0410**	654.2	66.067	715.42	17.2327
11	**673**	0.539	696.5	**0.04911**	732.1	66.158	821.71	17.1968
12	**604.4**	0.612	633.4	**0.05612**	661	67.703	728.8	17.5043
13	**585**	0.537	615.9	**0.05213**	635.4	65.97	718.74	16.844
14	**646.3**	0.463	667.7	**0.04914**	706.1	66.624	775.71	17.2474
15	**618.1**	0.486	637.7	**0.0615**	675.4	66.658	737.34	17.2888
16	**604.2**	0.595	641.8	**0.04216**	665.8	66.501	780.78	17.2441
17	**612.1**	0.627	649.3	**0.05917**	675.7	66.217	745.77	17.1397
18	**555.4**	0.52	583.1	**0.0518**	609.6	66.428	686.36	17.1858
19	**683.8**	0.5	712.5	**0.04919**	751.3	66.516	829.83	17.1166
avg	**611.095**	0.5358	638.015	**0.0527385**	671.5	66.04545	747.76	17.121345
				80 jobs				
00	**767.6**	1.198	808.8	**0.09400**	878.5	135.27	940.45	36.005
01	**816.1**	1.253	859.5	**0.10201**	911.1	134.042	1010.21	36.0602
02	**829.9**	1.695	859.4	**0.09202**	920.2	133.865	1054.42	36.4215
03	**807.5**	1.084	849.5	**0.11903**	905.9	134.11	1005.69	36.128
04	**855.4**	1.063	889	**0.10304**	964.9	135.355	1076.89	36.7995
05	**762.6**	1.302	792.2	**0.11105**	865.8	133.561	926.18	36.8251

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
06	**814**	1.178	860.2	**0.08606**	914.9	133.103	1019.39	36.7483
07	**851.4**	1.002	887.4	**0.10707**	943.4	134.331	1037.94	36.5631
08	**806.5**	1.189	849.7	**0.108**	916.4	133.041	1005.84	36.0231
09	**828.6**	1.11	868.6	**0.09809**	930.1	134.767	1012.81	36.5497
10	**794.1**	1.308	832.4	**0.08710**	898.4	134.109	971.44	35.8299
11	**821.7**	1.1	869.3	**0.10211**	931	133.517	1013.7	36.8577
12	**769.6**	1.108	806.1	**0.10712**	876.5	133.202	939.15	36.0072
13	**812.2**	1.191	848.7	**0.10813**	903.6	134.832	1002.46	35.5502
14	**893.9**	1.198	926.2	**0.07114**	999.2	133.42	1058.02	36.852
15	**843.8**	1.153	880.7	**0.09615**	937.8	134.966	1041.88	36.2016
16	**800.9**	1.089	840.5	**0.11516**	897.7	134.559	990.47	36.0309
17	**810.9**	1.037	838.5	**0.08417**	903.6	134.33	989.66	36.372
18	**805**	1.117	829.8	**0.08118**	910.1	134.383	983.31	36.7133
19	**864.9**	1.244	901.6	**0.11919**	961.3	133.561	1036.73	36.4481
avg	**817.83**	1.18095	854.905	**0.099591**	918.52	134.1162	1005.832	36.34932
				100 jobs				
00	**1002**	2.047	1043.9	**0.19700**	1139.3	231.439	1255.33	64.3979
01	**993.9**	2.51	1030.4	**0.21901**	1140.6	231.642	1233.46	64.3982
02	**993.4**	1.785	1034.7	**0.21202**	1137.5	232.096	1198.75	64.1446
03	**992.4**	2.657	1041	**0.15603**	1125.9	232.177	1227.79	64.6167
04	**1052.8**	2.394	1087.7	**0.19504**	1187.7	232.417	1287.27	62.6407
05	**970.7**	1.79	1010.8	**0.205**	1105.3	231.696	1195.53	63.4506
06	**1067.5**	2.083	1108.9	**0.206**	1212.9	231.849	1304.59	62.5619
07	**1040.3**	2.438	1079.1	**0.21407**	1182.4	230.889	1280.94	64.1119
08	**986.2**	1.93	1025.8	**0.19908**	1120.2	230.026	1193.62	64.5546
09	**985.9**	2.841	1032.1	**0.17509**	1135.6	234.289	1200.46	63.4019
10	**952.5**	2.616	983	**0.12810**	1108.3	233.219	1148.73	64.2989
11	**1025.9**	2.133	1070.6	**0.17411**	1172.3	233.136	1277.13	64.7456
12	**1093.9**	2.279	1126.6	**0.23112**	1239.9	228.384	1321.49	63.5704
13	**981.7**	2.254	1015.2	**0.21513**	1123.7	230.971	1202.17	65.0471
14	**1084.3**	1.852	1121.4	**0.18714**	1228.8	230.958	1293.08	64.0098
15	**983.4**	1.929	1022.4	**0.23315**	1130.2	235.389	1198.82	65.6309
16	**1022.1**	2.347	1056.5	**0.19416**	1169.1	232.563	1221.61	63.7293
17	**1091.7**	2.341	1118.6	**0.12317**	1233.2	231.219	1321.32	64.4469
18	**1008.4**	2.15	1054.7	**0.2118**	1157.9	230.296	1260.59	64.4096
19	**961.7**	2.815	1007.8	**0.14119**	1109.7	232.409	1162.07	64.9579
avg	**1014.535**	2.25955	1053.56	**0.1908705**	1158.025	231.8532	1239.2375	64.15627
				200 jobs				
00	**1966.3**	14.558	2036	**1.00700**	2357.1	1418.084	2421.71	401.4904
01	**1991.1**	17.456	2062.1	**1.60901**	2372.1	1271.671	2443.61	385.0401
02	**1947.7**	15.62	2015.5	**1.57502**	2320.4	1413.943	2490.14	398.2593
03	**2014.8**	18.786	2087	**1.35203**	2379.1	1427.165	2515.01	403.7055
04	**1991.5**	14.416	2064.6	**1.46404**	2364.5	1420.372	2477.55	399.1032
05	**1945.4**	15.972	2028.4	**1.45505**	2328.1	1419.814	2427.71	398.5314
06	**2026.3**	14.781	2089.6	**1.70906**	2408.7	1418.365	2529.57	400.8225
07	**1933.7**	15.174	1985.8	**1.56707**	2294.5	1424.216	2403.65	401.4006
08	**2006.9**	15.456	2080.1	**1.14308**	2381.6	1407.886	2511.76	402.2906
09	**1978.3**	17.569	2040.4	**1.52309**	2335.8	1418.161	2504.58	400.2601
10	**1941.9**	15.99	2003.5	**1.4510**	2316.5	1402.445	2401.95	398.3075
11	**2042.7**	15.336	2118.7	**1.52711**	2415.7	1420.551	2549.17	398.9781
12	**2003.5**	17.046	2065.9	**1.71312**	2372.3	1417.683	2514.13	399.5363
13	**1986.1**	16.816	2056	**1.49613**	2342.7	1410.813	2441.47	399.7423
14	**2088.5**	17.149	2164	**1.39214**	2473.4	1421.332	2624.14	397.7332
15	**2156.6**	16.39	2214	**0.90915**	2531.7	1416.982	2678.97	403.5552
16	**2215**	**0.007**	2215	0.0116	2324.1	1432.781	2573.41	1577.0781
17	**2029.2**	12.612	2110	**1.23617**	2412.5	1417.781	2615.55	399.3081
18	**2061.8**	14.725	2134.7	**1.32118**	2437.9	1420.348	2517.29	400.5998
19	**2004**	14.896	2053.9	**1.00219**	2356	1414.928	2535.5	402.7578
avg	**2016.565**	15.03775	2081.26	**1.323212**	2376.235	1410.76605	2508.8435	458.425005

Table C.14: Experiment 3: Solutions and computation times of the best of (Experiment 2) vs de Paula et al. (2007)'s results (max grasp iterations waste = 10 , max vns iterations waste = 5 , timeout = inf). Algorithms minimizing the makespan on instances with six machines and tight due dates.

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
				20 jobs				
00	**205.3**	0.029	216.4	**0.01100**	207.5	5.701	249.35	1.3281
01	**209.4**	0.026	210.4	**0.01301**	211	5.653	246.2	1.3493
02	238	0.025	239.5	**0.01302**	**231.8**	5.713	273.68	1.3223
03	221.6	0.025	227.2	**0.01503**	**217.5**	5.644	270.75	1.3034
04	**220**	0.025	225.4	**0.01204**	224.2	5.623	266.62	1.3473
05	**235.7**	0.024	246.2	**0.01205**	236.7	5.716	297.37	1.3536
06	219	0.029	222.9	**0.01206**	**217.6**	5.645	272.36	1.3545
07	244.4	0.028	255.9	**0.01107**	**243.2**	5.712	282.62	1.3082
08	196.3	0.027	202.1	**0.01208**	**195.2**	5.767	234.02	1.3357
09	**211.4**	0.025	212.9	**0.00909**	211.4	5.632	244.14	1.3462
10	**205.5**	0.031	214.8	**0.01110**	207.4	5.721	243.44	1.3441
11	**247.3**	0.03	256.1	**0.01311**	249.3	5.75	296.63	1.422
12	197	0.023	203.2	**0.01112**	**193.3**	5.686	244.23	1.3516
13	**207.4**	0.025	220.5	**0.01413**	207.5	5.718	263.85	1.3588
14	171.3	0.027	178.9	**0.01114**	**170.5**	5.61	214.35	1.326
15	**176**.5	0.029	186.8	**0.01415**	176	5.652	216.6	1.3132
16	224.7	0.027	228.1	**0.01216**	**221.9**	5.618	269.79	1.3408
17	**202.2**	0.021	202.9	**0.01117**	202.7	5.709	224.97	1.3079
18	228.1	0.028	245.2	**0.01118**	**227.8**	5.723	289.28	1.3493
19	227	0.028	232.8	**0.01519**	**222**	5.614	275.1	1.3344
avg	214.405	0.0266	221.41	**0.012245**	**213.725**	5.68035	258.7675	1.339835
				40 jobs				
00	**433.9**	0.195	445.7	**0.02400**	457.8	39.69	523.68	11.395
01	**420.3**	0.154	442.2	**0.02201**	446.9	31.335	505.79	7.2025
02	**385.5**	0.205	409.4	**0.02102**	417.2	27.022	479.92	6.6972
03	**438.1**	0.165	454.8	**0.02503**	463.2	26.788	541.72	6.7788
04	**469.8**	0.18	491.2	**0.02604**	492.1	27.191	565.31	6.7081
05	**424.4**	0.156	450.7	**0.02205**	449.8	27.188	522.28	6.6278
06	**394.6**	0.146	410.4	**0.02306**	415.5	26.967	491.35	6.6687
07	**467.5**	0.144	485.4	**0.02107**	489.2	27.132	547.32	6.7982
08	**416.9**	0.168	435.1	**0.02308**	432.3	26.763	496.73	6.7003
09	**396.5**	0.179	418.1	**0.02409**	427.7	26.752	504.47	6.8242
10	**429.3**	0.19	451.9	**0.02510**	456.2	26.843	524.42	6.6803
11	**498.9**	0.199	529.2	**0.02211**	518.9	26.861	598.79	6.7261
12	**389.4**	0.158	405.6	**0.02212**	407.8	26.914	503.98	6.8284
13	**410**	0.167	429.5	**0.02413**	436	26.929	510.9	6.7909
14	**445.8**	0.19	464.9	**0.02214**	470.7	26.871	531.57	6.7801
15	**418.8**	0.139	432.5	**0.02315**	436.9	26.871	499.69	6.8781
16	**440.8**	0.186	459.6	**0.02416**	464.6	27.143	518.56	6.6313
17	**439.7**	0.142	464.2	**0.02417**	464.1	27.085	535.31	6.8805
18	**378**	0.198	399.3	**0.02518**	403.7	26.844	462.07	6.8444
19	**437.9**	0.168	452.8	**0.02319**	454.9	26.777	518.49	6.8137
avg	**426.805**	0.17145	446.625	**0.023345**	450.275	27.7983	519.1175	7.01273
				60 jobs				
00	**608.3**	0.47	638.2	**0.0500**	667.9	65.197	772.89	16.7907
01	**566.9**	0.547	594.2	**0.05901**	632.6	64.739	699.96	16.6509
02	**646.1**	0.685	678.5	**0.05502**	706.3	65.553	782.33	16.5593
03	**599.8**	0.518	627.3	**0.05703**	662.7	64.727	727.37	16.9267
04	**601.5**	0.589	622.8	**0.04704**	654.4	65.189	762.44	16.8679
05	**551.2**	0.56	574.9	**0.04405**	612.2	65.19	649.62	16.478
06	**615**	0.571	642.9	**0.05706**	673.4	65.147	734.54	16.6297
07	**623.6**	0.511	655.4	**0.05207**	697.7	65.376	765.57	16.9726
08	**643.4**	0.495	676.2	**0.05608**	707.4	64.415	807.64	16.9665
09	**584.7**	0.459	612	**0.05409**	648.3	65.164	716.73	16.6314
10	**588.4**	0.644	619.6	**0.04510**	654.2	65.123	721.32	16.7923
11	**672**	0.582	699.1	**0.0511**	732.1	64.836	809.31	16.7956
12	**607.1**	0.402	636	**0.04712**	661	65.437	741.6	17.0207
13	**586**	0.587	611.9	**0.05113**	635.4	64.577	718.74	16.4907
14	**644.8**	0.52	674.1	**0.04414**	706.1	65.224	791.71	16.7404

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
15	**615.4**	0.542	645.2	**0.04615**	675.4	65.027	743.94	16.9977
16	**601.8**	0.607	634.2	**0.05316**	665.8	64.722	740.28	16.9412
17	**616.3**	0.521	639.6	**0.06117**	675.7	65.106	741.57	16.4496
18	**557.2**	0.511	584	**0.0618**	609.6	64.988	679.56	16.6888
19	**683.6**	0.71	714.9	**0.05819**	751.3	65.142	832.33	16.6602
avg	**610.655**	0.55155	639.05	**0.0525255**	671.475	65.04395	746.9725	16.752545
				80 jobs				
00	**769.2**	1.278	810.2	**0.09600**	878.5	131.779	926.35	35.5389
01	**819.9**	1.057	857.2	**0.1101**	911.1	132.138	1026.11	35.3628
02	**829.4**	1.258	856	**0.07402**	920.2	132.111	1041.72	35.9061
03	**811.2**	1.327	848	**0.11703**	905.9	132.858	1006.19	35.2138
04	**854.4**	1.065	889.5	**0.104**	964.9	131.12	1076.89	35.928
05	**761.9**	0.98	793.6	**0.12505**	865.8	130.952	924.98	35.6002
06	**813.6**	1.225	854.1	**0.0906**	914.9	130.506	1031.19	35.4306
07	**852.7**	1.128	893.8	**0.11307**	943.4	131.085	1035.34	35.4515
08	**809.8**	1.127	850.1	**0.10608**	916.4	130.662	998.24	35.3172
09	**828.9**	0.95	869	**0.08109**	930.1	131.905	1012.81	35.8285
10	**792.6**	1.284	822.4	**0.08410**	898.4	132.861	995.14	35.5171
11	**825.3**	1.083	865.5	**0.10111**	931	131.966	998.5	35.1366
12	**767**	1.165	802.8	**0.10612**	876.5	131.275	939.15	35.3455
13	**811**	1.152	850.8	**0.0913**	903.6	131.811	975.56	34.9071
14	**894.7**	1.183	927.7	**0.06814**	999.2	131.197	1103.92	35.6907
15	**843.1**	1.239	878.7	**0.10715**	937.8	131.628	1041.88	35.8908
16	**794.8**	1.207	835.2	**0.11716**	897.7	130.67	994.97	34.81
17	**810.9**	1.038	838.5	**0.08317**	903.6	131.785	986.36	35.6775
18	**806.7**	1.151	833.5	**0.08718**	910.1	131.693	983.31	35.6933
19	**864.6**	1.025	895.9	**0.08219**	961.3	131.518	1045.23	35.6228
avg	**818.085**	1.1461	853.625	**0.097233**	918.52	131.576	1007.192	35.49345
				100 jobs				
00	**1000.6**	2.003	1016	**0.12300**	1139.3	228.864	1226.93	63.1644
01	**998.1**	1.927	1043	**0.22401**	1140.6	230.935	1220.16	62.2975
02	**995.8**	2.766	1033.7	**0.12602**	1137.5	227.988	1212.65	62.8398
03	**993.7**	2.001	1031.3	**0.18403**	1125.9	228.051	1209.09	63.2641
04	**1053**	2.44	1092	**0.20704**	1187.7	228.691	1286.87	62.7941
05	**967.8**	2.656	1008.3	**0.16205**	1105.3	229.798	1183.83	61.8098
06	**1067.5**	1.993	1109.3	**0.20406**	1212.9	228.344	1324.29	62.0804
07	**1040.5**	2.566	1082.7	**0.23507**	1182.4	226.634	1320.74	63.6924
08	**984.3**	2.29	1024.3	**0.16808**	1120.2	227.569	1200.62	62.1419
09	**985.5**	2.515	1032.2	**0.15609**	1135.6	229.154	1222.06	62.9934
10	**948.1**	2.638	1005	**0.18210**	1108.3	229.687	1183.93	64.0697
11	**1025.1**	2.308	1068.7	**0.17611**	1172.3	228.6	1264.43	62.85
12	**1090.8**	2.374	1121.5	**0.23612**	1239.9	227.831	1332.89	62.1021
13	**982.9**	2.116	1025.3	**0.21713**	1123.7	226.506	1215.57	62.8536
14	**1088.2**	2.384	1125.7	**0.17314**	1228.8	228.874	1303.08	62.8784
15	**982.1**	2.372	1023.7	**0.23315**	1130.2	232.043	1203.02	62.8333
16	**1020.3**	2.264	1062	**0.17816**	1169.1	227.509	1234.81	62.5989
17	**1090.3**	2.389	1126.7	**0.12517**	1233.2	227.493	1340.92	63.3303
18	**1004.7**	2.357	1054.7	**0.18918**	1157.9	229.03	1260.59	62.31
19	**962**	1.969	1011.9	**0.18719**	1109.7	227.855	1162.07	62.7185
avg	**1014.065**	2.3164	1054.9	**0.184345**	1158.025	228.5728	1245.4275	62.78113
				200 jobs				
00	**1967.6**	15.679	2036	**1.75600**	2357.1	1407.286	2435.31	394.6136
01	**1992.8**	15.929	2054.6	**1.67101**	2363.4	1400.956	2551.64	394.4816
02	**1949.3**	17.426	2004.5	**1.01802**	2320.4	1402.434	2465.64	393.0334
03	**2014.7**	19.199	2090.5	**1.89103**	2379.1	1394.938	2503.11	393.9398
04	**1996.8**	15.561	2054.6	**1.64604**	2364.5	1406.881	2503.15	395.2871
05	**1950.9**	15.071	2024.3	**1.59305**	2328.1	1396.708	2391.01	393.7088
06	**2023**	15.747	2095.8	**1.50306**	2408.7	1400.483	2471.07	393.9333
07	**1931.9**	13.161	1993.6	**1.36407**	2294.5	1401.825	2441.95	393.6465
08	**2010.1**	17.05	2070.9	**1.60408**	2381.6	1398.163	2560.76	393.9293
09	**1977.8**	16.022	2043	**1.37709**	2335.8	1407.853	2466.88	392.8633
10	**1939.9**	14.668	2010	**1.37310**	2316.5	1402.867	2401.95	395.6647

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
11	**2046.6**	14.543	2116	**1.11111**	2415.7	1398.074	2523.67	389.8224
12	**1998.7**	14.679	2070.8	**1.43312**	2372.3	1400.385	2479.43	394.2595
13	**1987.1**	13.371	2049.6	**1.42813**	2342.7	1411.445	2499.07	394.1065
14	**2086**	18.216	2146.8	**0.98414**	2473.4	1402.917	2573.64	392.3317
15	**2153.8**	15.96	2238.1	**1.53115**	2531.7	1400.7	2653.37	390.377
16	**1955**	18.076	2019.9	**1.70916**	2313.5	1404.397	2496.45	397.3907
17	**2024.4**	17.117	2100.4	**1.81517**	2412.5	1399.886	2544.05	395.1666
18	**2054.1**	16.923	2125.2	**1.69318**	2437.9	1403.02	2566.99	390.959
19	**2001.8**	16.968	2064.8	**1.62219**	2356	1399.14	2490.9	393.235
avg	**2003.115**	16.0683	2070.47	**1.506195**	2375.27	1402.0179	2501.002	393.63749

Table C.15: Experiment 3: Solutions and computation times of the best of (Experiment 2) vs de Paula et al. (2007)'s results (max grasp iterations waste = 10 , max vns iterations waste = 5 , timeout = inf). Algorithms minimizing the total slackness on instances with six machines and loose due dates.

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
				20 jobs				
00	**4.9**	0.052	7	**0.01700**	5.9	5.943	21.89	1.2083
01	**3.3**	0.054	7.7	**0.01601**	3.8	5.93	17.78	1.203
02	6.4	0.048	9.8	**0.01302**	**5.1**	6.016	30.61	1.1926
03	**5.1**	0.055	8.1	**0.01703**	5.1	6.002	21.01	1.2392
04	**4.6**	0.055	7.2	**0.01704**	5.1	5.968	15.41	1.2438
05	**4.5**	0.043	6.9	**0.01605**	4.5	5.983	23.05	1.2613
06	5.7	0.046	9.7	**0.01506**	**4.4**	5.863	14.24	1.2223
07	5.8	0.049	7.1	**0.01607**	**2.2**	5.924	20.02	1.2234
08	**4**	0.05	8	**0.01308**	4	5.894	12.6	1.1784
09	7.2	0.048	8.9	**0.01609**	**4.1**	5.956	11.11	1.2196
10	4.9	0.059	9.3	**0.01510**	**4.5**	5.945	18.65	1.2235
11	**5.7**	0.051	9.7	**0.01711**	5.8	5.909	23.28	1.1979
12	**3.7**	0.062	6.6	**0.01812**	4.6	5.985	17.96	1.2715
13	**4.7**	0.04	5.7	**0.01313**	4.7	5.955	17.77	1.2695
14	**4.1**	0.057	9.5	**0.01814**	4.3	5.921	13.33	1.2281
15	**4.2**	0.055	9.7	**0.01515**	5.6	5.943	19.06	1.1813
16	**1.8**	0.04	1.9	**0.01216**	6.4	5.944	19.34	1.1784
17	5.5	0.042	7.1	**0.01417**	**4.5**	5.971	18.65	1.2601
18	4.2	0.053	6.6	**0.01518**	**3.7**	5.985	21.17	1.2425
19	4.2	0.058	10.9	**0.01319**	**3.8**	5.956	16.88	1.2316
avg	4.725	0.05085	7.87	**0.015395**	**4.605**	5.94965	18.6905	1.223815
				40 jobs				
00	0	0.139	0	**0.02600**	0.1	27.66	0.91	5.596
01	0	0.139	0.3	**0.02401**	0	27.481	0.4	5.4861
02	0	0.137	0	**0.02702**	0.1	27.459	3.61	5.5319
03	0	0.154	0.1	**0.02703**	0.1	27.505	0.91	5.5495
04	0	0.151	0.2	**0.02804**	0.1	27.44	1.31	5.547
05	0	0.138	0.1	**0.02505**	0.1	27.355	2.01	5.4885
06	0	0.126	0	**0.02606**	0	27.657	0.4	5.5687
07	0	0.135	0	**0.02607**	0.3	27.721	0.83	5.6301
08	0	0.142	0.1	**0.02508**	0.4	27.81	0.54	5.706
09	0	0.139	0.1	**0.02809**	0	27.581	2.9	5.7881
10	0	0.125	0	**0.0210**	0.2	27.438	1.32	5.5108
11	0	0.137	0.2	**0.02211**	0.1	27.712	0.31	5.6962
12	0	0.132	0	**0.02412**	0.2	27.302	1.02	5.6802
13	0	0.148	0.2	**0.02613**	0.1	27.829	0.91	5.6589
14	0	0.142	0.2	**0.0314**	0.3	27.816	1.13	5.6826
15	0	0.135	0	**0.02715**	0.4	27.75	1.64	5.872
16	0	0.138	0.3	**0.02216**	0.1	27.512	1.01	5.4722
17	0	0.135	0	**0.02717**	0.3	27.976	0.63	5.5596
18	0	0.135	0.2	**0.03218**	0.3	27.51	0.73	5.439
19	0	0.138	0.1	**0.02719**	0.1	27.639	0.01	5.4729

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
avg	0	0.13825	0.105	**0.026153**	0.165	27.60765	1.1265	5.596815
				60 jobs				
00	0	0.209	0	**0.03900**	0	66.799	0.1	13.1659
01	0	0.177	0	**0.03901**	0	65.997	0.6	12.8567
02	0	0.211	0	**0.03802**	0	66.619	0	13.1489
03	0	0.181	0	**0.03503**	0	65.98	0	13.057
04	0	0.183	0	**0.03704**	0	67.354	0	13.0624
05	0	0.184	0	**0.03405**	0.1	66.299	0.41	12.7929
06	0	0.191	0	**0.03706**	0	66.888	0	13.1938
07	0	0.19	0	**0.03507**	0.1	66.524	0.01	13.0114
08	0	0.193	0	**0.03908**	0.1	66.85	0.01	13.385
09	0	0.2	0	**0.03409**	0.1	66.393	0.11	13.3453
10	0	0.194	0	**0.03910**	0.1	66.021	0.71	13.1631
11	0	0.177	0	**0.03411**	0	66.308	0	12.9018
12	0	0.197	0	**0.03712**	0.1	66.329	0.01	13.1589
13	0	0.204	0	**0.03913**	0.1	66.536	0.01	12.8396
14	0	0.19	0	**0.03914**	0	66.763	0	13.1443
15	0	0.187	0	**0.03715**	0	66.313	0.2	13.7903
16	0	0.188	0	**0.03716**	0	65.948	0	12.9488
17	0	0.2	0	**0.03717**	0	66.466	0.2	13.2746
18	0	0.18	0	**0.03618**	0	66.555	0	12.5955
19	0	0.184	0	**0.03519**	0	66.989	0.3	13.6339
avg	0	0.191	0	**0.036945**	0.035	66.49655	0.1335	13.123505
				80 jobs				
00	0	0.252	0	**0.04800**	0	134.341	0	27.1041
01	0	0.232	0	**0.04401**	0	135.078	0	27.6788
02	0	0.23	0	**0.04802**	0	134.267	0.2	26.9517
03	0	0.241	0	**0.04703**	0.2	133.41	0.02	27.334
04	0	0.246	0	**0.04904**	0	134.648	0	26.8338
05	0	0.258	0	**0.05405**	0	134.687	0	27.1827
06	0	0.232	0	**0.04906**	0	133.006	0	27.0266
07	0	0.236	0	**0.04507**	0	135.178	0	27.0078
08	0	0.243	0	**0.04708**	0	134.267	0.9	27.2277
09	0	0.245	0	**0.04609**	0	135.32	0	27.419
10	0	0.239	0	**0.04710**	0	135.779	0	27.4959
11	0	0.239	0	**0.05111**	0	133.061	0	26.4871
12	0	0.236	0	**0.04812**	0	134.426	0	26.4636
13	0	0.243	0	**0.04913**	0	134.568	0	26.8998
14	0	0.238	0	**0.0514**	0	135.124	0	27.9934
15	0	0.245	0	**0.04715**	0	135.751	0	27.7841
16	0	0.234	0	**0.0516**	0	134.937	0	27.6497
17	0	0.24	0	**0.04717**	0	134.703	0	27.6183
18	0	0.234	0	**0.04818**	0	136.907	0	27.5187
19	0	0.25	0	**0.04319**	0	132.719	0.1	27.3809
avg	0	0.24065	0	**0.04808**	0.01	134.60885	0.061	27.252885
				100 jobs				
00	0	0.311	0	**0.06100**	0.1	231.444	0.01	48.9374
01	0	0.295	0	**0.06101**	0	234.801	0	47.6301
02	0	0.318	0	**0.05902**	0	231.591	0	46.3041
03	0	0.31	0	**0.05903**	0	234.437	0	48.4997
04	0	0.314	0	**0.06404**	0	231.571	0	47.3881
05	0	0.302	0	**0.0605**	0	233.333	0	46.8953
06	0	0.306	0	**0.06206**	0	234.945	0	48.7775
07	0	0.327	0	**0.06507**	0	233.93	0	46.885
08	0	0.324	0	**0.06208**	0	234.551	0	47.3321
09	0	0.305	0	**0.06109**	0	232.569	0	48.0089
10	0	0.304	0	**0.0610**	0	234.35	0	48.088
11	0	0.308	0	**0.0611**	0	232.728	0	46.9568
12	0	0.309	0	**0.05912**	0	234.539	0.1	50.0179
13	0	0.304	0	**0.06113**	0	232.841	0	47.7711
14	0	0.314	0	**0.0614**	0	231.052	0	47.4282
15	0	0.305	0	**0.06115**	0	234.058	0	47.6488

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
16	0	0.323	0	**0.05816**	0	233.895	0	47.2865
17	0	0.311	0	**0.0617**	0	233.247	0	48.2517
18	0	0.313	0	**0.06418**	0	232.761	0	47.7011
19	0	0.335	0	**0.06319**	0	230.983	0	48.0073
avg	0	0.3119	0	**0.0613515**	0.005	233.1813	0.0055	47.79078
				200 jobs				
00	0	0.89	0	**0.17600**	0	1416.256	0	296.5776
01	0	0.908	0	**0.17401**	0	1423.58	0	303.378
02	0	0.898	0	**0.18102**	0	1416.682	0	295.5022
03	0	0.89	0	**0.17603**	0	1423.919	0	298.5469
04	0	0.912	0	**0.17904**	0	1425.8	0	299.952
05	0	0.888	0	**0.17305**	0	1417.626	0	299.2836
06	0	0.893	0	**0.17506**	0	1429.151	0	300.3631
07	0	0.917	0	**0.18107**	0	1423.961	0	297.9141
08	0	0.923	0	**0.17708**	0	1419.424	0	299.8564
09	0	0.879	0	**0.17709**	0	1427.914	0	300.4844
10	0	0.902	0	**0.17610**	0	1373.378	0	288.8488
11	0	0.894	0	**0.17811**	0	1378.656	0	288.0736
12	0	0.903	0	**0.17712**	0	1380.534	0	288.1134
13	0	0.899	0	**0.17813**	0	1381.835	0	293.9205
14	0	0.909	0	**0.17814**	0	1421.127	0	295.6737
15	0	0.883	0	**0.17515**	0	1419.497	0	300.7487
16	0	0.009	0	**0.00716**	0	1417.074	0	302.0814
17	0	0.893	0	**0.17617**	0	1565.231	0	358.6641
18	0	0.898	0	**0.17618**	0	1629.15	0	413.343
19	0	0.903	0	**0.17919**	0	1801.593	0	424.7783
avg	0	0.85455	0	**0.168545**	0	1449.6194	0	312.30519

Table C.16: Experiment 3: Solutions and computation times of the best of (Experiment 2) vs de Paula et al. (2007)'s results (max grasp iterations waste = 10 , max vns iterations waste = 5 , timeout = inf). Algorithms minimizing the total slackness on instances with six machines and tight due dates.

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
				20 jobs				
00	**5.4**	0.052	8.4	**0.01600**	5.9	5.744	21.49	1.1474
01	**3.3**	0.057	7.7	**0.01701**	3.8	5.718	17.78	1.1628
02	**4**	0.051	7.9	**0.01702**	5.1	5.75	30.61	1.15
03	**5.1**	0.054	8.1	**0.01403**	5.1	5.721	21.01	1.1981
04	**4.8**	0.049	7.9	**0.01704**	5.1	5.723	15.41	1.2153
05	**4.5**	0.044	6.9	**0.01805**	4.5	5.74	23.05	1.227
06	4.7	0.059	7.5	**0.01706**	**4.4**	5.682	14.24	1.1832
07	5.8	0.048	7.1	**0.01607**	**2.2**	5.778	20.02	1.1868
08	**4**.5	0.054	5.6	**0.01708**	4	5.713	12.6	1.1363
09	7.2	0.045	8.9	**0.01509**	**4.1**	5.691	11.11	1.1731
10	5.4	0.053	9.5	**0.01710**	**4.5**	5.731	18.65	1.1901
11	**5.7**	0.052	9.7	**0.01411**	5.8	5.862	23.28	1.1822
12	**4**	0.054	9.5	**0.0212**	4.6	5.632	17.96	1.2092
13	**3.9**	0.048	8.1	**0.01413**	4.7	5.722	14.07	1.1572
14	**4.1**	0.058	9.5	**0.01614**	4.3	5.671	13.33	1.1921
15	**4.4**	0.055	8.7	**0.01715**	5.6	5.631	19.06	1.1351
16	**4.8**	0.056	7.1	**0.01616**	6.4	5.634	19.34	1.1344
17	5.5	0.044	7.1	**0.01117**	**4.5**	5.753	18.65	1.2253
18	4.2	0.056	6.6	**0.01918**	**3.7**	5.714	21.17	1.2014
19	4.2	0.055	10.9	**0.01519**	**3.8**	5.765	16.88	1.2025
avg	4.775	0.0522	8.135	**0.016299**	**4.605**	5.71875	18.4855	1.180475
				40 jobs				
00	0	0.138	0	**0.02800**	0.1	30.258	0.91	7.8338
01	0	0.139	0.3	**0.02801**	0	42.884	1.6	7.0474
02	0	0.138	0.2	**0.02602**	0.1	27.591	3.61	5.5331
03	0	0.14	0.3	**0.02503**	0.1	27.216	1.61	5.4506

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
04	0	0.14	0.1	**0.02404**	0.1	27.324	1.31	5.4954
05	0	0.152	0.4	**0.02605**	0.1	27.028	0.61	5.2988
06	0	0.126	0	**0.02906**	0	27.149	0.4	5.4689
07	0	0.137	0	**0.02607**	0.3	27.429	0.83	5.5929
08	0	0.149	0.1	**0.02808**	0.4	27.416	0.54	5.7026
09	0	0.135	0.2	**0.02709**	0	27.364	1.2	5.7114
10	0	0.12	0	**0.02110**	0.2	27.405	1.32	5.4925
11	0	0.138	0.2	**0.02711**	0.1	27.31	0.31	5.639
12	0	0.123	0	**0.0212**	0.2	27.404	1.82	5.5854
13	0	0.145	0.2	**0.02413**	0.1	27.524	2.01	5.7124
14	0	0.149	0.3	**0.02514**	0.3	27.837	1.13	5.6317
15	0	0.135	0	**0.02915**	0.4	27.305	1.64	5.7935
16	0	0.135	0.1	**0.02616**	0.1	27.25	1.01	5.421
17	0	0.132	0.2	**0.02717**	0.3	27.305	0.63	5.5285
18	0	0.136	0.2	**0.02818**	0.3	27.302	0.43	5.4372
19	0	0.14	0.2	**0.02719**	0.1	27.261	1.21	5.5731
avg	0	0.13735	0.15	**0.026199**	0.165	28.2781	1.2065	5.74746
				60 jobs				
00	0	0.189	0	**0.03600**	0	65.742	0.6	13.2082
01	0	0.177	0	**0.03401**	0	65.186	0	12.7596
02	0	0.188	0	**0.03702**	0	65.745	0.2	12.8755
03	0	0.179	0	**0.03703**	0	65.258	0.9	13.6228
04	0	0.197	0	**0.03804**	0	65.64	0	12.808
05	0	0.184	0	**0.03705**	0.1	65.964	0.21	13.1284
06	0	0.193	0	**0.03406**	0	66.058	0	13.1268
07	0	0.183	0	**0.03507**	0.1	66.099	0.01	13.2079
08	0	0.19	0	**0.03708**	0.1	66.425	0.01	13.1985
09	0	0.189	0	**0.03409**	0.1	65.825	0.21	13.3865
10	0	0.189	0	**0.03710**	0.1	65.333	0.21	13.2833
11	0	0.182	0	**0.03411**	0	66.37	0	13.024
12	0	0.184	0	**0.03512**	0.1	65.777	1.21	12.9957
13	0	0.201	0	**0.04113**	0.1	66.76	0.01	12.86
14	0	0.198	0	**0.03714**	0	65.334	0	13.0774
15	0	0.178	0	**0.03715**	0	65.715	0	13.4615
16	0	0.181	0	**0.03816**	0	65.769	0.2	12.8229
17	0	0.183	0	**0.03717**	0	65.771	0.2	13.0931
18	0	0.185	0	**0.03618**	0	65.844	0	12.8084
19	0	0.188	0	**0.03419**	0	65.816	0	13.2856
avg	0	0.1869	0	**0.036345**	0.035	65.82155	0.1985	13.101705
				80 jobs				
00	0	0.272	0	**0.0500**	0	137.012	0	26.9132
01	0	0.231	0	**0.04801**	0	133.501	0	27.2681
02	0	0.23	0	**0.04702**	0	133.066	0.4	26.6716
03	0	0.245	0	**0.04803**	0.2	132.244	0.02	27.5804
04	0	0.242	0	**0.04904**	0	131.755	0	26.4495
05	0	0.242	0	**0.04805**	0	133.765	0	26.9235
06	0	0.23	0	**0.04506**	0	132.227	0	26.9237
07	0	0.239	0	**0.04507**	0	132.8	0	26.658
08	0	0.246	0	**0.04608**	0	133.461	0	26.7901
09	0	0.248	0	**0.04909**	0	133.456	0	26.8966
10	0	0.241	0	**0.04510**	0	132.905	0	27.8815
11	0	0.237	0	**0.04811**	0	132.962	0	25.8112
12	0	0.233	0	**0.04912**	0	132.09	0	26.292
13	0	0.238	0	**0.04513**	0	133.457	0	26.9717
14	0	0.239	0	**0.04814**	0	132.633	0	27.2223
15	0	0.245	0	**0.04615**	0	132.887	0	27.2207
16	0	0.241	0	**0.04816**	0	132.83	0	26.727
17	0	0.237	0	**0.05117**	0	133.299	0	27.1919
18	0	0.26	0	**0.05418**	0	133.509	0	27.1689
19	0	0.245	0	**0.04619**	0	133.305	0	27.4925
avg	0	0.24205	0	**0.047845**	0.01	133.1582	0.021	26.95272
				100 jobs				

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
00	0	0.31	0	**0.06200**	0.1	230.01	0.01	48.633
01	0	0.3	0	**0.0601**	0	229.184	0	48.1774
02	0	0.319	0	**0.06102**	0	231.299	0	47.0789
03	0	0.307	0	**0.06203**	0	230.642	0	47.5332
04	0	0.316	0	**0.06304**	0	228.958	0	47.2558
05	0	0.307	0	**0.06205**	0	232.216	0	48.2126
06	0	0.306	0	**0.06106**	0	231.749	0	47.0799
07	0	0.318	0	**0.06307**	0	232.911	0	47.6831
08	0	0.319	0	**0.0608**	0	231.522	0	47.6472
09	0	0.305	0	**0.06109**	0	231.462	0	47.4992
10	0	0.304	0	**0.0610**	0	232.808	0.1	49.3598
11	0	0.309	0	**0.06211**	0	229.557	0	47.4597
12	0	0.304	0	**0.06112**	0	233.124	0	48.4454
13	0	0.311	0	**0.06413**	0	230.456	0	46.8776
14	0	0.317	0	**0.06114**	0	230.769	0	47.5969
15	0	0.308	0	**0.06115**	0	231.648	0	47.4248
16	0	0.323	0	**0.06216**	0	237.089	0	47.3749
17	0	0.306	0	**0.05917**	0	236.205	0.1	48.6945
18	0	0.314	0	**0.06218**	0	231.197	0	47.4597
19	0	0.309	0	**0.06219**	0	229.981	0	48.0761
avg	0	0.3106	0	**0.0616305**	0.005	231.63935	0.0105	47.778485
				200 jobs				
00	0	0.89	0	**0.17600**	0	1417.155	0	296.4585
01	0	0.901	0	**0.17801**	0	1418.13	0	304.857
02	0	0.903	0	**0.17902**	0	1415.718	0	294.6948
03	0	0.897	0	**0.17903**	0	1416.721	0	304.0391
04	0	0.906	0	**0.17804**	0	1421.808	0	297.7938
05	0	0.891	0	**0.17905**	0	1419.07	0	299.04
06	0	0.883	0	**0.17906**	0	1427.986	0	308.2376
07	0	0.914	0	**0.17807**	0	1424.364	0	302.5164
08	0	0.928	0	**0.17708**	0	1419.38	0	301.997
09	0	0.891	0	**0.1809**	0	1424.186	0	298.3316
10	0	0.909	0	**0.17910**	0	1431.471	0	298.2261
11	0	0.887	0	**0.17811**	0	1617.702	0	318.5242
12	0	0.904	0	**0.17612**	0	1421.839	0	299.8439
13	0	0.904	0	**0.17713**	0	1420.653	0	301.2093
14	0	0.906	0	**0.17914**	0	1413.461	0	299.5531
15	0	0.878	0	**0.1815**	0	1423.677	0	302.4437
16	0	0.905	0	**0.17716**	0	1422.081	0	303.4311
17	0	0.897	0	**0.17917**	0	1430.87	0	304.189
18	0	0.902	0	**0.17618**	0	1420.377	0	301.7157
19	0	0.904	0	**0.1819**	0	1420.012	0	303.2132
avg	0	0.9	0	**0.1784885**	0	1431.33305	0	302.015755

Table C.17: Experiment 3: Solutions and computation times of the best of (Experiment 2) vs de Paula et al. (2007)'s results (max grasp iterations waste = 10 , max vns iterations waste = 5 , timeout = inf). Algorithms minimizing the total weighted tardinesses on instances with six machines and loose due dates.

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
				20 jobs				
00	0	0.022	0	**0.01400**	0	5.831	0	2.1531
01	**243**	0.026	243	**0.01101**	243	5.843	267.3	2.1183
02	**49**	0.024	49	**0.01202**	49	5.816	53.9	1.7016
03	**50**.6	0.033	52.4	**0.01103**	50	5.811	61	2.3481
04	0	0.021	0	**0.0104**	0	6.162	0	2.3122
05	**380**	0.036	382.9	**0.01405**	380	5.847	434.6	2.0937
06	199.4	0.04	199.6	**0.01206**	**198**	6.275	287.9	2.0875
07	99.9	0.031	99.9	**0.01207**	**98**	6.309	126.8	3.2359
08	**144**	0.024	144	**0.01308**	144	6.26	158.4	2.115
09	**11**	0.026	11	**0.01209**	11	6.597	12.1	2.1477
10	0	0.022	0	**0.01110**	0	6.072	0	1.8502

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
11	0	0.021	0	**0.01111**	0	6.231	0	1.9801
12	**36**	0.03	36	**0.01212**	36	6.26	66.2	2.229
13	**16**	0.022	16	**0.01113**	16	6.195	17.6	2.3605
14	**46**	0.022	46	**0.01114**	46	6.256	50.6	1.8196
15	**146**	0.028	146	**0.01315**	146	6.372	160.6	2.0982
16	0	0.02	0	**0.0116**	0	5.799	0	1.6239
17	**109**	0.025	109	**0.01217**	109	5.791	119.9	2.3831
18	**144**	0.029	144	**0.01418**	144	5.86	158.4	2.226
19	**36**	0.026	36	**0.01219**	36	5.884	39.6	2.2124
avg	85.495	0.0264	85.74	**0.012085**	**85.3**	6.07355	100.745	2.154805
				40 jobs				
00	**4**	0.093	4	**0.0200**	4	26.936	4.4	11.6026
01	**128**	0.088	128	**0.02301**	128	27.445	140.8	9.2615
02	**27**	0.143	36.6	**0.0302**	27	27.36	90.9	10.849
03	0	0.145	0	**0.02903**	0.8	27.321	0.08	10.6221
04	**85**	0.11	87.2	**0.02404**	85	27.89	104.5	10.619
05	0	0.067	0	**0.02105**	0	27.284	0	11.6144
06	**32**	0.084	32	**0.0206**	32	29.491	35.2	8.8481
07	**105**	0.098	105	**0.0207**	105	29.431	115.5	10.6981
08	**250**	0.103	250	**0.0208**	250	28.989	275	13.6749
09	0	0.095	0	**0.02209**	0	29.549	30.4	10.0219
10	**60**	0.08	60	**0.0210**	60	28.817	66	12.3327
11	**109**	0.115	109	**0.02211**	109	29.183	119.9	10.5893
12	**90**	0.099	90	**0.02112**	90	28.891	99	9.9401
13	**114**	0.083	114	**0.0213**	114	29.211	125.4	10.2511
14	0	0.096	0	**0.02114**	0	28.358	0	10.9818
15	0	0.105	0	**0.0215**	0	27.282	47.9	10.0042
16	**221**	0.093	221	**0.02416**	221	27.531	243.1	9.8151
17	**37**	0.088	37	**0.0217**	37	27.02	50.6	9.234
18	0	0.08	0	**0.0218**	0	27.222	0	10.5412
19	0	0.116	0	**0.02319**	0	27.44	17.6	11.392
avg	**63.1**	0.09905	63.69	**0.022527**	63.14	28.13255	78.314	10.644655
				60 jobs				
00	**36**	0.237	36	**0.04100**	36	65.642	42.3	24.8552
01	**26**	0.2	26	**0.03701**	26	65.831	28.6	25.1871
02	**235.2**	0.529	278.7	**0.08902**	238.5	65.929	343.35	24.9009
03	**95**	0.293	99.4	**0.06203**	95	65.222	148.5	26.6872
04	**113**	0.207	113	**0.03904**	113	79.842	124.3	27.0262
05	**145**.9	0.517	176.3	**0.05705**	145	65.641	192.8	23.3361
06	0	0.172	0	**0.03406**	0	65.831	0	24.7611
07	**214**	0.228	214	**0.03807**	214	69.225	302.1	24.0775
08	0	0.372	12.6	**0.0508**	0	65.637	21	23.7877
09	**222**	0.275	223.4	**0.05409**	222	71.376	258.2	28.1536
10	**88**	0.185	88	**0.03910**	88	70.974	96.8	27.3364
11	**115**	0.311	115	**0.04511**	115	70.658	134.2	24.3408
12	**90**	0.321	90	**0.04712**	90	71.597	99	26.8807
13	3	0.268	3	**0.04213**	3	71.022	3.3	26.2672
14	196.2	0.425	202.2	**0.07614**	**193.6**	66.191	323.66	27.2411
15	**10**	0.18	10	**0.03915**	10	65.581	11	25.1801
16	**24**	0.239	24	**0.0416**	24	65.551	26.4	22.2491
17	**12**	0.167	12	**0.03717**	12	65.786	13.2	22.8616
18	225.4	0.454	226.6	**0.07418**	**220.4**	65.441	278.34	22.7751
19	**126**	0.222	126	**0.04219**	126	66.254	138.6	26.6564
avg	98.835	0.2901	103.81	**0.049303**	**98.575**	67.96155	129.2825	25.228055
				80 jobs				
00	11.4	0.622	14.4	**0.08400**	**9**	132.619	18.1	48.5969
01	0	0.356	0	**0.0601**	0	131.918	0	55.3778
02	**24**	0.38	24	**0.06702**	24	132.969	26.4	52.7979
03	**16**	0.667	16	**0.14403**	16	132.578	56.2	49.1908
04	0	0.367	0	**0.06604**	0	300.777	0	65.2607
05	**87**	0.511	87	**0.08805**	87	134.446	95.7	44.8286
06	0	0.351	0	**0.0706**	0	131.837	0	53.0747

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
07	0	0.393	0	**0.08107**	0	144.424	0	51.9374
08	**94**	0.423	94	**0.07708**	94	132.547	103.4	57.1357
09	**87**	0.382	87	**0.06709**	87	135.246	95.7	48.9606
10	**96**	0.719	102.6	**0.12710**	96	133.238	127.6	52.5568
11	**30**	0.555	30	**0.08711**	30	133.42	65	55.153
12	0	0.334	0	**0.06112**	0	142.593	0	53.4713
13	**333**	0.576	334.6	**0.10213**	333	145.206	381.3	55.3746
14	**15**	0.387	15	**0.0714**	15	131.059	16.5	67.2049
15	**33**	0.462	33	**0.08715**	33	141.907	36.3	56.9147
16	**172**	0.698	187.6	**0.11216**	172	131.667	204.8	46.9707
17	0	0.325	0	**0.0617**	0	132.349	0	53.5209
18	**88**	0.437	88	**0.07418**	88	132.933	96.8	49.4793
19	**298**.3	0.879	300.4	**0.11519**	298	131.967	330.8	58.2017
avg	69.235	0.4912	70.68	**0.085216**	**69.1**	143.285	82.73	53.80045
				100 jobs				
00	0	0.64	0	**0.1100**	0	229.313	0	91.3223
01	**74**	0.822	74	**0.13401**	74	230.517	81.4	93.9817
02	**39**	0.694	39	**0.13902**	39	227.383	42.9	87.3613
03	**18**	0.849	18	**0.11803**	18	227.144	19.8	94.9734
04	**138**	1.01	138	**0.18604**	138	229.23	169.8	98.632
05	**12**	0.844	12	**0.15105**	12	230.032	13.2	81.0322
06	**812.5**	3.275	838.5	**0.47606**	972	226.968	1081.1	91.9218
07	**33**	0.674	33	**0.107**	33	247.794	36.3	97.3864
08	**180**	0.746	180	**0.13308**	180	232.174	198	90.1904
09	**77**	1.191	85.1	**0.19909**	77	248.822	106.7	90.4192
10	**139**	0.73	139	**0.11610**	139	255.288	152.9	94.6078
11	**45**.1	1.299	45.5	**0.18911**	45	230.804	50.3	95.5754
12	102.9	1.535	107.7	**0.23412**	**101.4**	230.302	119.64	90.5402
13	0	0.628	0	**0.11813**	0	227.082	0	84.4302
14	**196**	0.737	196	**0.10714**	196	254.596	215.6	94.0816
15	0	0.666	0	**0.12615**	0	232.896	0	100.1726
16	**162**	1.036	162	**0.18916**	162	227.251	180.6	93.2551
17	**146**	0.791	146	**0.12717**	146	228.839	160.6	90.0289
18	0	0.687	0	**0.13518**	0	230.859	0	101.5599
19	0	0.94	0	**0.15219**	0	227.906	0	93.9526
avg	**108.725**	0.9897	110.69	**0.1623915**	116.62	233.76	131.442	92.77125
				200 jobs				
00	**10**	4.405	10	**0.81100**	10	1392.314	11	606.7954
01	**42**	4.562	42	**0.75901**	42	1398.964	46.2	624.7694
02	**124**	6.629	124	**0.96102**	124	1401.469	136.4	587.5809
03	**51**	4.22	51	**0.75803**	51	1391.505	56.1	638.0885
04	**24**	4.776	24	**0.79504**	24	1402.543	26.4	542.2463
05	**152**	8.175	152	**1.39705**	153.6	1393.071	167.36	561.8881
06	**75**	6.139	75	**1.0906**	75	1399.888	82.5	562.0078
07	**124**	9.032	124	**1.41507**	161.4	1536.598	140.14	578.0488
08	**27**	6.615	27	**1.17908**	27	1516.304	29.7	530.3834
09	**43**	8.164	45	**1.58209**	44.2	1452.115	71.42	604.4575
10	**12**	3.792	12	**0.60210**	12	1392.437	13.2	604.1157
11	**40**	6.396	40	**1.09211**	40	1404.945	44	622.1935
12	**322.1**	11.684	326.6	**2.48312**	330.2	1389.633	387.02	526.5763
13	**57**	5.871	57	**1.10613**	57	1391.821	62.7	572.2471
14	**74**	5.957	74	**1.17814**	74	1477.127	81.4	562.0727
15	**294.5**	14.867	305.6	**2.56215**	314.2	1492.946	353.42	548.2206
16	268	**0.009**	268	0.0116	**51**	1421.605	56.1	1565.7505
17	**57**	7.459	65.4	**0.82217**	57	1390.333	73.5	587.9573
18	**75**	6.45	75	**1.35918**	75	1395.702	82.5	508.7422
19	**8**	8.27	8	**1.84619**	8.7	1397.522	25.07	591.6012
avg	93.98	6.6736	95.28	**1.190544**	**86.565**	1421.9421	97.3065	626.28716

Table C.18: Experiment 3: Solutions and computation times of the best of (Experiment 2) vs de Paula et al. (2007)'s results (max grasp iterations waste = 10 , max vns iterations waste = 5 , timeout = inf). Algorithms minimizing the total weighted tardinesses on instances with six machines and tight due dates.

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
				20 jobs				
00	**1400**	0.047	1406.8	**0.01700**	1416	5.73	1687.6	1.356
01	**1805.5**	0.068	1807.2	**0.02201**	1811.7	5.86	2057.07	1.425
02	**1676**	0.055	1686.2	**0.01802**	1697	5.786	1877.4	1.3706
03	**1905.3**	0.061	1906.4	**0.01903**	1930	5.725	2190.9	1.3055
04	**1662**	0.045	1667.6	**0.01604**	1696.2	6.151	1879.72	1.4301
05	**2000.5**	0.044	2009.2	**0.01905**	2004.8	5.756	2243.18	1.3776
06	**2138**	0.054	2138	**0.0206**	2142.3	6.092	2453.33	1.4732
07	**1744.6**	0.046	1755.5	**0.01707**	1760.4	6.214	2113.04	1.5904
08	**2001**	0.043	2005.3	**0.01808**	2011.3	6.242	2230.33	1.4812
09	**1462**	0.044	1462	**0.01809**	1476.8	6.225	1639.18	1.4635
10	**1650.3**	0.048	1653.3	**0.01910**	1662.4	6.176	1909.44	1.3596
11	**1695**	0.044	1700.6	**0.01811**	1706	6.213	1915.3	1.4183
12	**1558**	0.047	1562.4	**0.01412**	1589.3	6.195	1762.83	1.4745
13	**1351**	0.052	1362	**0.01513**	1410.7	6.195	1524.17	1.3675
14	**1551**	0.065	1564.1	**0.01614**	1590.9	6.173	1767.19	1.4123
15	**1436**	0.042	1444	**0.01915**	1474.6	6.189	1603.06	1.4739
16	**2144**	0.04	2144	**0.01316**	2147.3	5.745	2372.13	1.3275
17	**1946**	0.044	1949	**0.01617**	1946.8	5.694	2342.48	1.3734
18	**2002**	0.043	2002	**0.01418**	2009	5.754	2460.9	1.3284
19	**2176**	0.042	2204.8	**0.01819**	2176	5.767	2457.1	1.3857
avg	**1765.21**	0.0487	1771.52	**0.017422**	1782.975	5.9941	2024.3175	1.40971
				40 jobs				
00	**5842.6**	0.469	5955.9	**0.07900**	6212.3	27.034	6530.33	6.8854
01	**4791.2**	0.472	4848.9	**0.08701**	5038.7	27.21	5403.07	6.77
02	**4962**	0.36	4994.1	**0.07102**	5220.6	27.289	5580.36	6.9309
03	**8384.5**	0.551	8483.6	**0.06903**	8847.5	26.711	9456.85	6.8001
04	**6145.2**	0.668	6192.1	**0.10304**	6468.5	27.654	7041.05	6.7754
05	**4436.4**	0.492	4492.3	**0.06805**	4884.2	27.067	5281.52	6.8277
06	**5232.5**	0.499	5323	**0.07906**	5664.6	27.008	5998.26	6.8308
07	**5712.5**	0.466	5804.2	**0.07407**	6021.4	30.816	6655.94	7.3006
08	**5296.7**	0.574	5353.8	**0.08208**	5715.1	29.263	5991.21	6.9683
09	**5575.7**	0.543	5642.5	**0.0809**	5846.7	29.299	6293.77	7.1659
10	**5429.4**	0.608	5521.8	**0.09610**	5797.5	29.385	6177.95	7.1125
11	**6631.4**	0.506	6717.7	**0.08111**	6914.8	28.071	7526.18	7.0081
12	**5317.3**	0.533	5348.4	**0.07912**	5778.2	28.697	5956.22	6.9927
13	**4956.7**	0.516	5026	**0.07113**	5197.7	29.298	5534.77	6.7898
14	**5924.2**	0.678	6026.4	**0.08914**	6257.8	27.537	6704.28	6.8947
15	**6124.2**	0.468	6188.6	**0.11115**	6513.9	29.205	6866.19	6.8785
16	**7748.2**	0.547	7860.6	**0.07216**	8209.4	26.791	8706.64	6.7581
17	**5771.5**	0.496	5790.7	**0.08617**	6100.4	27.061	6416.84	6.7891
18	**4759.4**	0.56	4856.8	**0.08518**	5134	26.964	5476.4	6.7764
19	**7712.1**	0.607	7805.8	**0.119**	8119	27.047	8598.7	6.7667
avg	**5837.685**	0.53065	5911.66	**0.084176**	6197.115	27.97035	6609.8265	6.901085
				60 jobs				
00	**9801.7**	2.078	9910.8	**0.32400**	11219.7	64.804	11045.47	16.5844
01	**8671.5**	2.211	8905.5	**0.3101**	10042.5	65.371	9843.75	16.5211
02	**14591.1**	2.383	14866.5	**0.29302**	15862.9	65.687	16299.49	16.5797
03	**11366.9**	1.806	11662.4	**0.2503**	13107.8	65.023	13071.38	16.2833
04	**11553.5**	2.552	11826.9	**0.31604**	12905.1	79.929	13112.71	18.1739
05	**9511.3**	1.878	9688.9	**0.22505**	10513.4	65.542	10606.24	16.8442
06	**9431.6**	3.116	9652.4	**0.23106**	10804.3	66.504	10679.63	17.0774
07	**12059.9**	2.708	12317.7	**0.32107**	13675.8	65.054	13627.68	17.0894
08	**12246.8**	2.139	12500.4	**0.3308**	13403.2	71.24	13794.32	17.529
09	**12082.9**	2.279	12255.5	**0.2709**	13006.6	70.958	13548.66	17.1468
10	**9988.1**	2.321	10217.2	**0.29910**	11431.3	70.474	11331.73	17.3634
11	**11516.1**	2.495	11752.2	**0.26711**	12782.9	70.721	13274.99	17.1211
12	**11646.2**	1.942	11786.7	**0.30212**	12991.1	70.285	13000.91	17.5515
13	**10779.5**	2.623	10984.6	**0.27313**	11837.8	71.2	12156.78	16.942
14	**14508.1**	2.978	14714.4	**0.24414**	15451.7	65.977	16271.37	16.7287

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
15	**10911.5**	2	11101.9	**0.28115**	12256.6	65.619	12304.26	16.4529
16	**13855**	2.85	14076.8	**0.34916**	15143.4	65.105	15523.74	16.8885
17	**9123.7**	2.198	9393.4	**0.33817**	10928.4	65.585	10524.24	16.3605
18	**10520.8**	2.167	10612.6	**0.42218**	12020.9	64.785	11918.09	16.4975
19	**14521.8**	2.689	14696	**0.2819**	15631.3	65.404	16336.13	16.6694
avg	**11434.4**	2.37065	11646.14	**0.296525**	12750.835	67.76335	12913.5785	16.920235
				80 jobs				
00	**18125.8**	6.396	18465.1	**0.85500**	20542.5	132.785	20400.95	35.5985
01	**16635.4**	5.777	16995.4	**0.94701**	19890.1	132.171	18778.11	35.1231
02	**19451.5**	6.98	19981.4	**0.88402**	24087.9	131.802	22155.29	34.4322
03	**19406.6**	6.018	19971.4	**1.06403**	22767.4	132.457	22091.04	35.0797
04	**20312.9**	6.867	20732.2	**0.77404**	24116.7	131.999	22966.67	35.8219
05	**17445.4**	7.834	17784.9	**0.81705**	20770	132.389	19663.3	35.7349
06	**19005.8**	6.754	19617.8	**0.80106**	22447.3	130.899	21635.23	35.1379
07	**21606**	5.62	22064.3	**0.74207**	25752	139.526	24692.5	35.7146
08	**22457.3**	7.747	22919.2	**0.89308**	25613.7	133.256	25138.67	35.6366
09	**19125.9**	6.97	19618.9	**0.72909**	22240.8	133.78	21690.68	35.326
10	**18447.2**	7.747	18748	**0.76610**	21652.5	144.546	20707.15	36.8876
11	**19360.6**	7.693	19910.2	**0.92711**	22996.3	134.117	22086.53	34.5727
12	**15180.2**	7.142	15666.3	**0.73712**	18492.9	132.253	17526.59	34.8903
13	**17174.2**	6.931	17411	**0.50713**	19593.5	144.756	19551.75	36.6336
14	**19970.2**	7.95	20287.5	**1.0514**	23320.1	132.548	22627.51	35.5668
15	**20624.1**	7.364	21183	**0.76215**	23917.6	132.749	23426.96	35.3549
16	**24411.2**	6.563	24751.2	**0.65816**	27541.7	131.388	27426.07	35.0388
17	**22011.8**	9.764	22462	**0.95117**	24529.2	130.705	24864.22	35.4425
18	**17517.1**	8.091	17901.9	**1.05918**	20757.2	131.378	20112.02	35.1218
19	**21769**	6.976	22243.9	**0.94219**	25007	131.702	24670.2	35.0122
avg	**19501.91**	7.1592	19935.78	**0.843408**	22801.82	133.8603	22110.572	35.40633
				100 jobs				
00	**26705.1**	13.399	27499.7	**1.75300**	33271.3	228.285	30598.53	62.8405
01	**24531.3**	20.057	25459.6	**1.95101**	31017.5	232.088	28415.65	62.6878
02	**26876.1**	15.178	27549.9	**1.22302**	33327.7	231.215	30610.87	62.7435
03	**33338.4**	18.326	34256.6	**1.97803**	39346	229.119	37479.2	62.7819
04	**31927.1**	14.023	32839.9	**2.0104**	39781.2	231.085	36382.92	64.1675
05	**27919**	14.948	28617.5	**1.9505**	34091.2	230.194	31508.42	62.5174
06	**32199.7**	17.494	33149.4	**1.64706**	39364.4	230.312	36738.44	63.1972
07	**22580.4**	15.216	23126.8	**1.12407**	28961.1	229.895	25654.71	63.2335
08	**29789.3**	13.14	30490.4	**2.05608**	35729.5	230.735	33733.35	63.4245
09	**32022.8**	17.127	32833.1	**1.46609**	36617.3	253.545	35984.13	65.3385
10	**23221.3**	16.691	23630.4	**1.910**	29425	250.82	26156	64.883
11	**29449.3**	18.833	30196.5	**2.32811**	35420.2	232.63	33369.72	62.669
12	**33156.6**	16.252	34143.4	**2.20912**	39997.2	235.876	38518.32	63.2806
13	**23536.2**	18.281	24377	**2.50613**	28814	231.591	27118.3	63.3951
14	**34172.3**	18.214	35009.2	**1.36514**	40133.6	228.669	38587.96	63.1379
15	**27452.5**	12.865	28328.3	**2.18415**	33414.1	231.733	31036.41	63.1533
16	**28249.3**	15.214	29110.1	**1.90916**	35006.4	228.228	32062.84	62.4198
17	**31059.4**	14.901	31919.2	**1.98817**	37522.7	229.66	35947.97	62.625
18	**25846.4**	15.178	26568.6	**1.86218**	31394.6	229.846	29320.46	63.2656
19	**31042.1**	14.882	31747.9	**1.91619**	36914.9	229.017	35220.99	62.2217
avg	**28753.73**	16.01095	29542.675	**1.8668805**	34977.495	232.72715	32722.2595	63.199165
				200 jobs				
00	**93417.3**	236.65	96525.5	**15.37800**	131290	1427.54	107173.7	398.45
01	**84970.9**	200.905	87983.3	**28.79701**	120227.2	1430.37	98406.02	397.866
02	**92958.7**	213.292	96532.2	**21.84202**	129437.5	1417.202	107365.25	394.1942
03	**104530.8**	239.379	108078.4	**26.06103**	145610.5	1598.265	120608.35	411.2355
04	**109011.8**	218.763	113091.9	**22.74304**	150806.9	1434.332	124747.19	398.6332
05	**92981.5**	169.873	96017	**14.86405**	124249.3	1464.462	104761.73	402.6232
06	**106362.6**	217.796	110275.8	**29.28706**	145514.6	1519.523	122668.46	405.8123
07	**93894.4**	171.196	97057.7	**18.79807**	131257.4	1457.358	107568.44	398.8178
08	**100404.9**	216.646	103971	**27.89308**	141371.3	1421.293	116468.13	395.5863
09	**119688.5**	225.878	122558	**15.54309**	156145.9	1441.264	135446.39	401.1874
10	**92773.1**	171.91	96074.4	**25.40710**	130648.2	1496.24	106271.02	404.227

	grasp+vns		grasp+swapUmove		old grasp		old vns	
Inst	sol	time	sol	time	sol	time	sol	time
11	**100095.6**	212.132	103327.6	**18.83511**	138755.5	1424.12	115087.85	396.629
12	**97933.9**	193.933	101095.6	**21.97412**	133972	1412.924	111866.6	392.3914
13	**91487.2**	207.339	94920.2	**21.08213**	125458.5	1423.454	105065.55	398.0514
14	**104777.8**	172.003	108082.4	**25.14614**	140124.1	1563.758	119218.81	410.7858
15	**111851.4**	208.1	116166.8	**17.55515**	151270.2	1424.505	127734.22	396.9285
16	**104617.7**	207.889	108884	**21.22116**	145568.4	1421.626	120684.74	397.2336
17	**99937.4**	211.28	103099.7	**27.91517**	136215.6	1424.321	114351.76	396.5281
18	**106352.1**	219.793	109951.3	**19.46118**	147056.4	1419.436	121465.74	395.1276
19	**109668.8**	216.128	113229.4	**25.18819**	146944.8	1413.219	125481.88	395.7939
avg	**100885.82**	206.54425	104346.11	**22.249595**	138596.215	1451.7606	115622.0915	399.40511

C.2 Experiments with Lower Bounds

Table C.19: Lower bounds obtained with the Linear Programming relaxation of formulations based on Manne (1960)'s (MLP) and Wagner (1959)'s (WLP) works for problems minimizing the makespan on instances with six machines and loose due dates.

Inst	**MLP**		**WLP**	
	LB	**Time (s)**	**LB**	**Time (s)**
		6 jobs		
00	0.00	0.00	26	0.00
01	0.00	0.00	27	0.01
02	0.00	0.01	42	0.01
03	0.00	0.02	36	0.00
04	0.00	0.01	39	0.01
05	0.00	0.02	24	0.01
06	0.00	0.00	32	0.01
07	0.00	0.01	44	0.00
08	0.00	0.01	32	0.01
09	0.00	0.00	33	0.01
10	0.00	0.01	30	0.00
11	0.00	0.01	35	0.01
12	0.00	0.00	46	0.01
13	0.00	0.00	33	0.00
14	0.00	0.00	36	0.02
15	0.00	0.00	45	0.00
16	0.00	0.02	39	0.01
17	0.00	0.01	29	0.01
18	0.00	0.00	40	0.01
19	0.00	0.01	39	0.00
		8 jobs		
00	0.00	0.00	41	0.02
01	0.00	0.01	53	0.01
02	0.00	0.01	43	0.02
03	0.00	0.02	50	0.01
04	0.00	0.00	70	0.02
05	0.00	0.00	44	0.01
06	0.00	0.03	51	0.01
07	0.00	0.01	59	0.02
08	0.00	0.01	48	0.02
09	0.00	0.00	67	0.03
10	0.00	0.01	62	0.02
11	0.00	0.01	43	0.02
12	0.00	0.00	63	0.01
13	0.00	0.00	50	0.02
14	0.00	0.02	36	0.02
15	0.00	0.00	57	0.02
16	0.00	0.01	43	0.02
17	0.00	0.01	43	0.02

Inst	MLP		WLP	
	LB	Time (s)	LB	Time (s)
18	0.00	0.01	53	0.02
19	0.00	0.01	48	0.03
		10 jobs		
00	0.00	0.02	55	0.05
01	0.00	0.00	58	0.04
02	0.00	0.00	62	0.04
03	0.00	0.00	85	0.03
04	0.00	0.01	71	0.03
05	0.00	0.00	57	0.03
06	0.00	0.00	71	0.03
07	0.00	0.01	63	0.03
08	0.00	0.01	49	0.04
09	0.00	0.01	56	0.04
10	0.00	0.01	63	0.04
11	0.00	0.01	49	0.04
12	0.00	0.01	65	0.03
13	0.00	0.00	63	0.03
14	0.00	0.00	47	0.04
15	0.00	0.00	79	0.03
16	0.00	0.01	61	0.04
17	0.00	0.01	50	0.04
18	0.00	0.00	78	0.03
19	0.00	0.00	72	0.05
		12 jobs		
00	0.00	0.02	61	0.06
01	0.00	0.01	57	0.06
02	0.00	0.01	65	0.07
03	0.00	0.00	67	0.06
04	0.00	0.00	79	0.06
05	0.00	0.00	76	0.06
06	0.00	0.02	76	0.07
07	0.00	0.01	69	0.06
08	0.00	0.01	67	0.06
09	0.00	0.01	65	0.06
10	0.00	0.01	75	0.06
11	0.00	0.03	90	0.06
12	0.00	0.01	82	0.07
13	0.00	0.01	59	0.06
14	0.00	0.01	63	0.06
15	0.00	0.01	73	0.05
16	0.00	0.01	74	0.07
17	0.00	0.02	64	0.06
18	0.00	0.01	73	0.07
19	0.00	0.02	84	0.05
		14 jobs		
00	0.00	0.01	94	0.10
01	0.00	0.01	85	0.10
02	0.00	0.02	88	0.12
03	0.00	0.01	91	0.10
04	0.00	0.01	88	0.11
05	0.00	0.01	87	0.10
06	0.00	0.01	85	0.10
07	0.00	0.02	88	0.10
08	0.00	0.01	74	0.10
09	0.00	0.02	86	0.10
10	0.00	0.01	113	0.10
11	0.00	0.00	89	0.11
12	0.00	0.02	81	0.10
13	0.00	0.02	62	0.10
14	0.00	0.01	79	0.11
15	0.00	0.01	99	0.10
16	0.00	0.00	94	0.10

Inst	MLP		WLP	
	LB	Time (s)	LB	Time (s)
17	0.00	0.02	73	0.11
18	0.00	0.00	87	0.12
19	0.00	0.01	88	0.11
		16 jobs		
00	0.00	0.01	80	0.18
01	0.00	0.00	88	0.17
02	0.00	0.01	110	0.17
03	0.00	0.01	109	0.16
04	0.00	0.01	107	0.15
05	0.00	0.00	111	0.17
06	0.00	0.00	98	0.18
07	0.00	0.02	85	0.19
08	0.00	0.01	104	0.18
09	0.00	0.01	123	0.16
10	0.00	0.01	99	0.17
11	0.00	0.01	108	0.18
12	0.00	0.01	82	0.17
13	0.00	0.01	69	0.17
14	0.00	0.01	62	0.17
15	0.00	0.02	107	0.17
16	0.00	0.01	102	0.17
17	0.00	0.01	80	0.19
18	0.00	0.01	106	0.18
19	0.00	0.01	103	0.18

Table C.20: Lower bounds obtained with the Linear Programming relaxation of formulations based on Manne (1960)'s (MLP) and Wagner (1959)'s (WLP) works for problems minimizing the makespan on instances with six machines and tight due dates.

Inst	MLP		WLP	
	LB	Time (s)	LB	Time (s)
		6 jobs		
00	0	0.01	26	0.01
01	0	0.01	27	0.01
02	0	0.03	42	0.01
03	0	0.01	36	0.01
04	0	0	39	0.01
05	0	0	24	0.01
06	0	0.01	32	0
07	0	0.01	44	0.01
08	0	0.01	32	0.01
09	0	0	33	0.01
10	0	0	30	0.01
11	0	0.01	35	0.01
12	0	0.02	46	0.01
13	0	0.01	33	0.02
14	0	0	36	0.01
15	0	0.02	45	0.01
16	0	0	39	0.01
17	0	0.01	29	0.01
18	0	0.01	40	0.02
19	0	0	39	0.01
		8 jobs		
00	0	0	41	0.02
01	0	0.01	53	0.02
02	0	0	43	0.02
03	0	0.01	50	0.02
04	0	0	70	0.02
05	0	0	44	0.02
06	0	0.01	51	0.01
07	0	0.01	59	0.03

Inst	MLP		WLP	
	LB	Time (s)	LB	Time (s)
08	0	0	48	0.01
09	0	0.01	67	0.01
10	0	0.01	62	0.02
11	0	0.02	43	0.01
12	0	0.02	63	0.02
13	0	0	50	0.03
14	0	0	36	0.01
15	0	0	57	0.01
16	0	0	43	0.02
17	0	0	43	0.01
18	0	0	53	0.03
19	0	0	48	0.02
10 jobs				
00	0	0	55	0.03
01	0	0	58	0.04
02	0	0	62	0.03
03	0	0	85	0.03
04	0	0.01	71	0.04
05	0	0.01	57	0.03
06	0	0.03	71	0.03
07	0	0.02	63	0.03
08	0	0	49	0.03
09	0	0.01	56	0.04
10	0	0.02	63	0.03
11	0	0.01	49	0.03
12	0	0.02	65	0.03
13	0	0.02	63	0.05
14	0	0.01	47	0.04
15	0	0	79	0.03
16	0	0.01	61	0.03
17	0	0.01	50	0.03
18	0	0.01	78	0.04
19	0	0	72	0.06
12 jobs				
00	0	0.02	61	0.06
01	0	0	57	0.06
02	0	0.01	65	0.07
03	0	0.02	67	0.08
04	0	0.02	79	0.06
05	0	0.01	76	0.07
06	0	0	76	0.07
07	0	0	69	0.06
08	0	0.01	67	0.06
09	0	0.01	65	0.06
10	0	0.03	75	0.06
11	0	0.01	90	0.07
12	0	0.01	82	0.08
13	0	0	59	0.07
14	0	0.02	63	0.07
15	0	0.01	73	0.06
16	0	0.01	74	0.07
17	0	0	63	0.06
18	0	0	73	0.06
19	0	0.01	84	0.06
14 jobs				
00	0	0.01	94	0.11
01	0	0.01	85	0.10
02	0	0.02	88	0.10
03	0	0.01	91	0.11
04	0	0.01	88	0.12
05	0	0.01	87	0.11
06	0	0.01	85	0.11

Inst	MLP		WLP	
	LB	Time (s)	LB	Time (s)
07	0	0.02	88	0.11
08	0	0.01	74	0.11
09	0	0.01	86	0.10
10	0	0.01	113	0.11
11	0	0.01	89	0.10
12	0	0	81	0.11
13	0	0.01	62	0.10
14	0	0.01	79	0.11
15	0	0.01	99	0.11
16	0	0.01	94	0.11
17	0	0.01	73	0.11
18	0	0	87	0.11
19	0	0	88	0.10
		16 jobs		
00	0	0.01	80	0.17
01	0	0	88	0.17
02	0	0.01	110	0.15
03	0	0.01	109	0.17
04	0	0.01	107	0.16
05	0	0.02	111	0.18
06	0	0.01	98	0.18
07	0	0.02	85	0.21
08	0	0	104	0.18
09	0	0.02	123	0.17
10	0	0	99	0.17
11	0	0.01	108	0.18
12	0	0.01	82	0.18
13	0	0.01	69	0.16
14	0	0.01	62	0.18
15	0	0.01	107	0.18
16	0	0	102	0.18
17	0	0.02	80	0.20
18	0	0.01	106	0.18
19	0	0	103	0.18

Table C.21: Lower bounds obtained with the Linear Programming relaxation of formulations based on Manne (1960)'s (MLP) and Wagner (1959)'s (WLP) works, and with the time-indexed formulation proposed by Ravetti (2007) (TLP), for problems minimizing the total weighted tardiness on instances with six machines and loose due dates.

Inst	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
				6 jobs				
00	9	0.79	0	0	0	0.01	9	0.26
01	47	0.02	0	0	0	0.01	40	0
02	28	4.09	0	0	0	0.02	28	0.54
03	199	0.06	0	0.01	0	0.01	159	0.03
04	21	2.19	0	0	0	0.01	21	0.36
05	0	0.12	0	0	0	0.01	0	0.06
06	113	0.01	0	0.02	0	0.01	113	0.01
07	107	0.63	0	0	0	0.01	56	0.31
08	0	4.79	0	0.01	0	0	0	0.63
09	14	0.05	0	0	0	0.02	10	0.02
10	41	0.07	0	0.02	0	0	41	0.03
11	0	0.49	0	0	0	0	0	0.15
12	120	1.24	0	0	0	0.01	120	0.40
13	7	0.40	0	0.01	0	0.01	0	0.11
14	0	9.87	0	0	0	0	0	1.39
15	51	2.27	0	0.02	0	0	51	0.37
16	20	0.31	0	0	0	0.01	20	0.07
17	0	1.97	0	0.01	0	0.02	0	0.19
18	21	3.83	0	0.01	0	0	21	0.41

Inst	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
19	0	4.13	0	0.01	0	0	0	0.55
				8 jobs				
00	25	1.52	0	0.01	0	0.02	25	0.24
01	0	22.69	0	0.01	0	0.02	0	2.92
02	0	21.27	0	0.01	0	0.01	0	3.16
03	190	5.89	0	0.01	0	0.03	128	1.19
04	12	62.71	0	0	0	0.02	12	10.38
05	0	30.91	0	0	0	0.02	0	4.17
06	72	1.82	0	0	0	0.01	0	0.23
07	102	11.72	0	0.01	0	0.02	68	1.89
08	0	2.70	0	0.01	0	0.01	0	0.30
09	92	46.29	0	0	0	0.02	92	7.80
10	10	27.40	0	0	0	0.01	10	5.86
11	70	7.87	0	0	0	0.01	70	1.22
12	100	8.87	0	0.01	0	0.01	81	2.03
13	32	31.32	0	0	0	0.02	32	5.44
14	0	28.08	0	0	0	0.01	0	3.81
15	56	8.51	0	0.01	0	0.02	56	1.52
16	126	6.93	0	0	0	0.02	126	1.20
17	64	2.56	0	0	0	0.02	64	0.47
18	0	3.31	0	0	0	0.02	0	0.59
19	74	4.75	0	0	0	0.01	22	1.05
				10 jobs				
00	30	39.82	0	0.01	0	0.03	30	7.06
01	36	15.78	0	0.01	0	0.04	36	2.52
02	275	20.50	0	0	0	0.03	241	3.59
03	10	81.20	0	0.01	0	0.04	10	15.41
04	42	66.38	0	0.01	0	0.03	42	11.75
05	95	19.45	0	0	0	0.03	29	3.70
06	67	89.30	0	0	0	0.02	67	16.92
07	0	10.66	0	0	0	0.03	0	1.46
08	33	32.72	0	0.01	0	0.04	3	6.73
09	56	103.76	0	0.01	0	0.03	8	20.71
10	18	90.42	0	0.01	0	0.03	0	11.44
11	95	29.18	0	0.01	0	0.03	81	4.98
12	188	38.48	0	0.02	0	0.06	183	8.79
13	54	72.12	0	0.01	0	0.02	54	13.02
14	0	9.03	0	0	0	0.03	0	1.41
15	4	123.02	0	0	0	0.03	3	23.43
16	0	64.37	0	0.01	0	0.02	0	9.63
17	165	18.27	0	0.02	0	0.02	80	3.14
18	0	100.56	0	0.01	0	0.04	0	14.90
19	0	108.60	0	0.01	0	0.03	0	15.97
				12 jobs				
00	0	148.78	0	0.01	0	0.05	0	19.98
01	0	54.43	0	0.01	0	0.06	0	7.78
02	11	118.43	0	0.01	0	0.07	11	21.18
03	216	88.58	0	0	0	0.06	171	20.53
04	96	229.25	0	0	0	0.07	96	40.64
05	20	196.71	0	0.01	0	0.05	18	41.54
06	61	184.30	0	0	0	0.05	61	31.24
07	0	162.12	0	0.01	0	0.06	0	23.06
08	89	63.79	0	0	0	0.05	34	14.69
09	204	101.26	0	0	0	0.06	131	18.21
10	53	133.12	0	0.01	0	0.04	21	22.03
11	0	250.39	0	0.03	0	0.06	0	31.87
12	0	219.62	0	0.01	0	0.06	0	30.09
13	8	127.54	0	0.01	0	0.05	8	16.93
14	102	135.38	0	0.01	0	0.05	102	25.77
15	26	91.34	0	0.01	0	0.05	26	16.17
16	156	104.52	0	0	0	0.04	156	21.35
17	0	232.53	0	0.01	0	0.06	0	31.86

Inst	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
18	120	112.43	0	0.02	0	0.06	120	20.64
19	249	118.02	0	0	0	0.05	249	23.54
				14 jobs				
00	0	780.89	0	0.01	0	0.08	0	106.87
01	54	159.27	0	0.01	0	0.10	54	27.82
02	156	254.95	0	0.01	0	0.09	156	43.83
03	165	232.37	0	0.01	0	0.09	165	46.08
04	7	527.94	0	0	0	0.10	7	77.07
05	0	519.81	0	0.02	0	0.09	0	66.54
06	145	229.08	0	0	0	0.11	68	53
07	0	319.77	0	0.01	0	0.10	0	41.55
08	0	564.73	0	0	0	0.09	0	76.11
09	0	447.32	0	0.01	0	0.09	0	60.43
10	68	477.39	0	0.01	0	0.09	68	87.94
11	74	489.46	0	0	0	0.11	74	89.84
12	0	155.35	0	0	0	0.09	0	21.02
13	54	288.58	0	0.01	0	0.08	54	53.40
14	72	244.20	0	0.02	0	0.09	32	82.71
15	0	425.65	0	0	0	0.13	0	60.69
16	0	407.11	0	0	0	0.08	0	61.26
17	0	286.68	0	0	0	0.10	0	42.25
18	60	543.67	0	0.02	0	0.08	60	103.34
19	0	377.07	0	0.01	0	0.09	0	45.06
				16 jobs				
00	78	45.06	0	0.02	0	0.19	21	49.98
01	0	578.25	0	0.01	0	0.18	0	79.56
02	0	710.47	0	0.02	0	0.19	0	110.62
03	-	7200	0	0	0	0.24	270	78.05
04	0	999.71	0	0.02	0	0.13	0	134.88
05	62	753.21	0	0.01	0	0.12	62	130.53
06	0	540.79	0	0	0	0.19	0	73.06
07	155	2448.14	0	0.01	0	0.15	14	137.76
08	88	539.94	0	0.01	0	0.20	88	118.40
09	45	1060.41	0	0.01	0	0.16	-	7200
10	0	825.55	0	0.01	0	0.16	0	127.48
11	158	936.99	0	0.01	0	0.16	158	168.90
12	198	670.37	0	0.01	0	0.16	198	54.51
13	18	187.77	0	0	0	0.20	18	32.81
14	0	295.49	0	0.02	0	0.15	0	45.24
15	37	45.24	0	0	0	0.18	29	71.51
16	90	511.19	0	0.02	0	0.19	90	95.38
17	0	328.48	0	0.01	0	0.14	0	44.81
18	14	569.86	0	0	0	0.16	14	89.84
19	6	591.56	0	0.01	0	0.15	6	107.11

Table C.22: Lower bounds obtained with the Linear Programming relaxation of formulations based on Manne (1960)'s (MLP) and Wagner (1959)'s (WLP) works, and with the time-indexed formulation proposed by Ravetti (2007) (TLP), for problems minimizing the total weighted tardiness on instances with six machines and tight due dates.

Inst	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
				6 jobs				
00	136	107.11	0	0	0	0.02	72	0.07
01	131	0.01	0	0	0	0.01	113	0.01
02	191	0.08	0	0.01	0	0.01	128	0.06
03	262	0.05	0	0	0	0.02	222	0.04
04	138	0.41	0	0.01	0	0.02	85	0.24
05	103	0.02	0	0.01	0	0.01	77	0.01
06	300	0.03	0	0.01	0	0	300	0.01
07	462	0.38	0	0	0	0.01	325	0.19
08	224	0.46	0	0.01	0	0.02	133	0.14
09	176	0.04	0	0.01	0	0.01	110	0.04

Inst	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
10	238	0.03	0	0.02	0	0.01	181	0.01
11	151	0.06	0	0	0	0.01	135	0.02
12	362	0.09	0	0.02	0	0.02	329	0.06
13	200	0.12	0	0.01	0	0.01	111	0.07
14	235	0.07	0	0.01	0	0.01	220	0.09
15	209	0.09	0	0	0	0.02	186	0.08
16	163	0.32	0	0	0	0.01	128	0.08
17	241	0.35	0	0	0	0	179	0.08
18	251	0.14	0	0	0	0	179	0.06
19	179	0.07	0	0	0	0.01	115	0.05
				8 jobs				
00	359	1.65	0	0.01	0	0.01	220	0.30
01	259	6.52	0	0.01	0	0.01	89	1.28
02	148	0.38	0	0.01	0	0.02	49	0.12
03	370	1	0	0.01	0	0.03	237	0.26
04	642	2.78	0	0	0	0.02	386	0.43
05	253	5.98	0	0	0	0.03	55	0.99
06	538	2.32	0	0	0	0.01	297	0.50
07	689	3.01	0	0.01	0	0.02	466	0.68
08	317	2.61	0	0.01	0	0.02	205	0.33
09	573	8.68	0	0.01	0	0.01	446	1.83
10	400	4.05	0	0.01	0	0.02	232	0.42
11	382	1.43	0	0.01	0	0.02	259	0.37
12	699	1.65	0	0.01	0	0.01	444	0.57
13	238	4.96	0	0	0	0.02	90	0.68
14	361	0.68	0	0.01	0	0.01	277	0.42
15	438	4.42	0	0	0	0.02	244	0.79
16	527	3.07	0	0	0	0.03	307	0.42
17	470	0.36	0	0	0	0.01	242	0.15
18	388	0.43	0	0.01	0	0.02	219	0.25
19	621	4.47	0	0.01	0	0.02	390	1.08
				10 jobs				
00	438	16.17	0	0.01	0	0.04	154	2.93
01	364	10.91	0	0.01	0	0.04	136	2.13
02	838	9.01	0	0.01	0	0.03	511	1.65
03	611	8.08	0	0.01	0	0.03	346	1.94
04	669	6.53	0	0	0	0.04	330	0.90
05	662	8.88	0	0.01	0	0.03	184	1.92
06	745	14.95	0	0	0	0.04	382	3.45
07	680	6.54	0	0	0	0.03	296	1.70
08	545	6.52	0	0	0	0.03	189	1.48
09	424	0.71	0	0.02	0	0.03	192	0.24
10	691	5	0	0	0	0.04	358	0.90
11	481	2.48	0	0.01	0	0.04	178	0.70
12	830	4.29	0	0	0	0.03	402	0.95
13	491	9.84	0	0	0	0.03	213	2.62
14	494	6.14	0	0.01	0	0.04	125	1.69
15	590	5.69	0	0.01	0	0.03	284	1.33
16	454	10.40	0	0.01	0	0.05	222	1.72
17	675	6.39	0	0.01	0	0.03	319	1.32
18	582	65.39	0	0.01	0	0.04	288	7.56
19	467	17.84	0	0	0	0.03	148	1.93
				12 jobs				
00	477	12.47	0	0.01	0	0.06	237	2.11
01	613	15.35	0	0.01	0	0.07	137	2.85
02	458	58.29	0	0	0	0.06	142	5.82
03	878	51.25	0	0.01	0	0.06	286	4.07
04	719	55.52	0	0.01	0	0.06	285	6.97
05	735	42.52	0	0.01	0	0.07	262	5.42
06	581	117.80	0	0	0	0.06	166	16.09
07	959	9.01	0	0	0	0.06	368	2.63
08	830	73.41	0	0	0	0.06	204	4.31

Inst	Exact		MLP		WLP		TLP	
	Obj	Time (s)	LB	Time (s)	LB	Time (s)	LB	Time (s)
09	755	27.42	0	0.01	0	0.06	318	5.72
10	785	46.31	0	0.01	0	0.06	319	7.57
11	752	105.87	0	0	0	0.06	238	13.95
12	615	106.91	0	0.01	0	0.07	150	10.58
13	340	13.07	0	0.01	0	0.06	31	1.89
14	568	1180.94	0	0.01	0	0.05	202	6.74
15	892	21.85	0	0	0	0.06	241	3.50
16	818	19.26	0	0.01	0	0.07	439	2.99
17	456	88.42	0	0	0	0.06	45	2.32
18	944	110.73	0	0.01	0	0.06	412	5.81
19	1197	29.51	0	0	0	0.05	563	6.19
				14 jobs				
00	-	7200	0	0	0	0.14	130	20.07
01	823	93.48	0	0.01	0	0.10	301	11.48
02	952	1419.33	0	0.01	0	0.13	400	7.34
03	1469	44.98	0	0.01	0	0.10	594	4.96
04	-	7200	0	0	0	0.09	281	16.84
05	716	97.52	0	0	0	0.08	224	20.15
06	926	112.34	0	0.01	0	0.14	329	19.72
07	979	2950.22	0	0.01	0	0.10	220	14.84
08	487	1235.80	0	0.01	0	0.09	142	18.82
09	675	34.83	0	0.01	0	0.18	214	6.95
10	819	152.48	0	0	0	0.10	197	16.89
11	784	942.29	0	0.01	0	0.10	143	36.40
12	1223	137.13	0	0	0	0.10	224	15.43
13	-	7200	0	0.02	0	0.11	107	8.24
14	1060	5286.01	0	0.01	0	0.13	337	15.20
15	-	7200	0	0	0	0.11	291	22.73
16	1133	249.69	0	0.01	0	0.11	403	26.79
17	1171	1597.59	0	0	0	0.11	330	9.98
18	-	7200	0	0.02	0	0.11	291	20.70
19	907	102.79	0	0	0	0.09	127	17.29
				16 jobs				
00	-	7200	0	0.03	0	0.21	332	19.06
01	910	280.62	0	0.01	0	0.20	96	11.65
02	-	7200	0	0.01	0	0.25	204	53.24
03	1489	214.36	0	0	0	0.34	506	32.02
04	1265	4097.93	0	0.02	0	0.15	343	21.78
05	-	7200	0	0	0	0.24	264	26.98
06	1256	340.42	0	0.03	0	0.18	238	22.12
07	1262	145.61	0	0.02	0	0.20	387	9.11
08	-	7200	0	0	0	0.31	394	24.55
09	-	7200	0	0.01	0	0.28	250	113.22
10	978	321.73	0	0.02	0	0.22	225	20.14
11	-	7200	0	0.01	0	0.43	322	50.26
12	1532	279.35	0	0.01	0	0.31	463	18.94
13	792	54.33	0	0.02	0	0.28	139	9.21
14	1121	9.21	0	0	0	0.31	230	18.99
15	-	7200	0	0.01	0	0.19	424	50.82
16	-	7200	0	0.02	0	0.33	357	75.43
17	1383	693.88	0	0.02	0	0.23	301	25.57
18	-	7200	0	0.03	0	0.27	205	54.17
19	1557	179.32	0	0.02	0	0.38	320	42.08

Table C.23: Lower bounds for problems minimizing the total weighted tardiness on instances with six machines and loose due dates. The bold values are optimal. In the Time-Indexed Lagrangean Dual with Precedence (TILDP) and Assignment (TILDA) constraints dualized columns, if the values are suboptimal, the subgradient method stopped because the π variable was too small. '-' values mean that it was not possible to finish processing before 2 hours. On the TILDP and TILDA columns, values in italics are the last lower bounds found before the timeout of 2 hours. The values in the 'TILP' column are obtained with the linear programming relaxation of the time-indexed formulation proposed by Ravetti (2007).

Inst	TILP		TILDP			TILDA		
	LB	**Time (s)**	**LB**	**SM iterations**	**Time (s)**	**LB**	**SM iterations**	**Time (s)**
				6 jobs				
00	9	0.26	**9**	1	0	**9**	57	63.84
01	40	0	40	25	0.02	**47**	5	0.06
02	28	0.78	**28**	1	0.01	-	-	7200
03	159	0.04	155	132	0.15	**199**	8	0.32
04	21	0.53	**21**	1	0	**21**	51	117.99
05	0	0.06	**0**	-	0	**0**	-	0
06	113	0.01	**113**	1	0	**113**	5	0.13
07	56	0.29	56	164	1.22	-	-	7200
08	0	0.92	**0**	-	0.01	**0**	-	0.05
09	10	0.04	10	2	0.10	**14**	16	5.25
10	41	0.03	**41**	1	0	**41**	3	0.17
11	0	0.15	**0**	-	0	**0**	-	0.02
12	120	0.47	**120**	1	0	**120**	58	125.42
13	0	0.11	0	-	0.54	**7**	17	9.72
14	0	1.63	**0**	-	0.01	**0**	-	0.13
15	51	0.39	**51**	1	0	**51**	85	2349.18
16	20	0.08	**20**	1	0	-	-	7200
17	0	0.20	**0**	-	0.01	**0**	-	0.03
18	21	0.46	**21**	1	0.01	*20*	7	7200
19	0	0.58	**0**	-	0.01	**0**	-	0.05
				8 jobs				
00	25	0.39	**25**	1	0	**25**	57	78.60
01	0	4.25	**0**	-	0.01	**0**	-	0.24
02	0	3.87	**0**	-	0.01	**0**	-	0.24
03	128	1.17	116	208	4.58	**190**	93	4161.37
04	12	10.33	**12**	1	0.03	-	-	7200
05	0	5.41	**0**	-	0.01	**0**	-	0.33
06	0	0.23	0	94	1.20	**72**	7	13.48
07	68	1.86	67	5	6.38	*91*	6	7200
08	0	0.35	**0**	-	0	**0**	-	0.04
09	92	10.57	**92**	1	0.03	-	-	7200
10	10	6.65	**10**	1	0.01	-	-	7200
11	70	1.32	**70**	1	0.01	**70**	4	476.52
12	81	2.37	81	1	6.82	-	-	7200
13	32	6.14	**32**	1	0.02	-	-	7200
14	0	4.09	**0**	-	0	**0**	-	0.30
15	56	1.61	**56**	1	0.02	-	-	7200
16	126	1.29	**126**	1	0.01	-	-	7200
17	64	0.49	**64**	1	0	*62*	13	7200
18	0	0.59	**0**	-	0	**0**	-	0.12
19	22	1.03	22	341	6.18	**74**	6	3748.99
				10 jobs				
00	30	8.59	**30**	1	0.03	-	-	7200
01	36	2.54	**36**	1	0.02	-	-	7200
02	241	3.62	240	5	20.13	-	-	7200
03	10	15.39	**10**	1	0.10	-	-	7200
04	42	11.74	**42**	1	0.05	-	-	0.05
05	29	3.70	23	83	17.45	-	-	7200
06	67	16.94	**67**	1	0.06	-	-	7200
07	0	1.45	**0**	-	0.01	**0**	-	0.21
08	3	6.72	2	7	32.69	-	-	7200
09	8	20.72	8	303	110.21	-	-	7200
10	0	11.46	1	304	80.79	-	-	7200
11	81	4.99	81	11	28.10	-	-	7200
12	183	8.79	181	9	47.03	-	-	7200
13	54	13	**54**	1	0.05	-	-	7200.01
14	0	1.41	**0**	-	0.01	**0**	-	0.19
15	3	23.23	3	13	108.25	-	-	7200
16	0	9.59	**0**	-	0.01	**0**	-	0.71

Inst	TILP		TILDP			TILDA		
	LB	**Time (s)**	**LB**	**SM iterations**	**Time (s)**	**LB**	**SM iterations**	**Time (s)**
17	80	3.13	62	183	17.95	-	-	7200
18	0	14.83	**0**	-	0.02	**0**	-	1.23
19	0	16.06	**0**	-	0.01	**0**	-	1.20
				12 jobs				
00	0	19.91	**0**	-	0	**0**	-	1.71
01	0	7.76	**0**	-	0.02	**0**	-	0.75
02	11	21.20	**11**	1	0.08	-	-	7200
03	171	20.56	164	9	90.28	-	-	7200
04	96	40.67	**96**	1	0.13	-	-	7200
05	18	41.55	12	5	169.79	-	-	7200
06	61	31.32	**61**	1	0.12	-	-	7200
07	0	23.06	**0**	-	0.01	**0**	-	1.96
08	34	14.68	25	15	62.20	-	-	7200
09	131	18.23	123	16	81.95	-	-	7200
10	21	22.03	21	56	110.74	-	-	7200
11	0	31.88	**0**	-	0	**0**	-	2.63
12	0	30.13	**0**	-	0.01	**0**	-	2.52
13	8	16.92	**8**	1	0.08	-	-	7200
14	102	25.80	**102**	1	0.09	-	-	7200
15	26	16.23	**26**	1	0.07	-	-	7200
16	156	21.40	**156**	1	0.07	-	-	7200
17	0	31.86	**0**	-	0.01	**0**	-	2.32
18	120	20.70	**120**	1	0.07	-	-	7200
19	249	23.55	**249**	1	0.10	-	-	7200
				14 jobs				
00	0	91.79	**0**	-	0.02	**0**	-	8.39
01	54	27.90	**54**	1	0.10	-	-	7200
02	156	43.93	**156**	1	0.15	-	-	7200
03	165	46.17	**165**	1	0.16	-	-	7200
04	7	76.71	**7**	1	0.32	-	-	7200
05	0	66.68	**0**	-	0.01	**0**	-	5.42
06	68	53.02	65	13	210.17	-	-	7200
07	0	41.57	**0**	-	0.01	**0**	-	3.53
08	0	76.46	**0**	-	0.03	**0**	-	6.19
09	0	60.33	**0**	-	0.01	**0**	-	5.01
10	68	88.02	**68**	1	0.27	-	-	7200
11	74	89.80	**74**	1	0.28	-	-	7200
12	0	21.05	**0**	-	0.01	**0**	-	1.80
13	54	53.49	**54**	1	0.18	-	-	7200
14	32	81.88	32	154	223.15	-	-	7200
15	0	60.71	**0**	-	0.01	**0**	-	4.93
16	0	61.02	**0**	-	0.01	**0**	-	4.56
17	0	42.28	**0**	-	0.01	**0**	-	3.36
18	60	103.88	**60**	1	0.32	-	-	7200
19	0	45.18	**0**	-	0.01	**0**	-	3.57
				16 jobs				
00	21	50	21	13	240.89	-	-	7200
01	0	78.98	**0**	-	0.01	**0**	-	5.54
02	0	110.56	**0**	-	0.02	**0**	-	6.60
03	270	78.63	270	1	377.88	-	-	7200
04	0	136.02	**0**	-	0.01	**0**	-	7.88
05	62	131.82	**62**	1	0.39	-	-	7200
06	0	72.94	**0**	-	0.02	**0**	-	5.34
07	14	137.57	15	203	495.20	-	-	7200
08	88	118.65	**88**	1	0.32	-	-	7200
09	-	7200	3	51	912.07	-	-	7200
10	0	127.41	**0**	-	0.03	**0**	-	7.80
11	158	170.21	**158**	1	0.47	-	-	7200.01
12	198	54.54	198	1	273.98	-	-	7200
13	18	32.85	**18**	1	0.11	-	-	7200
14	0	45.25	**0**	-	0.01	**0**	-	2.96
15	29	71.61	26	51	359.85	-	-	7200

Inst	TILP		TILDP			TILDA		
	LB	Time (s)	LB	SM iterations	Time (s)	LB	SM iterations	Time (s)
16	90	95.52	**90**	1	0.31	-	-	7200
17	0	44.76	**0**	-	0.01	**0**	-	3.37
18	14	89.44	**14**	1	0.34	-	-	7200
19	6	107.14	**6**	1	0.33	-	-	7200.01

Table C.24: Lower bounds for problems minimizing the total weighted tardiness on instances with six machines and tight due dates. The bold values are optimal. In the Time-Indexed Lagrangean Dual with Precedence (TILDP) and Assignment (TILDA) constraints dualized columns, if the values are suboptimal, the subgradient method stopped because the π variable was too small. '-' values mean that it was not possible to finish processing before 2 hours. On the TILDP and TILDA columns, values in italics are the last lower bounds found before the timeout of 2 hours. The values in the 'TILP' column are obtained with the linear programming relaxation of the time-indexed formulation proposed by Ravetti (2007).

Inst	TILP		TILDP			TILDA		
	LB	Time (s)	LB	SM iterations	Time (s)	LB	SM iterations	Time (s)
				6 jobs				
00	72	0.08	72	280	0.26	**136**	6	0.51
01	113	0.01	113	126	0.05	**131**	4	0.04
02	128	0.05	126	234	0.37	**191**	4	0.21
03	222	0.05	218	131	0.23	-	-	7200
04	85	0.24	78	134	1.25	**138**	4	1.42
05	77	0	77	103	0.04	**103**	4	0.05
06	300	0.01	**300**	1	0	**300**	4	0.07
07	325	0.19	313	236	1.61	-	-	7200
08	133	0.15	131	45	0.83	185	5	376.23
09	110	0.04	99	226	0.21	**176**	4	0.15
10	181	0.02	172	346	0.13	**238**	5	0.13
11	135	0.02	135	192	0.15	**151**	4	0.15
12	329	0.07	328	102	0.47	-		7200
13	111	0.07	107	310	0.48	**200**	4	0.58
14	220	0.09	219	5	0.80	-	-	7200
15	186	0.06	186	58	0.38	-	-	7200
16	128	0.09	122	94	0.56	**163**	5	1.47
17	179	0.11	179	128	0.64	-	-	7200
18	179	0.06	179	122	0.44	**251**	4	0.75
19	115	0.05	115	127	0.28	**179**	6	0.82
				8 jobs				
00	220	0.35	220	338	1.48	**359**	4	5.51
01	89	1.27	86	153	6.44	250	5	6429.14
02	49	0.12	49	52	0.68	**148**	9	4.16
03	237	0.26	225	156	1.99	-	-	7200
04	386	0.43	383	190	2.94	-	-	7200
05	55	0.98	55	214	6.55	-	-	7200
06	297	0.50	297	612	4.26	**538**	6	15.28
07	466	0.66	448	281	4.41	**689**	6	21.98
08	205	0.33	202	375	3.16	**317**	4	10.08
09	446	1.82	442	251	9.78	**573**	7	55.79
10	232	0.43	232	372	3.01	**400**	8	2727.25
11	259	0.37	254	521	3.73	**382**	5	4.58
12	444	0.55	429	419	4.61	**699**	6	11.46
13	90	0.67	90	10	3.95	**238**	7	694.58
14	277	0.42	262	192	3.27	**361**	7	26.77
15	244	0.77	237	197	5.94	431	247	4104.88
16	307	0.41	305	296	3.09	527	7	1673.25
17	242	0.17	237	299	1.21	**453**	8	3.02
18	219	0.26	220	190	1.51	**388**	5	2.79

Inst	TILP		TILDP			TILDA		
	LB	**Time (s)**	**LB**	**SM iterations**	**Time (s)**	**LB**	**SM iterations**	**Time (s)**
19	390	1.09	378	138	6.63	-	-	-
				10 jobs				
00	154	2.96	146	477	19.68	*228*	3	7200
01	136	2.13	130	230	12.17	*280*	5	7200
02	511	1.65	506	203	10.68	-	-	-
03	346	1.94	338	409	12.48	**611**	8	180.83
04	330	0.91	324	255	7.28	**669**	8	1952.47
05	184	1.91	176	431	11.29	**662**	4	69.99
06	382	3.44	368	264	20	**745**	8	2250.27
07	296	1.70	285	499	12.31	-	6	671
08	189	1.49	177	317	8.60	-	-	-
09	192	0.24	184	266	2.22	414	5	617.02
10	358	0.89	351	1113	11.38	**691**	6	45.69
11	178	0.71	173	138	3.20	**481**	4	77.17
12	402	0.95	399	259	6.51	**819**	5	46.79
13	213	2.62	213	385	15.25	-	-	7200
14	125	1.70	125	329	9.46	**494**	5	765.08
15	284	1.31	275	230	8.69	588	7	6340.94
16	222	1.71	222	107	10.26	-		7200
17	319	1.31	318	366	7.99	**675**	60	422.97
18	288	7.56	287	170	32.45	*122*	2	7200
19	148	1.92	146	151	8.92	*299*	3	7200
				12 jobs				
00	237	2.10	234	175	13.99	-	-	7200
01	137	2.84	137	558	16.21	*558*	5	7200
02	142	5.82	142	340	26.03	*393*	3	7200
03	286	4.07	282	319	24.47	*874*	6	7200
04	285	6.99	275	353	36.32	*620*	3	7200
05	262	5.45	260	408	34.17	*428*	2	7200
06	166	16.10	161	213	92.28	*397*	2	7200
07	368	2.63	356	456	17.01	-	-	7200
08	204	4.32	189	351	30.30	*324*	2	7200
09	318	5.72	308	313	30.99	*687*	6	7200
10	319	7.57	294	631	74.78	**785**	6	887.32
11	238	14.00	232	492	71.38	-	-	7200
12	150	10.61	151	121	47.87	-	-	7200
13	31	1.88	25	167	11.78	*194*	2	7200
14	202	6.76	200	77	34.86	*261*	2	7200
15	241	3.48	240	400	21.67	-	-	7200
16	439	2.98	433	169	17.00	-	-	7200
17	45	2.31	46	126	14.32	-	-	7200
18	412	5.80	412	411	29.02	-	-	7200
19	563	6.15	561	312	37.55	*571*	2	7200
				14 jobs				
00	130	20.07	130	352	136.46	-	-	7200
01	301	11.47	288	230	50.67	-	-	7200
02	400	7.34	395	248	41.08	-	-	7200
03	594	4.95	594	476	47.87	*843*	3	7200
04	281	16.90	281	447	90.17	-	-	7200
05	224	20.15	221	385	93.62	-	-	7200
06	329	19.73	327	368	119.18	*593*	2	7200
07	220	14.83	220	174	57.62	-	-	7200.01
08	142	18.80	131	247	81.86	-	-	7200
09	214	6.93	191	537	46.79	-	-	7200
10	197	16.95	192	179	76.35	-	-	7200
11	143	36.38	143	152	91.84	-	-	7200
12	224	15.42	204	654	96.73	-	-	7200
13	107	8.24	103	176	45.66	-	-	7200
14	337	15.20	319	366	67.46	-	-	7200
15	291	22.72	290	193	89.82	-	-	7200
16	403	26.81	395	391	134.73	-	-	7200
17	330	9.97	326	231	42.71	-	-	7200

Inst	TILP		TILDP			TILDA		
	LB	Time (s)	LB	SM iterations	Time (s)	LB	SM iterations	Time (s)
18	291	20.70	290	76	68.72	-	-	7200
19	127	17.27	127	473	91.39	-	-	7200
				16 jobs				
00	332	19.08	329	338	97.85	-	-	7200
01	96	11.66	89	317	63.23	-	-	7200
02	204	53.23	204	142	159.95	-	-	7200
03	506	31.98	499	205	126.95	-	-	7200
04	343	21.81	338	417	106.85	-	-	7200
05	264	27.02	255	396	103.92	-	-	7200
06	238	22.19	238	441	83.18	-	-	7200
07	387	9.11	370	742	67.34	-	-	7200
08	394	24.70	389	139	109.94	-	-	7200
09	250	113.43	238	482	262.91	-	-	7200
10	225	20.03	225	183	56.28	-	-	7200
11	322	50.37	318	358	262.52	-	-	7200
12	463	18.98	456	682	99.61	-	-	7200
13	139	9.20	139	269	44.41	-	-	7200
14	230	19.05	220	448	119.23	-	-	7200
15	424	50.73	418	458	235.03	-	-	7200
16	357	75.40	345	160	261.02	-	-	7200
17	301	25.65	292	599	144.74	-	-	7200
18	205	54.14	195	400	253.58	-	-	7200
19	320	42.08	305	656	161.66	-	-	7200

Table C.25: Lower bounds for problems minimizing the total weighted tardiness on instances with loose due dates obtained with the Plain Subgradient Method (PSM) and Non-Delayed Relax-and-Cut (NDRC). The values marked in bold are optimal. If the values are suboptimal, the subgradient method stopped because of the timeout of 2 hours or because the π variable was too small. The former case is indicated by the values obtained before timeout, in italics.

Inst	PSM		NDRC	
	LB	Time (s)	LB	Time (s)
		10 jobs		
00	**30**	0.02	**30**	0.02
01	**36**	0.01	**36**	0.01
02	240	13.70	240	15.69
03	**10**	0.04	**10**	0.03
04	**42**	0.02	**42**	0.03
05	23	11.94	29	16.30
06	**67**	0.03	**67**	0.04
07	**0**	0.01	**0**	0.01
08	2	22.34	3	22.60
09	8	74.36	8	74.55
10	0	54.60	0	54.87
11	81	19.37	81	19.17
12	181	31.69	183	32.41
13	**54**	0.03	**54**	0.03
14	**0**	0.01	**0**	0
15	3	73.01	**3**	11.92
16	**0**	0.01	**0**	0.01
17	62	12.16	80	13.53
18	**0**	0.02	**0**	0.01
19	**0**	0.01	**0**	0.01
		20 jobs		
00	**0**	0.03	**0**	0.03
01	**243**	0.31	**243**	0.31
02	39	826.70	42	856.73
03	1	1057.66	1	1049.41
04	**0**	0.03	**0**	0.03
05	198	1012	222	1076.16

Inst	PSM		NDRC	
	LB	**Time (s)**	**LB**	**Time (s)**
06	112	520.45	121	551.93
07	*50*	7200	*49*	7200
08	**144**	0.30	**144**	0.30
09	0	736.45	0	743.05
10	**0**	0.02	**0**	0.03
11	**0**	0.04	**0**	0.04
12	34	901.71	**36**	137.69
13	**16**	0.47	**16**	0.48
14	**46**	0.29	**46**	0.29
15	146	0.19	146	0.19
16	**0**	0.02	**0**	0.03
17	**109**	0.33	**109**	0.32
18	125	953.84	130	967.21
19	**36**	0.45	**36**	0.45
		40 jobs		
00	*3*	7200	*3*	7200
01	**128**	3.90	**128**	3.90
02	-	7200	-	7200
03	**0**	0.25	**0**	0.25
04	*85*	7200	*85*	7200
05	**0**	0.19	**0**	0.18
06	**32**	4.26	**32**	4.26
07	**105**	5.05	**105**	5.05
08	*171*	7200	*178*	7200
09	**0**	0.22	**0**	0.19
10	**60**	4.41	**60**	4.40
11	**109**	5.64	**109**	5.65
12	**90**	3.88	**90**	3.87
13	**114**	4.39	**114**	4.40
14	**0**	0.19	**0**	0.20
15	**0**	0.20	**0**	0.19
16	**221**	4.23	**221**	4.24
17	**37**	4.25	**37**	4.24
18	**0**	0.18	**0**	0.17
19	**0**	0.25	**0**	0.25
		80 jobs		
00	-	7200	-	7200
01	**0**	1.23	**0**	1.20
02	**24**	42.27	**24**	42.25
03	-	7200	-	7200
04	**0**	1.31	**0**	1.31
05	*45*	7200	*45*	7200
06	**0**	1.22	**0**	1.21
07	**0**	1.30	**0**	1.30
08	**94**	41.21	**94**	41.08
09	**87**	40.46	**87**	40.39
10	*57*	7200	*57*	7200
11	**30**	43.80	**30**	43.84
12	**0**	1.17	**0**	1.15
13	-	7200	-	7200
14	**15**	43.64	**15**	43.53
15	**33**	40.39	**33**	40.42
16	*102*	7200	*102*	7200
17	**0**	1.17	**0**	1.17
18	*68*	7200	*68*	7200
19	*239*	7200	*239*	7200

Table C.26: Lower bounds for problems minimizing the total weighted tardiness on instances with tight due dates obtained with the Plain Subgradient Method (PSM) and Non-Delayed Relax-and-Cut (NDRC). The values marked in bold are optimal. If the values are suboptimal, the subgradient method stopped because of the timeout of 2 hours or because the π variable was too small. The former case is indicated by the values obtained before timeout, in italics.

Inst	PSM		NDRC	
	LB	**Time (s)**	**LB**	**Time (s)**
		10 jobs		
00	146	13.44	154	10.90
01	130	6.98	136	7.19
02	506	7.25	511	7.31
03	338	8.66	352	6.47
04	324	5.01	330	5.03
05	176	7.62	184	6.65
06	368	13.50	382	12.09
07	285	8.44	296	7.47
08	177	6.01	189	5.61
09	184	1.57	192	1.29
10	351	7.87	358	4.04
11	173	2.30	178	2.27
12	399	4.48	402	4.33
13	213	10.41	213	7.99
14	125	6.52	125	5.35
15	275	6.06	284	5.80
16	222	7.21	222	8.42
17	318	5.54	319	4.67
18	287	22.14	288	22.92
19	146	6.16	148	6.06
		20 jobs		
00	137	230.97	149	180.05
01	438	234.12	447	214.66
02	255	361.79	271	291.17
03	367	305.92	383	266.58
04	58	220.37	81	231.03
05	-	7200	-	7200
06	499	298.72	525	223.06
07	349	324.15	357	310.32
08	312	138.25	324	110.83
09	258	198.24	282	193.08
10	193	236.05	214	234.12
11	314	314	308	234.70
12	269	161.79	284	141.93
13	156	176.87	166	160.34
14	106	137.69	108	126.47
15	230	216.91	246	197.94
16	330	235.28	340	189.21
17	231	207.65	248	186.19
18	364	446.65	392	339.11
19	489	306.36	522	243.62
		40 jobs		
00	361	5713.45	387	4180.52
01	252	2910.42	270	3170.19
02	*207*	7200	*213*	7200
03	460	4310.81	498	3650.69
04	542	3607.32	566	3788.66
05	-	7200	-	7200
06	111	3330.82	127	2956.83
07	399	3573.12	402	3833.31
08	430	4003.93	465	4399.97
09	-	7200	-	7200
10	403	3442.31	417	3437.95
11	610	5552.79	635	4725.46
12	203	4636.34	222	4353.87
13	410	3465.22	430	3225.01
14	396	4657.92	425	4080.32
15	232	3459.54	247	3161.13
16	552	4867.65	577	3726.26

Inst	PSM		NDRC	
	LB	**Time (s)**	**LB**	**Time (s)**
17	303	4756.11	312	4427.20
18	167	5501.03	178	5201.15
19	418	5953.50	461	3898.40
		80 jobs		
00	*280*	7200	*278*	7200
01	*166*	7200	*161*	7200
02	*328*	7200.00	*327*	7200.00
03	*324*	7200.00	*322*	7200.00
04	*110*	7200	*103*	7200
05	*264*	7200.00	*260*	7200.00
06	-	7200	-	7200
07	-	7200	-	7200
08	*377*	7200.00	*376*	7200.00
09	*261*	7200.00	*260*	7200.00
10	*487*	7200	*491*	7200
11	*611*	7200	*651*	7200
12	-	7200	-	7200
13	-	7200	-	7200
14	*401*	7200	*422*	7200
15	*415*	7200.00	*412*	7200.00
16	*307*	7200	*323*	7200
17	*173*	7200.00	*172*	7200.00
18	*259*	7200	*277*	7200
19	*739*	7200.00	*737*	7200.00

Table C.27: Upper bounds for problems minimizing the total weighted tardiness on instances with six machines and loose due dates obtained with the Lagrangean Heuristic with (LHLS) and without the local search (LH). The values in italics were obtained by the lagrangean heuristic, and were better than the initial solution, obtained by VNS. The OPT values mean that the lagrangean heuristic did not run because the initial solution, obtained by VNS, was proven to be optimal by the first lower bound obtained. The NLB values mean that the lagrangean heuristic did not run because it was not possible to obtain a nonzero lower bound. The TMO values mean that no lower bound could be obtained before the timeout of one hour, consequently, the lagrangean heuristic could not run.

			LH			LHLS			VNS	
Inst	**LowerBound**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**
					10 jobs					
00	30	OPT	0	0.02	OPT	0	0.03	30	0.00	0.01
01	36	OPT	0	0.03	OPT	0	0.03	36	0.00	0.02
02	241	320	0.27	12.89	320	0.27	12.88	313	0.25	0.01
03	10	OPT	0	0.03	OPT	0	0.05	10	0.00	0.02
04	42	OPT	0	0.04	OPT	0	0.04	42	0.00	0.01
05	29	134	2.83	12.14	*95*	1.71	11.17	116	2.31	0.02
06	67	OPT	0	0.04	OPT	0	0.03	67	0.00	0.02
07	0	OPT	0	0.02	OPT	0	0.02	0	0	0.02
08	3	*33*	10.00	19.05	*33*	10.00	19.06	54	17.00	0.02
09	8	100	11.50	62.65	56	6.00	62.65	56	6.00	0.03
10	0	18	100	46.34	18	100	45.62	18	100	0.02
11	81	211	1.60	15.77	125	0.54	15.66	95	0.17	0.01
12	183	312	0.69	28.94	312	0.69	28.95	220	0.19	0.00
13	54	OPT	0	0.03	OPT	0	0.03	54	0.00	0.02
14	0	OPT	0	0.01	OPT	0	0.01	0	0	0.01
15	4	16	3.00	39.49	4	0.00	39.62	4	0.00	0.01
16	0	OPT	0	0.02	OPT	0	0.01	0	0	0.01
17	79	*165*	1.09	11.54	*165*	1.09	11.57	264	2.34	0.02
18	0	OPT	0	0.01	OPT	0	0.01	0	0	0.02
19	0	OPT	0	0.02	OPT	0	0.02	0	0	0.01
					20 jobs					
00	0	OPT	0	0.03	OPT	0	0.03	0	0	0.02

			LH			LHLS			VNS	
Inst	**LowerBound**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**
01	243	OPT	0	0.25	OPT	0	0.26	243	0.00	0.01
02	42	49	0.17	705.32	49	0.17	699.57	49	0.17	0.01
03	0	NLB	-	873.66	NLB	-	872.77	56	-	0.02
04	0	OPT	0	0.03	OPT	0	0.03	0	0	0.02
05	214	752	2.51	904.05	449	1.10	905.32	414	0.93	0.01
06	119	684	4.75	585.28	298	1.50	584.91	200	0.68	0.02
07	52	259	3.98	1124.84	98	0.88	1125.97	98	0.88	0.01
08	144	OPT	0	0.26	OPT	0	0.25	144	0.00	0.02
09	0	NLB	-	625.88	NLB	-	626.62	11	-	0.01
10	0	OPT	0	0.04	OPT	0	0.03	0	0	0.02
11	0	OPT	0	0.03	OPT	0	0.03	0	0	0.01
12	35	217	5.20	774.75	121	2.46	776.27	69	0.97	0.02
13	16	OPT	0	0.39	OPT	0	0.38	16	0.00	0.02
14	46	OPT	0	0.24	OPT	0	0.25	46	0.00	0.01
15	146	OPT	0	0.22	OPT	0	0.22	146	0.00	0.02
16	0	OPT	0	0.03	OPT	0	0.02	0	0	0.02
17	109	OPT	0	0.27	OPT	0	0.27	109	0.00	0.01
18	130	180	0.38	870.77	148	0.14	871.44	144	0.11	0.02
19	36	OPT	0	0.36	OPT	0	0.37	36	0.00	0.02
					40 jobs					
00	2	44	21.00	3600.00	44	21.00	3600.00	4	1.00	0.02
01	128	OPT	0	3.20	OPT	0	3.21	128	0.00	0.02
02	21	972	45.29	3600.00	116	4.52	3600.00	51	1.43	0.02
03	0	OPT	0	0.12	OPT	0	0.13	0	0	0.03
04	85	300	2.53	3600.00	139	0.64	3600.00	96	0.13	0.02
05	0	OPT	0	0.16	OPT	0	0.18	0	0	0.02
06	32	OPT	0	3.56	OPT	0	3.53	32	0.00	0.03
07	105	OPT	0	4.21	OPT	0	4.22	105	0.00	0.02
08	170	576	2.39	3600.00	450	1.65	3600.00	250	0.47	0.03
09	-	OPT	0	0.03	OPT	0	0.03	8	-	0.04
10	60	OPT	0	3.58	OPT	0	3.60	60	0.00	0.02
11	109	OPT	0	4.67	OPT	0	4.63	109	0.00	0.02
12	90	OPT	0	3.15	OPT	0	3.15	90	0.00	0.03
13	114	OPT	0	3.59	OPT	0	3.60	114	0.00	0.02
14	0	OPT	0	0.13	OPT	0	0.14	0	0	0.04
15	-	OPT	0	0.03	OPT	0	0.02	0	0	0.02
16	221	OPT	0	3.46	OPT	0	3.45	221	0.00	0.03
17	37	OPT	0	3.49	OPT	0	3.48	37	0.00	0.02
18	0	OPT	0	0.11	OPT	0	0.12	0	0	0.02
19	0	OPT	0	0.13	OPT	0	0.13	0	0	0.02
					80 jobs					
00	-	TMO	-	3600.00	TMO	-	3600.00	15	-	0.04
01	0	OPT	0	0.96	OPT	0	1.00	0	0	0.08
02	24	OPT	0	34.68	OPT	0	34.60	24	0.00	0.05
03	14	2519	178.93	3600.00	62	3.43	3600.00	55	2.93	0.08
04	0	OPT	0	1.49	OPT	0	1.40	0	0	0.06
05	45	2909	63.64	3600.00	153	2.40	3600.00	87	0.93	0.06
06	0	OPT	0	1.25	OPT	0	1.26	0	0	0.06
07	0	OPT	0	1.29	OPT	0	1.28	0	0	0.05
08	94	OPT	0	34.23	OPT	0	34.20	94	0.00	0.06
09	87	OPT	0	33.79	OPT	0	33.74	87	0.00	0.05
10	57	2675	45.93	3600.00	118	1.07	3600.00	106	0.86	0.08
11	30	OPT	0	36.05	OPT	0	35.59	30	0.00	0.06
12	0	OPT	0	0.83	OPT	0	0.84	0	0	0.07
13	324	2038	5.29	3600.00	341	0.05	3600.00	333	0.03	0.08
14	15	OPT	0	35.83	OPT	0	35.79	15	0.00	0.06
15	33	OPT	0	33.08	OPT	0	33.07	33	0.00	0.09
16	102	1935	17.97	3600.00	200	0.96	3600.00	172	0.69	0.06
17	0	OPT	0	0.84	OPT	0	0.85	0	0	0.05
18	68	1008	13.82	3600.00	88	0.29	3600.00	88	0.29	0.07
19	239	4482	17.75	3600.00	521	1.18	3600.00	301	0.26	0.07

Table C.28: Upper bounds for problems minimizing the total weighted tardiness on instances with six machines and tight due dates obtained with the Lagrangean Heuristic with (LHLS) and without the local search (LH). The values in italics were obtained by the lagrangean heuristic, and were better than the initial solution, obtained by VNS.

			LH			LHLS			VNS	
Inst	**LowerBound**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**
					10 jobs					
00	154	*455*	1.95	7.82	*455*	1.95	7.73	457	1.97	0.01
01	136	423	2.11	6.11	364	1.68	6.14	364	1.68	0.02
02	511	898	0.76	5.51	898	0.76	5.51	854	0.67	0.02
03	346	661	0.88	5.42	611	0.74	5.38	611	0.74	0.01
04	330	669	1.02	4.22	669	1.02	4.23	669	1.02	0.02
05	184	729	2.86	5.31	701	2.71	5.47	682	2.61	0.02
06	382	921	1.41	10.49	*745*	0.95	9.97	849	1.22	0.01
07	296	788	1.63	5.56	759	1.53	5.46	712	1.37	0.02
08	189	606	2.19	3.87	606	2.19	3.88	572	2.01	0.01
09	192	535	1.74	1.23	445	1.28	1.23	424	1.17	0.02
10	358	783	1.18	3.36	691	0.92	3.38	691	0.92	0.02
11	178	568	2.04	2.02	481	1.57	2.02	481	1.57	0.02
12	402	838	1.04	3.28	830	1.02	3.29	830	1.02	0.02
13	213	534	1.41	6.49	491	1.21	6.24	491	1.21	0.01
14	125	*540*	3.12	4.27	*494*	2.77	4.42	632	3.82	0.01
15	284	789	1.78	5.44	622	1.19	5.42	622	1.19	0.01
16	222	489	1.16	6.39	471	1.08	6.40	454	1.01	0.01
17	319	675	1.03	4.01	675	1.03	4.02	675	1.03	0.01
18	288	618	1.12	17.39	583	1.00	17.40	582	1.00	0.02
19	148	569	2.74	5.25	*467*	2.07	5.03	538	2.54	0.01
					20 jobs					
00	138	1690	11.25	214.65	1583	10.47	211.37	1512	9.96	0.02
01	435	2391	4.50	174.38	2108	3.85	174.40	1811	3.16	0.02
02	262	1729	5.60	245.69	*1684*	5.43	188.56	1704	5.50	0.02
03	370	2275	5.15	159.76	1983	4.36	160.02	1904	4.15	0.01
04	66	1829	26.71	190.85	*1662*	24.18	176.22	1801	26.29	0.02
05	520	2382	3.58	234.12	2091	3.02	234.59	2021	2.89	0.01
06	510	2195	3.30	243.70	2158	3.23	243.91	2155	3.23	0.03
07	338	2926	7.66	301.35	2074	5.14	301.92	1878	4.56	0.01
08	309	2789	8.03	82.21	2268	6.34	82.08	2045	5.62	0.01
09	271	1655	5.11	123.68	*1462*	4.39	115.69	1475	4.44	0.01
10	202	2342	10.59	136.80	*1727*	7.55	127.68	1749	7.66	0.01
11	311	2429	6.81	201.44	1919	5.17	201.21	1717	4.52	0.02
12	271	2161	6.97	94.70	1897	6.00	94.67	1566	4.78	0.02
13	152	1575	9.36	114.82	1472	8.68	114.94	1372	8.03	0.01
14	100	1762	16.62	97.62	1613	15.13	97.49	1581	14.81	0.01
15	231	2062	7.93	137.31	1561	5.76	137.47	1463	5.33	0.01
16	326	2533	6.77	132.91	2222	5.82	133.55	2144	5.58	0.01
17	231	2678	10.59	137.61	1946	7.42	137.93	1946	7.42	0.01
18	377	3478	8.23	236.61	*2002*	4.31	229.91	2118	4.62	0.01
19	500	2995	4.99	163.10	2625	4.25	163.61	2176	3.35	0.02
					40 jobs					
00	348	10048	27.87	3028.77	6345	17.23	3024.14	5961	16.13	0.05
01	250	6914	26.66	2317.51	5100	19.40	2303.94	4880	18.52	0.04
02	196	6666	33.01	2486.70	*5106*	25.05	2463.76	5163	25.34	0.05
03	446	11273	24.28	2675.73	8679	18.46	2716.72	8396	17.83	0.06
04	539	8640	15.03	2993.85	6353	10.79	2995.33	6335	10.75	0.04
05	118	7353	61.31	2579.66	4891	40.45	2570.88	4449	36.70	0.04
06	107	8460	78.07	2077.93	5786	53.07	2075.27	5281	48.36	0.04
07	390	10627	26.25	3142.67	*5945*	14.24	3140.68	5952	14.26	0.04
08	422	8835	19.94	2435.80	5437	11.88	2429.94	5368	11.72	0.03
09	285	10126	34.53	2141.47	5906	19.72	2141.54	5569	18.54	0.08
10	399	8435	20.14	2527.99	5740	13.39	2530.67	5596	13.03	0.05
11	600	8497	13.16	3351.66	6986	10.64	3397.03	6797	10.33	0.04
12	194	7118	35.69	2160.53	5775	28.77	2156.51	5523	27.47	0.04
13	404	6686	15.55	2672.54	*5244*	11.98	2663.47	5331	12.20	0.03
14	383	10245	25.75	2521.23	6313	15.48	2512.91	6067	14.84	0.06
15	227	9153	39.32	2322.79	6302	26.76	2320.13	6146	26.07	0.04
16	540	10413	18.28	2998.68	8488	14.72	2991.41	7846	13.53	0.03

			LH			LHLS			VNS	
Inst	**LowerBound**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**	**Sol**	**Gap (%)**	**Time (s)**
17	281	10443	36.16	2785.60	5808	19.67	2786.30	5786	19.59	0.02
18	142	7045	48.61	1905.86	5287	36.23	1906.96	4840	33.08	0.04
19	391	10791	26.60	2567.18	7980	19.41	2569.10	7814	18.98	0.02
					80 jobs					
00	278	30625	109.16	3600.00	18727	66.36	3600.00	18238	64.60	0.48
01	161	28213	174.24	3600.00	17198	105.82	3600.00	16689	102.66	0.36
02	327	35109	106.37	3600.00	20530	61.78	3600.00	19694	59.23	0.33
03	322	34408	105.86	3600.00	21566	65.98	3600.00	19409	59.28	0.29
04	103	34129	330.35	3600.00	21070	203.56	3600.00	20445	197.50	0.35
05	260	28683	109.32	3600.00	18674	70.82	3600.00	17416	65.98	0.30
06	80	29631	369.39	3600.00	19630	244.38	3600.00	19356	240.95	0.21
07	131	30755	233.77	3600.00	22516	170.88	3600.00	21949	166.55	0.20
08	376	37054	97.55	3600.00	23311	61.00	3600.00	22479	58.78	0.41
09	260	26506	100.95	3600.00	20051	76.12	3600.00	19422	73.70	0.25
10	474	32322	67.19	3600.00	19220	39.55	3600.00	18530	38.09	0.25
11	601	33873	55.36	3600.00	21244	34.35	3600.00	19950	32.19	0.24
12	125	29466	234.73	3600.00	15921	126.37	3600.00	15517	123.14	0.18
13	661	26577	39.21	3600.00	17719	25.81	3600.00	17427	25.36	0.18
14	382	35577	92.13	3600.00	21301	54.76	3600.00	20152	51.75	0.31
15	412	36659	87.98	3600.00	21687	51.64	3600.00	20611	49.03	0.67
16	290	34104	116.60	3600.00	25255	86.09	3600.00	24477	83.40	0.19
17	172	32786	189.62	3600.00	23274	134.31	3600.00	22299	128.65	0.73
18	252	32037	126.13	3600.00	18626	72.91	3600.00	17721	69.32	0.44
19	737	36233	48.16	3600.00	22401	29.39	3600.00	22089	28.97	0.41

Referências Bibliográficas

Abdekhodaee, A. H. e Wirth, A. (2002). Scheduling parallel machines with a single server: some solvable cases and heuristics. *Computers & Operations Research*, 29(3):295–315.

Abdekhodaee, A. H.; Wirth, A. e Gan, H. S. (2004). Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers & Operations Research*, 31(11):1867–1889.

Abdekhodaee, A. H.; Wirth, A. e Gan, H.-S. (2006). Scheduling two parallel machines with a single server: the general case. *Computers & Operations Research*, 33(4):994–1009.

Adamopoulos, G. I. e Pappis, C. P. (1998). Scheduling under a common due-date on parallel unrelated machines. *European Journal of Operation Research*, 105:494–501.

Aiex, R. M.; Resende, M. G. C. e Ribeiro, C. C. (2005). tttplots - a perl program to create time-to-target plots. Technical report, AT&T Labs Research, Florham Park, New Jersey, USA. http://www.research.att.com/ mgcr/tttplots/.

Akker, J. M. V. D.; Hoesel, C. P. M. V. e Savelsbergh, M. W. P. (1999). A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85:541–572.

Akker, J. M. V. D.; Hurkens, C. A. J. e Savelsbergh, M. W. P. (2000). Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124.

Allahverdi, A.; Ng, C.; Cheng, T. e Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032.

Anghinolfia, D. e Paolucci, M. (2007). A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, In Press. Corrected Proof, avaiable online November 2007.

Anglani, A.; Grieco, A.; Guerriero, E. e Musmanno, R. (2005). Robust scheduling of parallel machines with sequence-dependent set-up costs. *European Journal of Operational Research*, 161(3):704–720.

Armentano, V. A. e Yamashita, D. S. (2000). Tabu search for scheduling on identical parallel machines to minimize mean tardiness. *Journal of Intelligent Manufacturing*, 11:453–460.

Balakrishnan, N.; Kanet, J. J. e Sridharan, S. V. (1999). Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, 26(2):127–141.

Bank, J. e Werner, F. (2001). Heuristic algorithms for unrelated parallel-machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modeling*, 33:363–383.

Beasley, J. E. (1993). *Modern Heuristic Techniques for Combinatorial Problems*, chapter Lagrangian Relaxation, pp. 243–303. John Wiley & Sons, Inc. ISBN:0-470-22079-1.

Blazewicz, J.; Ecker, K. H.; Pesch, E.; Schmidt, G. e Weglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*. Springer - Verlag, Berlin.

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

Brucker, P.; Dhaenens-Flipo, C.; Knust, S.; Kravchenko, S. A. e Werner, F. (2002). Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, 5(6):429–457.

Brucker, P.; Kovalyov, M. Y.; Shafransky, Y. M. e Werner, F. (1998). Batch scheduling with deadlines on parallel machines. *Annals of Operations Research*, 83:23–40.

Cheng, T. C. E. e Shin, C. C. S. (1990). A state of the art review of parallel-machine scheduling research. *European Journal of Operation Research*, 47:271–292.

Cicirello, V. A. (2003). Weighted tardiness scheduling with sequence-dependent setups: A benchmark library. Technical report, Intelligent Coordination and Logistics Laboratory, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA. http://www.ozone.ri.cmu.edu/benchmarks.html.

Cicirello, V. A. (2006). On the design of an adaptive simulated annealing algorithm. In *First Workshop on Autonomous Search*, Providence, Rhode Island, USA. In conjunction with CP'2007. http://research.microsoft.com/constraint-reasoning/Workshops/Autonomous-CP07/Papers/2.pdf.

Cicirello, V. A. e Smith, S. F. (2005). Enhancing stochastic search performance by value-based randomization of heuristics. *Journal of Heuristics*, 11:5–34.

Crama, Y. e Spieksma, F. C. R. (1993). Scheduling jobs of equal length: complexity, facets and computational results. *Mathematical Programming*, 72(3):1436–4646.

Davis, E. e Jaffe, J. M. (1981). Algorithms for scheduling tasks on unrelated processors. *Journal of the Association for Computing Machinery*, 28:721–736.

De, P. e Morton, T. E. (1980). Scheduling to minimize makespan on unequal parallel processors. *Decision Sciences*, 11:596–603.

de Paula, M. R.; Ravetti, M. G. e Mateus, G. R. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, 18:101–115.

Du, J. e Leung, J. Y. (1990). Minimizing the total tardinesses on one machine is np-hard. *Mathematics of Operations Research*, 15:483–494.

Escudero, L. F.; Guignard, M. e Malik, K. (1994). A lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research*, 50(1):219–237.

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

Festa, P.; Pardalos, P. M.; Resende, M. G. C. e Ribeiro, C. C. (2002). Randomized heuristics for the max-cut problem. *Optimization Methods and Software*, 17(6):1033 – 1058.

Festa, P. e Resende, M. G. C. (2002). Grasp: An annotated bibliography. In Ribeiro, C. C. e Hansen, P., editores, *Essays and surveys in metaheuristics*, pp. 325–367. Kluwer Academic Publishers.

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

Gairing, M.; Monien, B. e Woclaw, A. (2007). A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoretical Computer Science*, 380:87–99.

Garey, M. R. e Johnson, D. S. (1997). *Computers and Intractability: A Guide to the Theory of NP-Completness*. W. H. Freeman.

Gavish, B. (1985). Augmented lagrangean based algorithms for centralized network design. *IEEE Transactions on Communications*, 33(12):1247– 1257.

Gendreau, M.; Laporte, G. e Guimarães, E. M. (2001). A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operation Research*, 133(1):183–189.

Ghirardi, M. e Potts, C. (2005). Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operation Research*, 165:457–467.

Glass, C. A.; Potts, C. e Shade, P. (1994). Unrelated parallel-machine scheduling using local-search. *Mathematical and Computer Modeling*, 20:41–52.

Glass, C. A.; Shafransky, Y. M. e Strusevich, V. A. (2000). Scheduling for parallel dedicated machines with a single server. *Naval Research Logistics*, 47(7):304–328.

Glover, F. e Kochenberger, G. A. (2002). *Handbook of Metaheuristics*. Kluwer Academics.

Glover, F. W. e Laguna, M. (1997). *Tabu Search*. Springer, 1 edição.

Gordon, V.; Proth, J.-M. e Chu, C. (2002). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operation Research*, 139:1–25.

Guignard, M. (1998). Efficient cuts in lagrangean 'relax-and-cut' schemes. *European Journal of Operation Research*, 105(1):216–223.

Guignard, M. (2003). Lagrangean relaxation. *TOP: Sociedad de Estatistica e Investigación Operational*, 11(2):151–228.

Hall, N. G.; Potts, C. N. e Sriskandarajah, C. (2000). Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102(3):223–243.

Hansen, P. e Mladenovic, N. (1999). Variable neighborhood search: Principles and applications. *European Journal of Operational Reasearch*, 130(3):449–467.

Hansen, P. e Mladenovic, N. (2002). Variable neighborhood search. In Pardalos, P. M. e Resende, M. G. C., editores, *Handbook of Applied Optimization*. Oxford University Press, New York.

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

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

Hariri, A. e Potts, C. (1991). Heuristics for scheduling unrelated parallel-machines. *Computers and Operations Research*, 18:323–331.

Heady, R. B. e Zhu, Z. (1998). Minimizing the sum of job earliness and tardiness in a multimachine system. *International Journal of Production Research*, 36(6):1619 – 1632.

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

Held, M. e Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):1436–4646.

Hoon Lee, Y. e Pinedo, M. (1997). Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464–474.

Hoos, H. H. e Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann - Elsevier.

Horowitz, E. e Sahni, S. (1977). Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the Association for Computing Machinery*, 24:280–289.

Hurink, J. L.; Knust, S.; Hurink, J. e Knust, S. (2001). List scheduling in a parallel machine environment with precedence constraints and setup times. *Operations Research Letters*, 29:231–239.

Ibarra, O. H. e Kim, C. E. (1976). Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the Association for Computing Machinery*, 24:280–289.

Jansen, K. e Porkolab, L. (1999). Improved approximation schemes for scheduling unrelated parallel machines. In *ACM Symposium on Theory of Computing*, pp. 408–417.

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

Koulamas, C. P. (1996). Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers & Operations Research*, 23(10):945–956.

Kravchenko, S. A. e Werner, F. (1997). Parallel machine scheduling problems with a single server. *Mathematical and Computer Modelling*, 26(12):1–11.

Kravchenko, S. A. e Werner, F. (2001). A heuristic algorithm for minimizing mean flow time with unit setups. *Information Processing Letters*, 79(6):291–296.

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

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

Lenstra, J. K.; Shmoys, D. B. e Éva Tardos (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271.

Lin, S. e Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Logendran, R.; McDonellb, B. e Smuckera, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, 34(11):3420–3438.

Lucena, A. (2005). Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140(1):375–410.

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

Martello, S.; Soumis, F. e Toth, P. (1997). Exact and approximation algorithms for makespan minimization on unrelated parallel-machines. *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 75:169–188.

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

Mendes, A. S.; Müller, F. M.; França, P. M. e Moscato, P. (2002). Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning and Control*, 13(12):143–154.

Mokotoff, E. e Chrétienne, P. (2002). A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operation Research*, 141:515–525.

Musier, R. F. H. e Evans, L. B. (1989). An approximate method for the production scheduling of industrial batch processes with parallel units. *Computer & Chemical Engineering*, 13:229–238.

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

Park, Y.; Kim, S. e Lee, Y.-H. (2000). Scheduling jobs on parallel machines applying neural network and heuristic rules. *Computers & Industrial Engineering*, 38(1):189–202.

Pfund, M.; Fowler, J. W. e Gupta, J. N. D. (2004). A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling. *Journal of the Chinese Institute of Industrial Engineers*, 21:230–241.

Pfund, M.; Fowler, J. W.; Pfund, M.; Gadkari, A. e Chen, Y. (2008). Scheduling jobs on parallel machines with setup times and ready times. *Computers & Industrial Engineering*, 54(4):764–782.

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

Potts, C. (1985). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*, 10:155–164.

Queyranne, M. e Schulz, A. S. (1994). Polyhedral approaches to machine scheduling. Preprint 408/1994, Dept. of Mathematics, Technical University of Berlin.

Rabadi, G.; Moraga, R. J. e Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17:85–97.

Ravetti, M. G. (2007). *Algorithms for a Scheduling Problem with Parallel Machines and Sequence Dependent Setups*. PhD thesis, Universidade Federal de Minas Gerais.

Ravetti, M. G.; Rocha, P. L. R.; Mateus, G. R. e Pardalos, P. M. (2007). A scheduling problem with unrelated parallel machines and sequence dependent setups. *Int. Journal of Operational Research*, 2(4):380–399.

Resende, M. G. C. e Ribeiro, C. C. (2003). Grasp and path-relinking: Recent advances and applications. In Ibaraki, T. e Yoshitomi, Y., editores, *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*. Tutorial presented at the V Metaheuristics International Conference, Kyoto, Japan, August 2003.

Ribeiro, C. C.; Aloise, D.; Noronha, T. F.; Rocha, C. e Urrutia, S. (2008). A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints. *European Journal of Operational Research*, 191(3):981–992.

Rocha, P. L.; Ravetti, M. G. e Mateus, G. R. (2004). The metaheuristic grasp as an upper bound for a branch and bound algorithm in a scheduling problem with non-related parallel machines and sequence-dependent setup times. In *EU/ME Workshop*, Nottingham, England.

Rocha, P. L.; Ravetti, M. G.; Mateus, G. R. e Pardalos, P. M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, 35(4):1250–1264.

Ross, G. T. e Soland, R. M. (1975). A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8(1):91–103.

Serna, M. e Xhafa, F. (2002). Approximating scheduling unrelated parallel machines in parallel. *Computational Optimization and Applications*, 21:325–338.

Sivrikaya-SerifogIu, F. e Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. *Computers & Operations Research*, 26(8):773–787.

Sousa, J. P. e Wolsey, L. A. (1992). A time indexed formulation of non-preemptive single-machine scheduling problems. *Mathematical Programming*, 52:353–367.

Srivastava, B. (1998). An effective heuristic for minimizing makespan on unrelated parallel-machines. *Journal of the Operational Research Society*, 49:886–894.

Suresh, V. e Chaudhuri, D. (1994). Minimizing maximum tardiness for unrelated parallel machines. *International Journal of Production Economics*, 34:223–229.

Tahar, D. N.; Yalaoui, F.; Chu, C. e Amodeo, L. (2006). A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. *International Journal of Production Economics*, 99(1-2):63–73.

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

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

Weng, M. X.; Lub, J. e Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, 70(3):215–226.

Wolsey, L. A. (1998). *Integer Programming*. Wiley.

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

Ümit Bilge; Kıraç, F.; Kurtulan, M. e Pekgün, P. (2004). A tabu search algorithm for parallel machine total tardiness problem. *Computers & Operations Research*, 31(3):397–414.