UNIVERSIDADE DO ALGARVE

Optimised Search Heuristics: Combining Metaheuristics and Exact Methods to solve Scheduling Problems

Susana Isabel de Matos Fernandes

Doutoramento em Matemática

Especialidade de Investigação Operacional

2008

UNIVERSIDADE DO ALGARVE

Optimised Search Heuristics: Combining Metaheuristics and Exact Methods to solve Scheduling Problems

Susana Isabel de Matos Fernandes

Tese orientada por:

Helena Ramalhinho Lourenço

Doutoramento em Matemática

Especialidade de Investigação Operacional

2008

Resumo

Os problemas de optimização combinatória são objecto de estudo de muitos investigadores com diferentes formações científicas, como investigação operacional, inteligência artificial ou ciências da computação. Enquanto que o trabalho sobre problemas de optimização combinatória de investigadores da área de investigação operacional tem sido dirigido principalmente para o estudo das propriedades matemáticas dos problemas e para o desenvolvimento de algoritmos exactos, os investigadores com formação em ciências da computação e inteligência artifical têm investido principalmente no desenvolvimento de metaheuristicas para encontrar boas soluções para os problemas, tendo em mente a sua aplicação a instâncias reais. Os investigadores das áreas de ciências da computação e inteligência artificial não investem no desenvolvimento de algoritmos exactos talvez por estes terem a fama de serem demasiado lentos para terem utilidade na aplicação a problemas reais. Os investigadores com formação base em investigação operacional não costumam investir no desenvolvimento de metaheuristicas talvez por considerarem que a eficácia destes métodos depende essencialmente da afinação dos seus parâmetros por experiência computacional, carecendo de qualquer fundamentação teórica, e logo desprovidos de interesse matemático. Poderá ser também verdade que alguns investigadores não possuirão competências suficientes em técnicas avançadas de programação e que outros terão falta de conhecimentos de técnicas matemáticas mais elaboradas.

Recentemente, alguns investigadores investiram no desenvolvimento de procedimentos híbridos para resolver problemas de optimização combinatória que combinam algoritmos exactos com metaheurísticas, estreitando desta forma o fosso existente entre investigadores das áreas da matemática e da computação.

Nesta tese, estudamos estes novos métodos que combinam metaheuristicas com algoritmos exactos para resolver problemas de optimização combinatória, salientando quais os métodos que são combinados, como e quais se combinam uns com os outros e a que problemas têm sido aplicados. O capítulo 3 desta tese aborda esta questão, onde se propõe uma nova designação para estes métodos – *Optimised Search Heuristics* – e se apresenta um mapeamento da distribuição do tipo de combinações entre algoritmos

exactos e metaheurísticas pelo tipo de problemas a que são aplicados os novos procedimentos.

Este mapeamento evidencia que existe muito espaço para nova investigação nesta área. Neste trabalho estamos particularmente interessados em usar os algoritmos exactos para conduzir o processo de procura local nas metaheurísticas.

Sobre este tema produzimos e publicámos o artigo *Optimised Search Heuristics* (Fernandes and Lourenço 2007b).

Qualquer novo método desenvolvido para resolver problemas de optimização combinatória terá de ser testado em problemas pertencentes à classe dos NP-hard se quiser captar a atenção das comunidades científicas a trabalhar na área.

Nesta tese escolhemos estudar os problemas de sequenciamento e mais especificamente o problema *job shop scheduling*, famoso pela sua dificuldade tanto em teoria como na prática. Outra razão para escolher este problema para introduzir o novo método desenvolvido prende-se com o facto de a sua estrutura algébrica ter sido já alvo de inúmeros estudos. Provaram-se já muitas propriedades que permitem caracterizar desigualdades válidas que definem algumas facetas do envolvente convexo do conjunto de soluções admissíveis para o problema.

O capítulo 4 é dedicado à apresentação dos problemas de sequenciamento, a sua definição, formulações matemáticas e propriedades da sua estrutura algébrica.

Da indústria automóvel ao controlo de tráfico aéreo encontram-se muitas aplicações de problemas de sequenciamento. Estes problemas definem-se pela necessidade de executar um conjunto de tarefas que partilham um conjunto de recursos. Não se conhece um procedimento determinístico que consiga encontrar a solução óptima em tempo polinomial para a maioria dos problemas de sequenciamento. Assim, a investigação nesta área é muitas vezes orientada para o desenvolvimento de métodos que possam encontrar boas soluções em tempo útil.

A formulação matemática de problemas de sequenciamento e o estudo da sua estrutura algébrica recebeu muita atenção aquando do desenvolvimento de métodos exactos para os resolver. Mas os métodos exactos revelaram-se ineficientes para resolver instâncias reais do problema. Foram então desenvolvidos algoritmos de procura local que, partindo de soluções admissíveis construídas heuristicamente, conseguiam encontrar boas soluções rapidamente. Os métodos de procura local têm a desvantagem de parar no primeiro óptimo local que encontram ao percorrer o espaço de soluções admissíveis. A investigação evoluiu para o desenvolvimento de metaheuristicas, procedimentos em que o processo de procura consegue progredir para outras regiões do espaço de soluções admissíveis após encontrar um óptimo local. Nas metaheurísticas, o processo de procura é gerido “afinando” um conjunto de parâmetros dos algoritmos. Falta uma fundamentação teórica na afinação destes parâmetros, que é em regra geral baseada na experiência computacional para cada problema, e muitas vezes para cada instância. As metaheurísticas têm sido maioritariamente desenvolvidas por investigadores das áreas de inteligência artificial e ciências da computação, que normalmente não incluem técnicas exactas de optimização combinatória nos seus algoritmos. A estrutura algébrica dos problemas tem estado presente na base do desenho de estruturas de vizinhança de métodos de procura local, ainda que muitas vezes apenas de forma implícita. Mas os critérios para gerir várias vizinhanças e a forma de conduzir o processo de procura têm sido determinados quase exclusivamente por experimentação computacional ou intuição.

Neste trabalho desenvolvemos dois métodos da categoria *Optimised Search Heuristics*. Apresentamos resultados computacionais para um vasto conjunto de instâncias de referência de problemas de sequenciamento, assim como apresentamos comparações de desempenho dos métodos desenvolvidos com outros métodos bem sucedidos.

O primeiro dos dois métodos desenvolvidos é um procedimento simples que combina o método metaheurístico *GRASP* com o algoritmo exacto *branch-and-bound*, a que chamamos *GRASP_B&B* e que usamos para resolver o problema *job shop scheduling*. O *GRASP_B&B* é um procedimento muito rápido que constrói soluções admissíveis com uma qualidade aceitável, ideais para serem usadas como soluções iniciais de procedimentos mais elaborados. O método é constituído por duas fases que se repetem a cada iteração. Uma fase de construção onde em cada iteração é acrescentado um novo elemento à solução em construção, sendo esse elemento escolhido de uma forma gananciosa aleatoriezada, isto é, a escolha do novo elemento a incluir na solução é enviesada para elementos que provocam o maior aumento imediato na qualidade da solução. A qualidade com que cada novo elemento contribui para a solução a ser construída é avaliada pelo valor óptimo de sub-problemas de uma única máquina. Os sub-problemas de uma única máquina são resolvidos com o algoritmo exacto *branch-and-bound*. Na outra fase do método, posterior à fase de construção, executa-se uma procura local para passar da solução construída a um óptimo local, antes de seguir para a inserção de um novo elemento na solução.

É apresentada uma comparação do desempenho do nosso *GRASP_B&B* com outros procedimentos aplicados ao mesmo problema, também usados como geradores de soluções iniciais admissíveis para métodos mais elaborados. Nomeadamente, apresentamos comparações com outro método *GRASP* (Binato, Hery et al. 2002) e com um método que incorpora o algoritmo *branch-and-bound* da mesma forma que nós, o *shifting bootleneck procedure* (Adams, Balas et al. 1988).

Produzimos e publicámos o artigo "*A GRASP and Branch-and-Bound Metaheuristic for the Job Shop Scheduling*" (Fernandes and Lourenço 2007) que apresenta o novo método *GRASP_B&B*.

O capítulo 6 desta tese apresenta a estrela principal deste trabalho de investigação, o procedimento *Tabu_VVI*. É um método da categoria dos *Optimised Search Heuristics* que combina um módulo de verificação de desigualdades válidas violadas com um procedimento *Tabu Search*.

O método *Tabu_VVI* começa por utilizar o algoritmo *GRASP_B&B* para construir uma solução admissível inicial. Sobre esta solução é executado um processo *Tabu Search*, produzindo um bom óptimo local. Com o objectivo de prosseguir com a procura no espaço de soluções admissíveis, esse óptimo local é perturbado, sendo parcialmente destruído para depois se reconstruir uma nova solução completa (admissível). Para destruir parcialmente o óptimo local utilizamos um procedimento do tipo ganancioso aleatorizado (*greedy randomised*) para eliminar algumas das suas componentes, dando prioridade às componentes cuja eliminação produz um impacto maior no valor da solução parcial. Seguidamente o procedimento *Tabu_VVI* procura desigualdades válidas para o problema que sejam violadas pela solução parcial produzida. Estas desigualdades vão obrigar a que algumas componentes não sejam consideradas, restringindo desta forma a reconstrução de uma nova solução completa. Assim o percurso da procura no espaço de soluções admissíveis é forçado a saltar para uma região diferente, que será preferencialmente uma região de soluções de melhor qualidade, dado o tipo de desigualdades verificadas. Concretamente, as desigualdades violadas descartam componentes que comprovadamente dariam origem a soluções completas com um valor de função objectivo não melhor do que o da solução encontrada até então com maior qualidade. Esta mudança na direcção do percurso da procura no espaço de soluções admissíveis é conduzida pela informação sobre a estrutura algébrica da instância contida nas desigualdades válidas. É neste sentido que dizemos ser este um método em que a direcção do processo de procura da metaheurística *Tabu Search* é conduzida pela utilização da técnica exacta de verificação de desigualdades válidas violadas para descartar regiões do espaço de soluções admissíveis.

Apresentamos o novo método *Tabu_VVI* com uma aplicação ao problema *job shop scheduling* e relatamos resultados computacionais para um vasto conjunto de instâncias de referência do problema, incluindo comparações de desempenho com outros procedimentos aplicados ao mesmo problema. Nomeadamente, comparamos os resultados computacionais do *Tabu_VVI* com outros métodos que combinam metaheurísticas com técnicas exactas e com os três métodos mais bem sucedidos na aplicação ao *job shop scheduling*; o *Guided Local Search* de Balas e Vazacopoulos (Balas and Vazacopoulos 1998), o *Tabu Search* com *Shifting Bottleneck* de Pezzella e Mirelli (Pezzella and Merelli 2000) e o *Tabu Search* com *Path Relinking* de Nowicki e Smutnicki (Nowicki and Smutnicki 2005).

O nosso método *Tabu_VVI* ganha em comparação com os outros métodos que combinam algoritmos exactos com metaheurísticas, produzindo sempre soluções de melhor qualidade em menos tempo. Quando comparado com os métodos de Balas e Vazacopoulos e de Pezzella e Mirelli, o nosso *Tabu_VVI* revela-se muito competitivo, atingindo resultados do mesmo nível. Na comparação com o método que apresenta até à data os melhores resultados para o problema *job shop scheduling*, o *Tabu Search* com *Path Relinking* de Nowicki e Smutnicki, o nosso método apresenta resultados muito próximos dos deles quando é executado durante aproximadamente o mesmo tempo computacional.

O novo método *Tabu_VVI* é descrito no artigo *“Optimised Search Heuristic Combining Valid Inequalities and Tabu Search”* (Fernandes and Lourenço 2008).

Esperamos que os bons resultados atingidos com este novo procedimento sejam encorajadores e incentivem outros investigadores a ultrapassar o fosso entre as áreas de métodos exactos de optimização combinatória e metaheurísticas, desenvolvendo novos métodos na categoria dos *Optimised Search Heurisitics* que possam tirar partido das vantagens das técnicas de uma e outra área de investigação.

No desenvolvimento deste trabalho, nomeadamente na implementação do módulo de verificação de desigualdades válidas violadas por uma solução parcial do método *Tabu_VVI* aplicado ao problema *job shop scheduling*, deparámo-nos com um obstáculo. As desigualdades válidas são obtidas dos subproblemas de uma única máquina e definidas para todo o subconjunto de operações processadas numa máquina. Ora, o número de subconjuntos de um conjunto com n elementos é 2^n, um número dado por uma função exponencial no tamanho do problema a resolver. O que significa que a complexidade computacional de verificar todos os subconjuntos de todas as máquinas seria incomportável para um algoritmo que se quer eficiente. Decidimos então chegar a uma solução de compromisso não verificando todos os possíveis subconjuntos aquando da procura de desigualdades válidas violadas. O processo de construção dos subconjuntos a ser inspeccionados é enviesado para a construção de subconjuntos com maior possibilidade de potenciar a violação de uma desigualdade válida e funciona incluindo nos conjuntos, uma a uma, as operações de acordo com os seus parâmetros, como a data de disponibilidade, o tempo de processamento e o tempo que a operação permanece no sistema após terminar o seu processamento.

No artigo (Péridy and Rivreau 2005) sobre ajustes locais de limites de janelas de tempo para o processamento das operações nas máquinas é descrito um novo método de enumeração eficiente que poderá ser útil para a geração dos subconjuntos subjacentes às desigualdades válidas. Uma possível linha de trabalho futuro será averiguar a viabilidade prática de implementação deste novo método e testar se tal poderá melhorar a eficiência do novo método *Tabu_VVI*.

Outra forma directa de extender a linha de investigação iniciada nesta tese será a de aplicar o método *Tabu_VVI* a outros problemas de scheduling da classe NP-hard, como por exemplo a versão *total weighted tardiness* do problema *job shop scheduling* ou o problema *generalised job shop scheduling*. Seria também muito interessante aplicar o novo método a instâncias reais de problemas de sequenciamento. Posteriormente poder--se-á evoluir para a aplicação do método a versões multicritério de problemas de sequenciamento, ou a outros problemas para os quais sejam conhecidas desigualdades válidas, implementáveis de forma eficiente, sendo de particular interesse as que definem facetas do envolvente convexo do conjunto de soluções admissíveis do problema em questão.

O objectivo maior desta tese é o de desenvolver a investigação na área da complementariedade entre algoritmos exactos e metaheurísticas, esperando que a cooperação bem sucedida entre métodos das diferentes áreas possa fomentar a colaboração entre investigadores com diferentes formações de base a trabalhar sobre os mesmos problemas de optimização combinatória.

Assim, e resumindo, as principais contribuições desta tese são:

a) a proposta de uma designação para os métodos que combinam algoritmos exactos e (meta)heurísticas e um mapeamento da investigação neste domínio.

A designação *Optimised Search Heuristics* (*OSH*) é proposta para descrever metodologias onde a procura local do método heurístico é de alguma forma orientada por métodos exactos de optimização combinatória. Com estes métodos (*OSH*) pretende-se tirar partido das melhores características de ambos os métodos, metaheuristicos e exactos, fornecendo uma solução integrada que poderá levar a resultados excelentes.

Apresentamos a forma como estes procedimentos têm sido aplicados a problemas de optimização combinatória; construimos um mapeamento de métodos versus aplicações e concluímos que há muitas possibilidades de desenvolver investigação em métodos OSH e também uma grande oportunidade para os aplicar a problemas difíceis.

b) um novo método muito rápido para a construção de soluções admissíveis combinando *branch-and-bound* e *GRASP*.

Desenvolvemos um algoritmo simples para a o problema *job shop scheduling* que combina uma metaheurística de procura local, o *GRASP*, com um método exacto de programação inteira, o *branch-and-bound*. Aqui o *branch-and-bound* é utilizado dentro do *GRASP* para resolver sub-problemas de *one machine scheduling*.

c) um método inovador que combina a metaheurística *Tabu Search* com a verificação de desigualdades válidas violadas.

Desenvolvemos uma metaheurística *OSH* que utiliza a verificação de desigualdades válidas para conduzir a reconstrução de uma solução óptima local que foi parcialmente

destruída. Este novo método é apresentado através de uma aplicação ao problema *job shop scheduling*.

A ideia deste novo método é a de imitar os planos de corte da programação inteira, deixando as desigualdades válidas violadas descartar regiões pouco atraentes do espaço de soluções e orientar a procura de uma solução óptima local para uma região admissível com mais qualidade.

Palavras Chave: Metaheuristicas, Algoritmos Exactos, GRASP, Procura Tabu, Branch-and-Bound, Desigualdades Válidas, Problemas de Sequenciamento

Abstract

Scheduling problems have many real life applications, from automotive industry to air traffic control. These problems are defined by the need of processing a set of jobs on a shared set of resources. For most scheduling problems there is no known deterministic procedure that can solve them in polynomial time. This is the reason why researchers study methods that can provide a good solution in a reasonable amount of time.

Much attention was given to the mathematical formulation of scheduling problems and the algebraic characterisation of the space of feasible solutions when exact algorithms were being developed; but exact methods proved inefficient to solve real sized instances. Local search based heuristics were developed that managed to quickly find good solutions, starting from feasible solutions produced by constructive heuristics. Local search algorithms have the disadvantage of stopping at the first local optimum they find when searching the feasible region. Research evolved to the design of metaheuristics, procedures that guide the search beyond the entrapment of local optima. Recently a new class of hybrid procedures, that combine local search based (meta) heuristics and exact algorithms of the operations research field, have been designed to find solutions for combinatorial optimisation problems, scheduling problems included.

In this thesis we study the algebraic structure of scheduling problems; we address the existent hybrid procedures that combine exact methods with metaheuristics and produce a mapping of type of combination versus application and finally we develop new innovative metaheuristics and apply them to solve scheduling problems. These new methods developed include some combinatorial optimisation algorithms as components to guide the search in the solution space using the knowledge of the algebraic structure of the problem being solved. Namely we develop two new methods: a simple method that combines a GRASP procedure with a branch-and-bound algorithm; and a more elaborated procedure that combines the verification of the violation of valid inequalities with a tabu search. We focus on the job-shop scheduling problem.

Keywords: Metaheuristics, Exact Algorithms, GRASP, Tabu Search, Branch-and-Bound, Valid Inequalities, Scheduling Problems

Acknowledgements

I would like to thank my advisor, Helena Ramalhinho Lourenço, for accepting me as her PhD student, for her warm welcome and for all the support she has given me (scientific, logistic, financial and personal) throughout this period of my life.

I would like to thank the Department of Economics and Business of Universitat Pompeu Fabra in Barcelona for welcoming me as a visiting student - working closely with my PhD advisor was a key factor for the development of this thesis.

The research work was mainly developed during these last three years, while working full time on my thesis, which was made possible by the approval of my leave of absence from the Departamento de Matemática of the Faculdade de Ciências e Tecnologia, Universidade do Algarve.

My PhD research work was sponsored by the POCI2010 programme of the portuguese Fundação para a Ciência e a Tecnologia. Staying in Barcelona would have not been possible otherwise.

Ciência.Inovação 2010 Programa Operacional Ciência e Inovação 2010
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR

I am most grateful and honoured by the love of my family and friends.

Table of Contents

List of Tables

List of Figures

1. Introduction

The present document starts by presenting the motivations underlying the research work of this PhD thesis, highlighting the purpose of the work in section 1.1 and presenting in section 1.2 - Main Scope - the field to which this research work contributes. The structure of the document is described in section 1.3 and this first introductory chapter ends with a summary of the main contributions of our work in section 1.4.

1.1 Purpose

Scheduling problems have many real life applications, from automotive industry to air traffic control. These problems are defined by the need of processing a set of jobs on a shared set of resources. Building a solution means assigning a time interval to each job on each resource. The quality of a solution is measured by means of some objective function, usually related to the time needed or the cost associated to process all the jobs. When solving the problem the goal is to find the solution with the best value for the objective function. For most scheduling problems there is no known deterministic procedure that can solve them in polynomial time. This is the reason why researchers study methods that can provide a good solution in a reasonable amount of time.

In this thesis we intend to develop new innovative metaheuristics and use them to solve scheduling problems. These methods will include some combinatorial optimisation algorithms as components to guide the search in the solution space using the knowledge of the algebraic structure of the problem being solved. We will focus on the job-shop scheduling problem.

1.2 Main Scope

When speaking of combinatorial optimisation problems we define a problem as a set of instances with some common structure (e.g. a graph), including an objective (or cost, or evaluation) function; each instance having a set of feasible solutions. Dealing with an optimisation problem means we want to find the best solution of an instance, that is, the solution of that instance with the best (minimum or maximum) objective function value. A combinatorial optimisation problem is an optimisation problem where the set of solutions is discrete, or can be reduced to a discrete one. Examples of combinatorial optimisation problems are routing, packing, scheduling, matching or network flows problems, just to name a few areas.

Many combinatorial optimisation problems, scheduling problems included, belong to the NP-hard class. There is no knowledge of a polynomial deterministic algorithm that can solve them; but there are polynomial non-deterministic procedures that can "guess" a solution and verify its optimality (Garey and Johnson 1979), (Papadimitriou and Steiglitz 1982). This justifies the development of heuristic methods to solve these problems.

The development of methods to solve scheduling problems was started around the second half of the 20th century, with the boom of constructive heuristics based on sequencing rules (Griffer and Thompson 1960), (Roy and Sussman 1964). The process of building a solution is often performed in two stages, starting with the determination of the sequence of processing the jobs on each resource, and proceeding with the assignment of time intervals for each pair (job, resource).

Much attention was given to the mathematical formulation of scheduling problems and the algebraic characterisation of the space of feasible solutions when exact algorithms were being developed, like branch-and-bound and branch-and-cut (French 1982), (Balas 1985), (Carlier and Pinson 1989), (Applegate and Cook 1991). But exact methods proved inefficient to solve real sized instances. Local search based heuristics were developed that managed to quickly find good solutions, starting from feasible solutions produced by constructive heuristics. Local search algorithms have the disadvantage of stopping at the first local optimum they find when searching the feasible region. Research evolved to the design of metaheuristics, procedures that guide the search beyond the entrapment of local optima, like simulated annealing, tabu search, GRASP, genetic local search or iterated local search (Vaessens, Aarts et al. 1996), (Jain and Meeran 1999). The quality of the solutions achieved has increased considerably from the simple sequencing rules to present metaheuristics. The same goes for the complexity of the algorithms.

In metaheuristics the search process is managed by the fine tuning of a set of parameters of the algorithms. The setting of these parameters still lacks a theoretical foundation. What happens is that parameters are set empirically for each type of problem, and many times for each type of instance. Metaheuristics have been mainly developed by researchers of the fields of artificial intelligence and computer science, who generally do not include traditional combinatorial optimisation techniques in their algorithms.

The algebraic structure of the problems has been, many times only implicitly, in the foundations of the design of neighbourhood structures of local search procedures, but the criteria to manage various neighbourhoods and the way of guiding the search have been widely defined by intuition or experimentation.

In this work we will develop metaheuristics that guide the local search based on the algebraic structure of the problems being solved. Although there have been some efforts devoted to the guidance of the search of solutions based on specific measured characteristics of each instance, like (Schiavinotto and Stutzle 2004), major current research in the scheduling problems field has been mainly concerned with the definition of new neighbourhood structures (Jain, Rangaswamy et al. 2000), (Nowicki and Smutniki 1996) and the achievement of new lower and upper bounds (Goldberg, Paterson et al. 2001), (Dorndorf, Pesch et al. 2002).

The main goal of this thesis is to contribute to the development of local search based metaheuristics that can use information about the algebraic structure of the problems being solved.

Local search methods, the base of most metaheuristics, start with a feasible solution and move step by step to a better neighbouring solution. A neighbour is a solution that can be reached by performing one “simple” transformation (called move) on the current solution. A neighbourhood of a solution is the set of all its neighbours. The search process can verify all the neighbours before choosing the best one, or it can choose the first neighbour being visited that is better than the current solution; or it can check the neighbourhood in some compromised way between these two. The local search process stops at a solution that is better than all its neighbours, a local optimum solution. Generally this local optimum solution is not the global optimum, i.e., the best of all feasible solutions. When it is guaranteed that the local search process stops only at the global optimum, the neighbourhood is said to be an exact one.

To build a local search based metaheuristic there is the need to define the neighbourhood structures; the way to inspect them; when and how to perform intensification or diversification of the search, and the various forms of combining these different features. We will develop metaheuristics where the information redrawn from the algebraic structure can be used to make these design decisions, and we apply these new methods to solve scheduling problems.

To our knowledge this is a very innovative characteristic of this PhD thesis. We do not know of procedures that integrate metaheuristics and exact methods of combinatorial optimisation in this way.

1.3 Structure of this Document

Since the main goal of this work is to develop a new kind of metaheuristic and to apply it to solve combinatorial optimisation problems, we will start in chapter 2 with a brief presentation of the combinatorial optimisation field, followed by a short introduction of existent metaheuristics. We intend to combine metaheuristics and combinatorial optimisation methods, so a brief description of these exact procedures is also presented in chapter 2. These surveys on metaheuristics and exact combinatorial optimisation methods are by no means intended to be detailed and exhaustive; instead they only give a general idea of, and focus on, those methods found in procedures that combine both types.

Chapter 3 presents a literature review of methods that combine metaheuristics and exact methods, along with a classification of the different forms of combining them and the ways they interact. A new name for these new procedures is proposed – Optimised

Search Heuristics - and a mapping of the types of combinations versus problem applications is built.

In chapter 4 we present the most important properties known of the algebraic structure of scheduling problems in general and specifically for the one machine scheduling problem and the job-shop scheduling problem.

We will proceed presenting the proposed methods. In chapter 5 a method that incorporates a branch-and-bound in the construction phase of a GRASP is presented. This method is then used as a “constructor” of solutions for the more elaborate method presented in the next chapter 6, combining the verification of violated valid inequalities with a tabu search.

The document ends with a chapter of conclusions and future work, including the main challenges and difficulties.

1.4 Main Contributions

The chapters 2 and 4 of this text include an overview of the metaheuristics and combinatorial optimisation methods and the algebraic structure of scheduling problems, respectively. There are several journal articles with surveys on metaheuristics and many books dedicated to combinatorial optimisation methods. The structure of scheduling problems has been addressed by a few authors and we introduce some novelty here in the way the results are put together and presented. These introductory chapters are included in this document so it can be a reasonably self-contained text and easier to read.

The main contributions of this thesis are linked with the idea of guiding the search process of a metaheuristic using a procedure from the exact algorithms of operations research. These contributions are described in chapters 3, 5 and 6. They are:

a) a survey study of methods that combine exact and (meta)heuristic methods and a mapping of the research in this field (chapter 3).

The designation Optimised Search Heuristics (OSH) is proposed to describe heuristics where the search process is some how oriented by exact methods from the combinatorial optimisation field. These OSH methods can extract the best features of the metaheuristics and exact methods and provide an integrated solution method that, as proved already by several authors, can lead to excellent results for large scale problems in a short amount of time. We present how these procedures have been applied to combinatorial optimisation problems; build a mapping of procedures versus applications and conclude that there are many research opportunities to develop optimised search heuristics, and also a large opportunity to apply them to difficult and large dimension problems.

b) a new and very fast method for building feasible solutions combining branch-and-bound and GRASP (chapter 5).

We develop a simple algorithm for the job shop scheduling problem that combines a heuristic local search procedure, GRASP, with an exact method of integer programming, branch-and-bound. The branch-and-bound method is used within the GRASP to solve subproblems of the one machine scheduling problem.

c) an innovative method that combines tabu search with violated valid inequalities (chapter 6).

We develop an OSH procedure that uses valid inequalities to reconstruct a local optimal solution that has been partially destroyed. We first build a feasible solution with our GRASP procedure and perform a tabu search to get a “good” local optimal. In order to continue searching the solution space we perturb the current solution partially destroying it and then rebuilding it. A greedy randomised method is used to delete some elements from the local optimal solution. We then test the existence of valid inequalities violated by the partial solution. These allow us to establish a new search path for rebuilding a complete feasible solution, and hopefully lead us to an attractive unexplored region of the solution space. We named this procedure Tabu_VVI.

The idea of this new method is to mimic the cuts in integer programming, letting the violated valid inequalities discard unattractive regions of the solution space and guide the search from a local optimal solution to a more quality region of the search space.

2. Overview of Methods to solve Combinatorial Optimisation Problems

This chapter is an introduction to the research area of combinatorial optimisation. In the first section we introduce the field of combinatorial optimisation problems including some notes on the theory of NP-completeness. Section 2.2 presents an overview of metaheuritics applied to combinatorial optimisation problems. Finally, section 2.3 presents an overview of exact methods in combinatorial optimisation.

2.1 Combinatorial Optimisation

Combinatorial optimisation is commonly defined as “the mathematical study of the arrangement, grouping, ordering, or selection of discrete objects, usually finite in number” (Lawler 1976). Nemhauser and Wolsey (Nemhauser and Wolsey 1988) propose the following generic definition of a combinatorial optimisation problem:

Let $N = \{1,\ldots,n\}$ be a finite set and let $c = (c_1,\ldots,c_n)$ be an *n*-vector. For $F \subseteq N$, define $C(F) = \sum_{j\in F} c_j$. Given a collection of subsets $\mathcal{F}$ of N, the combinatorial optimisation problem is $COP \quad \min\{\, C(F) : F \in \mathcal{F}\}$. The characterisation of a specific combinatorial optimisation problem is determined by the description of the collection $\mathcal{F}$ of subsets on N. (for instance, for shop scheduling problems the subsets are the permutations of jobs for each machine, satisfying job precedence and machine availability constraints).

Combinatorial optimisation problems occur in many diverse scientific areas such as: economics (planning and management), linear and integer programming, graph theory (covering, partitioning, subgraphs, supergraphs, etc…), network design (routing, spanning trees, flow problems, etc…), sets and partitions, storage and retrieval (packing, compressing, etc…), sequencing and scheduling (parallel machine scheduling, shop scheduling, etc…), algebra and number theory, games and puzzles, logic, automata and language theory and program optimisation (code optimisation, etc…). An extensive compendium of combinatorial optimisation problems can be found in (Ausiello, Crescenzi et al. 1999). They arise in many applications like production planning and distribution, allocation of economic resources, crew scheduling and transports routing or gene sequencing. The table 2.1 shows some of the areas and applications where combinatorial optimisation appears.

Table 2.1 Applications of Combinatorial Optimisation

Applications of Combinatorial Optimisation		
Area	Type of problem	Problems
management and efficient use of scarce resources to increase productivity	operational problems	distribution of goods production scheduling machine sequencing
	planning problems	capital budgeting facility location portfolio selection
	design problems	telecommunications network design transportation networks design VLSI circuit design design of automated production systems
statistics		data analysis
		reliability
physics		determination of minimum energy states
cryptography		designing unbreakable codes
mathematics		combinatorics
		maximum common subgraph
		propositional logic maximum satisfability

To state the importance of the combinatorial optimisation field let us just remember that in 1975 L. Kantorovich and T. Koopmans received the Nobel Prize in Economics (there is no Nobel Prize in mathematics) for their work on the optimal allocation of resources.

Historically, the field of combinatorial optimisation starts with linear programming. Combinatorial optimisation problems can be approached as optimisation problems for polyhedra and mathematically formulated as integer linear programs. Many polynomial-time[1] solvable combinatorial optimisation problems, like maximum flow or matching, are in fact special cases of linear programming. So, many general algorithms for (integer) linear programming can be applied to solve combinatorial optimisation problems. There is also a large variety of combinatorial optimisation algorithms designed for a specific problem, taking advantage of some special structure.

A linear program is a problem of minimising (or maximising) a linear function in the presence of linear inequality and/or equality constraints. Formally $LP \quad \min\left\{cx : Ax \le b,\, x \in \Re_+^n\right\}$ where $\Re_+^n$ is the set of non-negative n-dimensional vectors and $x = (x_1, \ldots, x_n)$ are the variables. An instance of the problem is specified by the data (c, A, b) with c being a n-dimensional row vector, b a m-dimensional column vector and A a $m \times n$ matrix. The set $S = \left\{Ax \le b,\, x \in \Re_+^n\right\}$ is called the feasible region and $x \in S$ a feasible solution. The function $z = cx$ is the objective function. A dual program is associated to every linear program (called primal), where each variable of the dual program is related to each constraint of the primal program and each dual constraint to each primal variable. If both programs have feasible solutions then their optimal value is the same and duality relations can be used to compute the optimal solution. The dual of the program LP is the program $DLP \quad \max\left\{yb : yA \le c,\, y \in \Re_-^m\right\}$.

In a linear program finding the optimal solution, reduces to the selection of a solution from the finite set of vertices of the convex polytope defined by the linear constraints. The simplex algorithm of Dantzig (Dantzig 1949), and all its refined versions including primal and/or dual phases, finds an optimal solution to a linear program in a finite number of steps. The simplex algorithm moves from vertex to vertex of the polytope improving the objective function.

An integer linear program is a linear program with all variables being integers $IP \quad \min\left\{cx : Ax \le b,\, x \in \mathrm{Z}_+^n\right\}$. When variables can only assume values 0 or 1 the integer program is also called a binary one. For some special structured matrices, namely totally unimodular matrices, all the vertices of the polytope defined by $\left\{Ax \le b,\, x \in R_+^n\right\}$ are integers. So when A is totally unimodular and b is a vector of integers, solving the $IP \quad \min\left\{cx : Ax \le b,\, x \in \mathrm{Z}_+^n\right\}$ is the same as solving the $LP \quad \min\left\{cx : Ax \le b,\, x \in \Re_+^n\right\}$.

[1] An algoritm has polynomial running time when its running time is bounded by a polynomial in the size of the input data.

This is the case for some problems like maximum flow or matching which can be solved in polynomial time. We are especially interested in solving problems where this property does not hold.

Combinatorial optimisation problems are the subject of study of many practitioners with different scientific backgrounds like operations research, artificial intelligence and computation sciences. While ones are mainly devoted to the study of the mathematical properties of the problems and the development of exact optimisation algorithms; others developed metaheuristic methods, some are especially focused on solving real live applications of these problems. The next two sections 2.2 and 2.3 present, respectively, overviews of metaheuristics and exact methods used to solve combinatorial optimisation problems. But before closing this section on combinatorial optimisation we present a short introduction to the Complexity Theory. We have used the concepts of polynomial time algorithms and NP-hard problems several times now. In the next subsection the theory of NP-completeness is presented, with a clarification of related concepts used in this text.

Notes on Complexity Theory The theory on complexity was developed for decision problems. A problem can be expressed as a relation $P \subseteq I \times S$ where I is the set of problem instances and S is the set of problem solutions. In a decision problem, the relation P reduces to a function $f : I \to S$, where S is a binary set $S = \{yes, no\}$. Given an instance of a decision problem, to solve it is to be able to say if an instance is a yes instance. A decision problem is said to be polynomial solvable (or simply polynomial) if there is a deterministic algorithm to solve it that runs in a number of steps that is not bigger than a polynomial on the length of the encoding size of the input, for all its instances. In other words, if n is the size of the input of the problem and there is a deterministic algorithm that solves it with complexity $O(p(n))$, being $p(n)$ a polynomial function, then the problem is polynomial. The complexity class $\mathcal{P}$ is the class of polynomial decision problems. There are some decision problems for which there is no known polynomial deterministic algorithm that can solve them; but there are polynomial non-deterministic algorithms that do (a non-deterministic algorithm is one that can execute commands of the type “guess”). These problems form the class $\mathcal{NP}$. Clearly $\mathcal{P} \subseteq \mathcal{NP}$.

Given a decision problem P we say that the problem of identifying a *no* instance is its complementary problem. If there is a non-deterministic algorithm that solves the complementary problem of a decision problem in $\mathcal{NP}$, it is said that the complementary problem (identifying a *no* instance) is in Co- $\mathcal{NP}$.

Given two decision problems P_1 and P_2 we say that P_1 is reducible to P_2 if that is a method (named a reduction) to solve P_1 using an algorithm that solves P_2 (this implies that P_2 is at least as difficult as P_1). If the reduction is executed in polynomial time, it is said that P_1 is polynomial reducible to P_2. When all problems in class $\mathcal{NP}$ are polynomial reducible to a problem $P \in \mathcal{NP}$, problem P is an $\mathcal{NP}$ –complete problem.

A deterministic algorithm is pseudo-polynomial when it runs in a number of steps not bigger than a polynomial defined not only on the size of the input but also on se size of the values of the parameters of the instances. There is such an algorithm to solve the well known $\mathcal{NP}$ problem $\{0,1\}$-knapsack, just to name one example.

A decision problem is strongly $\mathcal{NP}$ –complete when it is still $\mathcal{NP}$ –complete even if any instance of length n is restricted to contain integers of size at most $p(n)$. There are no known pseudo-polynomial algorithms to solve strongly $\mathcal{NP}$ –complete problems; their existence would imply $\mathcal{P} = \mathcal{NP}$.

The theory on complexity can be extended to optimisation problems where class $\mathcal{PO}$ is the natural extension of class $\mathcal{P}$; $\mathcal{NPO}$ is the extension of $\mathcal{NP}$, the extension of the $\mathcal{NP}$ –complete class of decision problems is the $\mathcal{NP}$ –hard class of optimisation problems and class strongly $\mathcal{NP}$ –hard corresponds to class strongly $\mathcal{NP}$ –complete.

The definition of an optimisation problem leads to three different problems, corresponding to three different ways of addressing its solution:

1 Constructive Problem – given an instance of the problem find an optimal solution.

2 Evaluation Problem – given an instance of the problem, compute the optimal value of the objective function.

3 Decision Problem – given an instance of the problem and a positive integer k decide if the optimal value for the objective function is not bigger than k (when the goal is to minimise the objective function).

The decision version of an optimisation problem is never harder to solve than its constructive version. If the decision problem is $\mathcal{NP}$ –complete then the constructive problem is $\mathcal{NP}$ –hard.

The questions $\mathcal{P} \neq \mathcal{NP}$? $\mathcal{NP} \neq \mathcal{C}o$ - $\mathcal{NP}$? remain open in the theory of complexity, as well as their counterparts for optimisation problems. The Fig. 2.1 shows the believed relationships among the mentioned complexity classes for decision problems.

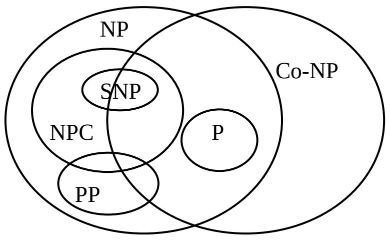


Fig. 2.1 Complexity Classes: P – $\mathcal{P}$; NP - $\mathcal{NP}$; Co-NP - $\mathcal{C}o$-$\mathcal{NP}$; NPC - $\mathcal{NP}$ –complete; PP – pseudo polynomial; SNP - strongly $\mathcal{NP}$ –complete

The book *Computers and Intractability: A Guide to the Theory of NP-completeness* (Garey and Johnson 1979) is a landmark in the literature of the field. There the reader can find the building of the theory, with the proofs to all the statements presented here on complexity classes, along with many other related theorems and propositions. It also proves the $\mathcal{NP}$–hard nature of many optimisation problems. Other references with an exposition on the theory of complexity of optimisation problems are (Ausiello, Crescenzi et al. 1999) (which we have followed closely in this section) and (Papadimitriou and Steiglitz 1982).

2.2 Overview of Metaheuristics

There is no unified definition of what is a metaheuristic among the research communities that work on them, like operations research, computer science and artificial intelligence. Metaheuristics were formerly known as modern heuristics. The new International Journal of Metaheuristics[2] defines them as, and we cote, "*In general, we consider a metaheuristic to be any algorithmic framework, nature inspired or otherwise, that defines a set of heuristic methods that can be directed towards a variety of different optimisation problems. In other words, metaheuristics represent "higher level" heuristic-based algorithms that can be applied to various individual problems with relatively few modifications needing to be made in each case.*"

As stated before, we are especially interested in metaheuristics as methods that allow escaping from, or avoiding, the local optimum entrapment of a search process when solving a combinatorial optimisation problem. There are many different types of metaheuristics with different underlying philosophies. It is difficult either to group or classify them but it seems to be consensual to consider two main groups: those that avoid getting stuck on a local optimum, working with a population of solutions and performing a biased sampling of the solutions space; and the ones that escape from local optima, working with only one current solution at a time. Metaheuristics are often classified according to the methods adopted in order to escape or avoid local entrapment. Such methods include the use of pure randomness; the use of neighbourhood-modification processes; the inclusion of penalties or weights to modify the objective function; the use of a statistical model for the frequency of the characteristics of the solutions chosen, etc. Other ways of looking at metaheuristics are its memory usage - short and/or long-term memory -, or the balance between intensification and diversification processes, that is, the exhaustive search of a region around a good solution and the orientation of the search to a more distant and unexplored region.

[2] http://www.inderscience.com/browse/index.php?journalCODE=ijmheur

Next, we will briefly describe the most widely used metaheuristics. Metaheuristics are general methods that can be applied to different kind of problems with very little problem specific adaptations. They are very popular in finding good solutions to many optimisation problems from different research fields like artificial intelligence, computer science or combinatorial optimisation, among others. For more information on the state of the art of the field of metaheuristics please refer to surveys like “A survey of AI-based meta-heuristics for dealing with local optima in local search” (Mills, Tsang et al. 2004) and “Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison” (Blum and Roli 2003), and their references.

2.2.1 Metaheuristics that work with One Solution

In the group of metaheuristics that work with only one current solution at a time there are those which escape local optima mainly by adding some form of randomness, like Simulated Annealing, Greedy Randomised Adaptive Search Procedure (GRASP) or Iterated Local Search; others perform some modification on the neighbourhood structure, like Variable Neighbourhood Search or Tabu Search; finally there are those which use penalties or weights to modify the objective function like Guided Local Search.

Simulated Annealing The simulated annealing procedure is not a constructive method so it needs an initial solution. It is a local search method that allows moves resulting in solutions of worse quality than the current one in order to escape local optima. The probability of accepting such a move, called the temperature, is decreased during the search. The method tries to mimic the annealing process of metals and glass. Early references to simulated annealing trace back to (Metropolis, Rosenbluth et al. 1956), (Kirkpatrick, Gelatt et al. 1983), (Cerny 1985). A variation on simulated annealing is simulated jumping (Amin 1999). See (Van-Laarhoven, Aarts et al. 1992) for applications to the job shop scheduling.

GRASP The acronym GRASP (Feo and Resende 1995), (Resende and Ribeiro 2003) means “greedy randomised adaptive search procedure”. It is an iterative constructive process where each iteration consists of two steps: a randomised building step of a greedy nature and a local search step. At the building phase, a feasible solution is constructed by joining one element at a time. Each element is evaluated by a heuristic function and incorporated (or not) in a restricted candidate list (RCL) according to its evaluation. The element to join the solution is chosen randomly from the RCL. After a new element is added, if the solution has already more then one element the algorithm proceeds with the local search step. The current solution is updated by the local optimum and this two-step process is repeated until the solution is complete. See (Binato, Hery et al. 2002) for an application to the job shop scheduling problem.

Iterated Local Search The iterated local search procedure (Stützle 1999), (Lourenço, Martin et al. 2002) applies local search to an initial solution until a local optimum x is reached. Then, it randomly perturbs the solution, usually called a kick move phase, and the local search re-starts. The new solution y is compared to the previous one (x) and an acceptance criteria decides which solution, x or y, is used to continue the procedure.

Tabu Search The tabu search method (Glover 1986), (Glover and Laguna 1997) keeps track of the most recently visited solutions maintaining a tabu list that stores some features of the solutions or of the moves that lead to them. During the search process, after a local optimum is reached, moves to solutions present in the tabu list are forbidden. The best of the solutions not in the tabu list is chosen, even if it is worse than the current one. Because the tabu list does not store the complete solutions, a tabu move can be performed if it satisfies some aspiration criteria, usually if it is best than the best solution found so far. See (Nowicki and Smutnicki 2005) for applications to job shop scheduling. There are several variations on tabu search like robust tabu search (Taillard 1991; Smyth, Hoos et al. 2003), iterated robust tabu search (Battiti and Tecchiolli 1994; Smyth, Hoos et al. 2003) or reactive tabu search (Battiti and Tecchiolli 1994).

Variable Neighbourhood Search The variable neighbourhood search procedure (Hansen and Mladenovic 1997), (Mladenović and Hansen 1997),(Hansen and Mladenović 2001) uses several different neighbourhood structures. Given a solution the process looks for neighbours in a neighbourhood A. After a local optimum is achieved, the process changes to a different neighbourhood B, usually bigger than A. The method cycles through all the different neighbourhoods used; it stops when there is no improvement for any of them.

Guided Local Search The guided local search procedure (Voudouris 1997), (Voudouris and Tsang 1999) dynamically changes the objective function of the problem being solved. After a local optimum is found, the objective function is changed so that penalties for the features of the local optimal solution are included, and so other neighbours become more attractive.

2.2.2 Metaheuristics that work with a Population of Solutions

In the group of metaheuristics that work with a population of solutions which avoid local entrapment by performing combinations and mutations of the solutions, some are strongly based in randomness, like Genetic Algorithms and Memetic Algorithms; others are based on the underlying search space like Scatter Search, and Path Relinking; and others, still, which are based on probability models like Ant Colony Optimisation or Estimation of Distribution Algorithms.

Genetic Algorithms The procedures designated genetic algorithms (Holland 1975) are inspired in natural evolution. They consider a set of initial solutions, called the initial population and perform a number of operations to the individuals of the population to generate new solutions. These operations include the combination of elements from different solutions (crossover), and the modification of some elements of one solution – mutation. The best evaluated individuals are chosen to constitute the next generation. Frequently, the elements of the solutions are represented by a binary code. See (Yamada and Nakano 1992) for an application to the job shop problem.

Memetic Algorithms The procedures designated memetic algorithms (Moscato 1989), also known as genetic local search, try to mimic the "cultural evolution" by incorporating local search into a genetic algorithm framework. An initial population is generated and local search is applied to each solution. Crossover and mutation operators are used to generate new individuals and local search is applied again to the resulting solutions. The best ones, according to quality and diversity, are chosen to constitute the next generation.

Scatter Search The scatter search method (Cung, Mautor et al. 1997), (Glover 1999), (Glover, Laguna et al. 2000) generates new solutions by linear combination of two solutions chosen from a reference set. The combination of solutions can produce infeasibility so there is usually a procedure to recover feasibility. Local search is performed to improve the new generated solutions. The reference set is updated with the best solutions generated while maintaining some level of diversity. See (Yamada and Nakano 1995) and (Jain and Meeran 1998) for applications to the job shop scheduling problem.

Path Relinking The path relinking method (Glover 1999) finds new solutions by generating paths between and beyond solutions of a reference set. It is analogous to scatter search, but replaces the linear combinations in the Euclidean space by paths in the neighbourhood space. While traversing paths starting from an initial solution, moves must progressively introduce attributes of a guiding solution. Path relinking is most commonly used as a component of other metaheuristics, like tabu search or GRASP. See a GRASP with path relinking (Aiex, Binato et al. 2003) and a tabu search with path relinking (Nowicki and Smutnicki 2005) for applications to the job shop scheduling problem.

Ant Colony Optimisation The ant colony optimisation procedures (ACO) (Dorigo, Maniezzo et al. 1996) are inspired in the behaviour of real ants when walking between food sources and their nests. Ants deposit a pheromone in the walking path. Paths with stronger pheromone concentration are more frequently chosen by the ants. The ACO mimics the pheromone trails with a probabilistic model. The metaheuristic is a constructive procedure where solutions are constructed adding components one by one to a partial solution under consideration. Each artificial ant performs a randomised walk on a completely connected graph whose vertices are the components of the solutions and the arcs are the set of connections between them. Each vertex and each connection have associated probability values that are updated during the process according to the frequency of usage and the quality of the solutions built. See (Dorigo and Stützle 2002) for applications of the method.

Estimation of Distribution Algorithms The procedures designated estimation of distribution algorithms (Mühlenbein and Paaβ 1996) are based on populations of distribution functions that evolve as the search progresses. They use probabilistic modelling of the elements of good solutions to estimate a distribution over the search space. This distribution is used to produce the next generation of solutions. The distribution function is then updated. See (Pelikan, Goldberg et al. 1999) for a survey on these methods and (Larrañaga and Lozano 2002) for its applications.

2.3 Overview of Combinatorial Optimisation Methods

Combinatorial optimisation problems can be approached as optimisation problems for polyhedra and mathematically formulated as integer linear programs. So, many general algorithms for (integer) linear programming can be applied to solve combinatorial optimisation problems.

General algorithms for integer programming fall into two partially overlapping categories: the enumerative methods like branch-and-bound or dynamic programming that perform some kind of intelligent enumeration of all possible solutions; and the cutting-plane methods that solve some relaxation of the original problem and then add a linear constraint that throws away the solution of the relaxation but does not exclude any integer feasible point.

In the preface of the Handbook of Combinatorial Optimization (Du and Pardalos 1998), Ding Zhu Du and Panos M. Pardalos identify 4 major factors that had a great effect on combinatorial optimisation, after its “birth” with the simplex method: on one side, the discoveries of the ellipsoid method in 1979 (Khachiyan 1979) and the interior point method in 1984 (Karmarkar 1984) providing polynomial time algorithms for linear programming, in the sense that linear programming relaxations are often the basis for combinatorial optimisation algorithms; on the other side the design of efficient integer programming software and the availability of parallel computers allowing us to solve to optimality problems with thousands of integer variables and approximate solutions to problems with millions of integer variables.

The next sections present short descriptions of some combinatorial optimisation methods, procedures that are combined with metaheuristics in the new proposed methods OSH.

Dynamic programming This method was first introduced by Richard Bellman (Bellman 1957) in the early 50ths for solving multistage decision problems (either deterministic or stochastic). At each stage a decision is required and each stage has a number of states associated with it. The decision at one stage transforms one state into a state of the next stage. Going through the stages (either in a forward or backward way), a new state in the problem is determined only by the state on the previous (or following) stage and the decision taken there. Bellman defined the principle of optimality which states: “An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision” (Bellman 1957).

Any optimisation problem that can be formulated verifying this optimality principle can be solved by dynamic programming. The algorithm performs an intelligent enumeration of all feasible solutions and it can solve an optimisation problem with a fixed number of constraints in pseudo-polynomial time (dependent on the size of the input and on the biggest absolute value of the data of the instance), through a recursive optimisation process that decomposes the initial problem into a nested family of subproblems.

Formally let T denote the set of stages, s_t the state and x_t the decision variable at stage t. Given the initial state s_0, the objective function is given by $z = \sum_{t=1}^{T} f_t(s_{t-1}, x_t)$ and each state is $s_t = g_t(s_{t-1}, x_t)$ for $t = 1 \ldots T$. So both the contribution to the objective function of stage t and the state in stage t depend only on s_{t-1} and x_t. This way the original problem can be addressed by recursively solving a series of T subproblems; each with only one decision variable and one state constraint, making the optimal decision for that state in each subproblem.

Considering an optimisation problem, the issue of applying dynamic programming to solve it lies on the difficulty in identifying stages and states. Different formulations with different stages and states can be defined for the same problem and the process can be thought of as a forward or backward recursion. For instance (Nemhauser and Wolsey 1988) presents two dynamic programming algorithms for the integer knapsack problem $z(b) = \max\left\{ \sum_{j \in N} c_j x_j : \sum_{j \in N} a_j x_j \leq b, x \in Z_+^n \right\}$, one with complexity $o\left(nb^2\right)$ and other with complexity $o(nb)$.

Besides the operations research field, dynamic programming is also very popular in the areas of economics (Adda and Cooper 2003) and computer science (Bertsekas 2000), among others. For more details on dynamic programming please refer to (Denardo 2003).

Branch-and-bound The branch-and-bound method has its origins in a work by Land and Doig (Land and Doig 1960) and further improvements by Dakin (Dakin 1965). The method constructs a tree structure as it searches the solution space using two main tools: branching and bounding.

Branching is a splitting procedure that, given a set of feasible solutions S builds a partition of mutually exclusive sets S_i ($S = \bigcup S_i, i \geq 2$). Note that $\min\{cx : x \in S\} = \min\{\min_i \{cx : x \in S_i\} : S = \bigcup S_i\}$. A set of solutions is represented by a node in the search tree and each subset in the partition is represented by a child node.

Bounding is the procedure of computing upper and lower bounds for the minimum value of the objective function within each (sub)set of solutions. If the lower bound for some node of the tree is greater than the global upper bound (usually the value of the best feasible solution found) then that node can be pruned, that is, the search does not proceed through that node. If it can be proved that the best descendant of a node S_a is at least as good as the best descendant of a node S_b, then S_a dominates S_b and the latter can be discarded. Nodes that have not yet been branched and that were not pruned are called active nodes.

The method stops when the upper bound matches the lower bound or when there are no more active nodes in the tree (in practice the procedure is often terminated after a given time; or when the gap between upper and lower bounds falls below a certain value).

When designing a branch-and-bound algorithm there are many strategic choices to be made (which quite often depend heavily on the problem at hand). At the beginning of the method there is the need to choose the way of computing an upper bound, usually done by heuristically building a feasible solution. There are many different ways for partitioning the solution space – a branching scheme must be chosen. Lower bounds can be tight but computationally expensive or not so tight but computed fast. How to use lower bounds and dominance relations? At each branching step which node should be branched? Recently experiences have been made using interior point methods to solve each subproblem (Lee and Mitchell 1997), replacing the widely used simplex method. The design choices critically dictate the efficiency of the method and there is not one universal layout that works for all applications. See (Mitchell and Lee 2001) for details on partitioning strategies, branching variable selection, node selection, preprocessing and reformulation, and subproblem solver.

Cutting-plane A cutting plane method was first developed by Gomory; the fractional cutting-plane algorithm designed to solve integer linear programs with the simplex method (Gomory 1958). It starts by solving the linear relaxation of the integer problem using the simplex method. If the optimal solution found for the linear relaxation is integer, then the problem is solved. If not, the value of this solution is a lower bound for the optimum (on a minimisation problem). In this case a linear restriction, called cutting-plane or simply cut, is added to the linear relaxation. It cuts off the current optimal solution and does not cut any integer solution of the original problem. These restrictions are also called valid inequalities. The linear relaxation with the new restriction is again solved using the simplex method and the process is repeated until we get an integer solution. Gomory (Gomory 1958) described a method for generating these cuts and proved it stops at an integer solution after a finite number of iterations. These cuts are known as Chvátal-Gomory cutting-planes, due to the parallel work by Chvátal (Chvátal 1973).

Geometrically, a linear relaxation $S_{LR} = \left\{ Ax \leq b,\, x \in R_+^n \right\}$ is a convex polytope that includes all feasible solutions to the integer problem $IP \quad \min\left\{ cx : Ax \leq b,\, x \in Z_+^n \right\}$, and excludes all other integer solutions. Many different polytopes have this property so an integer program has many linear relaxations. More generally, if we set apart the constraints in the program into different groups, a relaxation of a problem happens whenever a group of constraints is dropped out. A cutting-plane method takes advantage of this multiplicity of possible relaxations by finding a sequence of relaxations that more tightly constrain the solution space until eventually a feasible solution for the original integer problem is obtained. Ideally one would like to use the convex hull of the feasible solutions $S = \left\{ Ax \leq b,\, x \in Z_+^n \right\}$ as a relaxation; that is, the smallest convex set that contains S - $Conv(S)$. Finding the optimal solution on $Conv(S)$ would automatically lead to the optimal solution to the original integer program. However, in general, this polytope will have exponentially many facets and be difficult to construct. Typical relaxations form a polytope that strictly contains the convex hull and has vertices other than the integer solutions that solve the unrelaxed problem.

The number of cutting-planes we need to introduce to the feasible set of the linear relaxation depends on how far the relaxation is from $Conv(S)$ in the region of the optimal solution. So the method can be very time consuming.

Along with the fractional Chavátal-Gomory cuts, there are other types of cuts: when a formulation has a family of constraints with an exponential number of inequalities, it is usual to solve the relaxation without this family of constraints and to add them as cutting planes as needed (Dantzig, Fulkerson et al. 1954); the knapsack cuts use the notion that a knapsack problem has only one linear inequality constraint, they find facets for each knapsack problem and add them to the relaxation of the original problem as cuts (Crowder, Johnson et al. 1983); there are also the lift-and-project or disjunctive cuts (Balas, Ceria et al. 1996) for 0-1 problems where each variable can be fixed to 0 and to 1 generating a set of disjunctive inequalities; and the Fenchel cuts (Boyd 1994) that use ideas from lagrangean and convex dualities.

The cutting-plane method of Gomory has the disadvantage of only getting an integer feasible solution when the optimum is reached. There are some primal cutting plane algorithms where the current solution is always an integer one; the algorithm of Padberg and Hong (Padberg and Hong 1980) developed to solve the traveling salesman problem is an example. Primal cutting plane algorithms start with a solution that is an extreme point of the convex hull of the feasible integral region and generate cuts that enable moving from one extreme point to another adjacent extreme point of the convex hull, improving the value of the objective function. Besides the difficulty of finding the initial solution, there is the need to find strong cutting planes, that is, valid inequalities defining the facets of the convex hull of the integer feasible region.

Branch and cut The branch-and-cut is a procedure that combines branch-and-bound with cutting-planes. A pure branch-and-bound approach can be speeded up considerably by the use of a cutting plane scheme, because the cutting planes lead to a considerable reduction in the size of the tree. See for instance (Padberg and Rinaldi 1991).

At a node of the branch-and-bound tree, cutting-planes are added to tighten the relaxation, before branching. As finding good cuts can be computational expensive, cuts are usually not included in every node of the tree. When executed, the insertion of new cuts stops when the solution for the latter relaxation is not significantly better than the one of the previous relaxation of that node of the tree. There is a version of the method where cuts are introduced only in the root node of the branch-and-bound tree, called cut-and-branch. The cuts introduced in a node of the tree can be global, that is, valid for the original problem, or local, which means they are valid only for the subproblem of that branch of the tree.

Column Generation The method column generation was first suggested by Ford and Fulkerson (Ford and Fulkerson 1958). A column generation algorithm is based on the idea of solving a problem by considering only a subset of its variables, including new variables in the formulation, and dropping others, at each step. Such an algorithm may be of use when a linear problem is too large to handle, having too many variables (columns) and a relatively small number of constraints. Since the majority of the variables will have value zero in an optimal solution of a linear program, being non basic variables, the problem can be solved considering only a subset of them; the ones that are likely to improve the objective function. But in order for it to be practical there is the need to efficiently solve the subproblem of identifying which variable should enter the problem, the so called pricing problem. The original problem is usually reformulated using different variables, thus evidencing a structure that allows splitting it into a master problem and separable subproblem(s). The Dantzig Wolfe decomposition algorithm (Dantzig and Wolfe 1960) is a successful example of these methods.

The procedure starts by solving the master problem with only a subset of the variables. Given a solution, the dual prices of the constraints are used to define the objective function of the pricing problem, which is the minimum reduced cost of the left out variables. The pricing problem is solved and its objective value is the reduced cost of the variable to enter the problem. If this objective value is not negative then the solution to the master (minimisation) problem is optimal; if not, the corresponding new variable is included and the process is repeated.

Integer programming column generation algorithms were presented in (Barnhart, Johnson et al. 1998) and (Vanderbeck and Wolsey 1996).

For a detailed work on theory and practice of column generation in linear and integer programming please refer to (Lübbecke and Desrosiers 2005).

Branch-and-Price The idea underlying branch-and-price (Desrosiers, Soumis et al. 1984) is similar to the one of branch-and-cut; except that branch-and-price executes column generation (addition of variables) instead of the row generation (addition of inequalities) of branch-and-cut. Pricing and cutting are viewed as complementary procedures for tightening the relaxation of a problem.

At each node of the branch-and-bound tree, column generation is applied until no new variable enters the problem. If the current solution is not yet feasible for the original problem then the node is branched. Special rules for branching are required to develop an effective branch-and-price algorithm. See (Barnhart, Johnson et al. 1998) for more information on the subject.

Branch-and-Cut-and-Price These procedures are generalisations of the branch-and-bound method that perform both cutting plane and column generation at the nodes of the tree. Special care is mandatory when combining both column and row generation in order that one does not destroy the special structure needed for the other to be effective. These procedures are by no means easy to design but they do achieve good results; see (Vanderbeck 1998), (Akker, Hurkens et al. 2000), (Barnhart, Hane et al. 2000) or (Fukasawa, Lysgaard et al. 2004) for successful examples. There are some available frameworks online for implementing branch-and-cut-and-price algorithms, like MINTO – Mixed INTeger Optimizer[3] and ABACUS – A Branch-And-Cut System[4] (Jünger and Thienel 2000).

Lagrangean Relaxation This method is a relaxation technique which works by moving hard constraints into the objective function (Held and Karp 1970), (Held and Karp 1971).

When it is possible to set apart the constraints in the program $IP \quad \min\left\{cx : Ax \le b,\, x \in Z_+^n\right\}$ into two groups, say $[A_1 \mid A_2]x \le [b_1 \mid b_2]$, such that the IP relaxation $\min\left\{cx : A_1 x \le b_1,\, x \in Z_+^n\right\}$ is an easy program to solve; if we add the previously discarded constraints to the objective function we get the lagrangean relaxation $IP_{LR} \quad \min\left\{cx + \lambda(A_2 x - b_2) : A_1 x \le b_1,\, x \in Z_+^n\right\}$ where $\lambda \in R_+^{m_2}$ is a vector of non negative weights, called the lagrangean multipliers. If the constraints $A_2 x \le b_2$ are violated, the quantity $A_2 x - b_2$ will be positive and the objective function is penalised. For any fixed values of λ, the optimum of the IP_{LR} is never bigger than the optimal

[3] http://coral.ie.lehigh.edu/minto/
[4] www.informatik.uni-koeln.de/abacus/

value of IP, so we can address the original program IP by solving the lagrangean dual program $IP_{LR_\lambda} \quad \max\{ IP_{LR} : \lambda \geq 0\}$.

When designing a lagrangean relaxation algorithm there are many decisions to be made that will affect its efficiency: there are different lagrangean relaxations for the same problem which can generate lower bounds more or less tight; reformulating the problem prior to relaxation can be a good choice; the lagrangean subproblem may be decomposed into smaller problems; etc. A very important aspect is the search for optimal multipliers, to which the choice of the method to solve the lagrangean dual program (subgradiente method, dual ascent method, etc) is crucial. Please see (Guignard 2003) for a detailed discussion on lagrangean relaxation methods.

For further reading on combinatorial optimisation methods please refer to (Papadimitriou and Steiglitz 1982), (Schrijver 1986) or (Nemhauser and Wolsey 1988).

3. Optimised Search Heuristics

Recently, a new class of hybrid procedures, which combine local search based (meta) heuristics and exact algorithms of the operations research field, has been designed to find solutions for combinatorial optimisation problems. This research topic is becoming very prominent and caught the a attention of several researchers; see the recent surveys (Blum and Roli 2003), (Cotta 1998), (Cotta, Talbi et al. 2005), (Dumitrescu and Stützle 2003), (El-Abd and Kamel 2005), (Puchinger and Raidl 2005) and (Raidl 2006) for overviews of different angles. We designated these methods by Optimised Search Heuristics (OSH) since the search process is some how oriented by exact methods from the combinatorial optimisation field. Different combinations of different procedures are present in the literature, and there are several applications of the OSH methods to different problems (see the web page http://www.econ.upf.edu/~ramalhin/OSHwebpage/index.html). The main advantage of the OSH methods is that they combine different techniques with the objective of solving difficult and very large scale problems in a short amount of time.

In this third chapter we present how procedures that combine metaheuristics and exact algorithms, the OSH methods, have been applied to combinatorial optimisation problems. We compare and examine the correspondences of two existing classifications of such procedures. We then propose a more general classification by renaming an existing item and adding a new one. To stress the distribution of these applications over the different problems of combinatorial optimisation, we group them following a classification of NP optimisation problems and outline the combined use of heuristic and exact techniques. This survey on OSH methods contributes to a review of the state-of-the-art of the application of OSH methods to solve combinatorial optimisation problems.

3.1 Classifications of OSH Procedures

Two classifications of Optimised Search Heuristics can be found in the literature. The first one, by Dumitrescu and Stützle (Dumitrescu and Stützle 2003), presents a classification of solution methods that combines local search with exact algorithms. In particular, they consider that the main framework is based on local search and the subproblems are approached by exact methods. These authors consider the following categories:

DS.1 - exact algorithms to explore large neighbourhoods within local search.

DS.2 - information of high quality solutions found in several runs of local search is used to define smaller problems solvable by exact algorithms.

DS.3 - exploit lower bounds in constructive heuristics.

DS.4 - local search guided by information from integer programming relaxations.

DS.5 - use exact algorithms for specific procedures within metaheuristics.

The next classification, due to Puchinger and Raidl (Puchinger and Raidl 2005), considers the combination of exact methods and metaheuristics and includes the following categories:

PR.1 – collaborative

Algorithms exchange information but are not part of each other.

The authors consider two subcategories: one, sequential the other parallel and intertwined.

PR.1.1 - sequential execution

One technique does a preprocessing before the other or the second one is a post processing of the solution(s) generated by the first. Sometimes both techniques have equal importance and we cannot speak of pre or post processing.

PR.1.2 - parallel or intertwined execution

In parallel execution several processors perform simultaneous tasks acting as teams and interchanging information. In intertwined execution a single processor executes some steps of one procedure, then some steps of another.

PR.2 - integrative combinations

One technique is a subordinate embedded component of the other technique. The authors consider the following subcategories:

PR.2.1 - incorporating exact algorithms in metaheuristics

PR.2.1.1 - exactly solving relaxed problems

Solutions to relaxations heuristically guide neighbourhood search, recombination, mutation, repair and/or local improvement.

PR.2.1.2 - exactly searching large neighbourhoods

Exact algorithms are used to search neighbourhoods in local search based metaheuristics.

PR.2.1.3 - merging solutions

Exact algorithms are used to solve sub problems generating partial solutions. Merging these partial solutions is iteratively applied within a metaheuristics.

PR.2.1.4 - exact algorithms as decoders

In evolutionary algorithms where solutions are incompletely represented in the chromosome, exact algorithms are used to find the correspondent best solution.

PR.2.2 - incorporating metaheuristics in exact algorithms

PR.2.2.1 - metaheuristics for obtaining incumbent solutions and bounds

Metaheuristics are used to determine bounds and incumbent solutions.

PR.2.2.2 - metaheuristics for column and cut generation

In branch-and-cut and branch-and-price algorithms, metaheuristics are used to dynamically separate cutting-planes and pricing columns, respectively.

PR.2.2.3 - metaheuristics for strategic guidance of exact algorithms

Metaheuristics are used to determine the branching strategy in branch-and-bound techniques.

PR.2.2.4 - applying the spirit of metaheuristics

Branch-and-bound it self is used for doing the local search. No explicit metaheuristic is used.

3.1.1 Connecting the Classifications Dumitrescu & Stützle - Puchinger & Raidl

Almost all items in the classification of Dumitrescu and Stützle correspond to sub items of item PR.2.1 (incorporating exact algorithms in metaheuristics). The exceptions are procedures classified by Dumitrescu and Stützle in item DS.2. (information of high quality solutions found in several runs of local search is used to define smaller problems solvable by exact algorithms), that have a sequential nature, running a local search based heuristic several times before an exact algorithm; and also the work of (Umetani, Yagiura et al. 2003), allocated to item DS.4, that sequentially executes tabu search after solving the integer programming relaxation.

Some works included in item DS.1, exactly searching large neighbourhoods, can be viewed as a merging solutions kind of procedure.

We introduce a new item in the classification of Puchinger and Raidl, 2.1.5 exact algorithms for strategic guidance of metaheuristics. Here we include all works of item DS.3.

We believe item PR.2.1.3. merging solutions should be generalised and renamed exactly solving sub problems.

We can say that the classification of Dumitrescu and Stützle is more specific and the one of Puchinger and Raidl is more general.

Table 3.1 Correspondence Between Classifications of OSH Procedures

Correspondence between classifications	
Dumitrescu and Stützle	Puchinger and Raidl
DS.1. exact algorithms to explore large neighbourhoods within local search	PR.2.1.2. exactly searching large neighbourhoods 2.1.3. merging solutions – exactly solving sub problems
DS.2. information of high quality solutions found in several runs of local search is used to define smaller problems solvable by exact algorithms	PR.1.1. sequential execution
DS.3. exploit lower bounds in constructive heuristics.	**new proposed item** 2.1.5. exact algorithms for strategic guidance of metaheuristics
DS.4. local search guided by information from integer programming relaxations	PR.1.1. sequential execution
DS.5. use exact algorithms for specific procedures within metaheuristics	PR.2.1.3. merging solutions – exactly solving sub problems

3.1.2 Classification of Procedures versus Problem Type

Using the classification of NP optimisation problems proposed by Crescenci and Kann in http://www.nada.kth.se/~viggo/problemlist/, we show the distribution of the OSH heuristics application to the different combinatorial optimisation problems. In Table 3.2 we present a mapping of the problem type versus the type of combination used. Each entry of the table consists of the reference to the paper(s) and the initials of the exact and metaheuristic methods combined. Please consult the legend of the table for description of methods initials.

We can see that a lot of the research of procedures that combine metaheuristics with exact algorithms has been dedicated to the job shop scheduling problem and to routing problems. Packing problems and the multiple constraint knapsack problem have also received some considerable attention, as well as the more general class of mixed integer programming problems. We believe this can be viewed as a measurement of both the difficulty and the practical relevance of these problems. Practitioners are still not satisfied with the results achieved by traditional applications from stand-alone fields of knowledge.

When looking at the type of combination implemented, we see that the most popular are sequential execution, exactly searching large neighbourhoods (where dynamic programming is the most used exact algorithm) and exactly solving subproblems. Genetic algorithms have been the metaheuristics procedures more frequently used in combination with exact algorithms, maybe because of its low performance on their own.

The most common exact algorithms in these OSH procedures are, aside from dynamic programming, linear relaxations and branch-and-bound.

We believe that using exact algorithms for strategic guidance of metaheuristics is a very promising line of research. This way we can profit from the fast search of the space of solutions of the metaheuristics without getting lost in a "wandering" path, due to the guidance given by the exact algorithms. We find that another very interesting idea is the one of "applying the spirit of metaheuristics" when designing exact algorithms.

This analysis of hybrid procedures to solve combinatorial optimisation problems is presented in the paper *Hybrids combining Local Search Heuristics with Exact Algorithms* (Fernandes and Lourenço 2007b). Its main conclusion is that there are many research opportunities to develop Optimised Search Heuristics and a large opportunity to apply them to difficult problems. The OSH methods can extract the best features of the Metaheuristics and Exact Methods and provide an integrated solution method which, as proved already by several authors, can lead to excellent results.

In Annex A the reader can find a short abstract for each of the OSH procedures referenced, ordered by type of combination.

Table 3.2 Mapping Problem Type Versus the Type of OSH Procedures

	1.1 Sequential execution	**1.2 Parallel or intertwined execution**	**2.1.1 Exactly solving relaxed problems**	**2.1.2 Exactly searching large neighbourhoods**	**2.1.3 Exactly solving sub problems**	**2.1.4 Exact algorithms as decoders**	**2.1.5 Exact algorithms for strategic guidance of metaheuristics**	**2.2.1 Metaheuristics for obtaining incumbent solutions and bounds**	**2.2.2 Metaheuristics for column and cut generation**	**2.2.3 Metaheuristics for strategic guidance of exact algorithms**	**2.2.4 Applying the spirit of metaheuristics**
Mixed Integer	(Pedroso 2004) **LS, LR**				(Pedroso 2004) **TS, BB**					(French, Robinson et al. 2001) **BB, GA** (Kostikas and Fragakis 2004) **BB, GP**	(Danna, Rothberg et al. 2005) **BC, LS** (Fischetti and Lodi 2003) **BB, LS**
Graph Colouring					(Marino, Prugel-Bennett et al. 1999) **GA, LP**				(Filho and Lorena 2000) **CG, GA**		
Frequency Assignment							(Maniezzo and Carbonaro 2000) **LR, D, BB, ACO**				
Partitioning				(Ahuja, Orlin et al. 2000) **LS, DP** (Ahuja, Ergun et al. 2002) **LS, DP**	(Yagiura and Ibaraki 1996) **GA, DP**						
Maximum Independent Set					(Aggarwal, Orlin et al. 1997) **GA, IP**						
Maximum Clique					(Balas and Niehaus 1998) **GA, IP**						
Network Design					(Büdenbender, Grünert et al. 2000) **LS, IP**						(Danna, Rothberg et al. 2005) **BC, LS**
p-Median	(Rosing and ReVelle 1997) (Rosing and ReVelle 1998) (Rosing 2000) **LS, BB**									(Della-Croce, Ghirardi et al. 2004) **BS**	

	1.1 Sequential execution	**1.2 Parallel or intertwined execution**	**2.1.1 Exactly solving relaxed problems**	**2.1.2 Exactly searching large neighbourhoods**	**2.1.3 Exactly solving sub problems**	**2.1.4 Exact algorithms as decoders**	**2.1.5 Exact algorithms for strategic guidance of metaheuristics**	**2.2.1 Metaheuristics for obtaining incumbent solutions and bounds**	**2.2.2 Metaheuristics for column and cut generation**	**2.2.3 Metaheuristics for strategic guidance of exact algorithms**	**2.2.4 Applying the spirit of metaheuristics**
Quadratic Assignment					(Mautor and Michelon 1997) (Mautor and Michelon 2001) (Mautor 2002) **LS, IP**		(Maniezzo 1999) **LR, D, BB, ACO**				
Steiner Tree	(Klau, Ljubíc et al. 2004) **MA, CP**			(Klau, Ljubíc et al. 2004) **MA, CP**							
Traveling Salesman	(Applegate, Bixby et al. 1999) **ILK, BC** (Cook and Seymour 2003) **ILK, DP**			(Cowling and Keuthen 2005) **ILS, DP** (Burke, Cowling et al. 2001) **LS, VNS, DP** (Pesant and Gendreau 1996) (Pesant and Gendreau 1999) **LS, CP** (Congram 2000) **ILS, DP** (Voudouris and Tsang 1999) **GLS, DP**	(Yagiura and Ibaraki 1996) **GA, DP**						
Vehicle Routing	(Ibaraki, Kubo et al. 2001) **ILS, DP**			(Thompson and Orlin 1989) (Thompson and Psaraftis 1993) **DP, VNS**	(Shaw 1998) **BB, TS**						(Danna, Rothberg et al. 2005) **BC, LS**
Packing						(Puchinger, Raidl et al. 2004) **GA, BB** (Imahori, Yagiura et al. 2003) **ILS, DP**	(Dowsland, Herbert et al. 2004) **GA, BB**	(Alvim, Ribeiro et al. 2003) **D, TS**	(Puchinger and Raidl 2004) (Puchinger and Raidl 2004) **BP, GA**		

	1.1 Sequential execution	**1.2 Parallel or intertwined execution**	**2.1.1 Exactly solving relaxed problems**	**2.1.2 Exactly searching large neighbourhoods**	**2.1.3 Exactly solving sub problems**	**2.1.4 Exact algorithms as decoders**	**2.1.5 Exact algorithms for strategic guidance of metaheuristics**	**2.2.1 Metaheuristics for obtaining incumbent solutions and bounds**	**2.2.2 Metaheuristics for column and cut generation**	**2.2.3 Metaheuristics for strategic guidance of exact algorithms**	**2.2.4 Applying the spirit of metaheuristics**
Cutting Stock	(Umetani, Yagiura et al. 2003) **ILS, LR** (Bennell and Dowsland 2001) **TS, LP**										
Lot-sizing						(Staggemeier, Clark et al. 2002) **GA, LP**		(Ozdamar and Barbarosoglu 2000) **LgR, SA**			
Flow-Shop Scheduling	(Nagar, Heragu et al. 1995)**BB, GA**									(Della-Croce, Ghirardi et al. 2004)] **BS**	
Job-Shop Scheduling			(Tamura, Hirahara et al. 1994) **IP, GA, LgR**		(Caseau and Laburthe 1995) **LS, CrP** (Applegate and Cook 1991) **BB, LS** (Adams, Balas et al. 1988) **LS; BB** (Balas and Vazacopoulos 1998) **GLS, BB**		(Lourenço 1995) **ILS, BB** (Lourenço and Zwijnenburg 1996) **ILS, TS, BB**	(Schaal, Fadil et al. 1999) **IPM, C, GA, SA**			(Danna, Rothberg et al. 2005) **BC, LS**
One Machine Scheduling				(Congram, Potts et al. 2002) **ILS, DP** (Lourenço, Martin et al. 2002) **ILS, DP**	(Yagiura and Ibaraki 1996) **GA, DP**						
Parallel Machine Scheduling	(Clements, Crawford et al. 1997) **LS, DW, LR, BB**									(Ghirardi and Potts 2005) **BS**	

	1.1 Sequential execution	**1.2 Parallel or intertwined execution**	**2.1.1 Exactly solving relaxed problems**	**2.1.2 Exactly searching large neighbourhoods**	**2.1.3 Exactly solving sub problems**	**2.1.4 Exact algorithms as decoders**	**2.1.5 Exact algorithms for strategic guidance of metaheuristics**	**2.2.1 Metaheuristics for obtaining incumbent solutions and bounds**	**2.2.2 Metaheuristics for column and cut generation**	**2.2.3 Metaheuristics for strategic guidance of exact algorithms**	**2.2.4 Applying the spirit of metaheuristics**
Knapsack	(Vasquez and Hao 2001) **TS, LR** (Plateau, Tachat et al. 2002) **IPM, PR, SS**		(Chu and Beasley 1998) **GA, LR, SR** (Raidl 1998) **GA, LR, D**								
Generalised Assignment	(Feltl and Raidl 2004) **GA, LP**										(Pigatti, Aragão et al. 2005) **BCP, LS**
Markov Decision Processes	(Lin, Bean et al. 2004) **GA, IP**										
Generalised Schwefel Function					(Cotta and Troya 2003) **GA, BB**						
Optimisation of continuous problems							(Hedar and Fukushima 2004) **TS, NM**				

Heuristics ACO – ant colony optimisation, GA – genetic algorithm, GLS – guided local search, GP – genetic programming, ILK – iterated Lin-Kernighan, ILS – iterated local search, LS – local search, MA – memetic algorithm, PR – path relinking, SA – simulated annealing, SS – scatter search, TS – tabu search, VNS – variable neighbourhood search

Exact methods BB – branch-and-bound, BC – branch-and-cut, BCP – branch-and-cut-and-price, BP – branch-and-price, BS – beam search, C – cuts, CG – column generation, CP – cutting and pricing, CrP – constraint programming, D – duality, DP – dynamic programming, DW – Dantzig Wolf, IP – integer programming, IPM – interior point method, LP – linear programming, LgR – lagrangean relaxation, LR – linear relaxation, NM – Nelder-Mead method, SR – surrogate relaxation

4. Scheduling Problems

In this chapter, we introduce scheduling problems and study their algebraic structure. We start by presenting in section 4.1 the features which constitute a scheduling problem and the characteristics that define different types of these problems. A scheduling problem is usually mathematically formulated as a disjunctive program, so section 4.2 presents a short introduction to disjunctive programming. When studying the algebraic structure of scheduling problems we focus mainly on two types of problems: the one machine scheduling problem, presented in section 4.3, and the job shop scheduling problem in section 4.4. In this section dedicated to the job shop scheduling problem, we include a subsection in the end where we present a literature review of optimised search heuristics that have been applied to it.

4.1 Introduction to Scheduling Problems

A scheduling problem considers a set of jobs to be processed on a set of machines. A job consists of one or more operations (or activities); each operation is assigned to a machine and it uses a constant processing time (here we deal only with deterministic problems). It is assumed that two consecutive operations of the same job are assigned to different machines and that all the machines are always available to the system. To solve the problem we need to find a sequence, and the correspondent time intervals, for processing the operations in each machine. A solution to the problem is called a schedule. A feasible schedule is one respecting that each machine can only process one operation at a time; different machines can not process the same job simultaneously and also some additional constraints related to the specific problem type. The problem type is characterised by the machine environment, the job characteristics and the objective function to be optimised.

The machine environment can have only one stage, where one job corresponds to only one operation, or it may be multi-stage, and in this case one job corresponds to several operations.

There may be more than one machine in a single stage environment, but then they will all work in parallel having the same function. Parallel machines can be of three different types: identical, uniform or unrelated. In the first case, the processing times of the jobs are independent of the machine; uniform parallel machines are identical except that they have different speeds; for unrelated parallel machines, the processing times of the jobs are dependent on the machine assignment.

Multi-stage machine environments are also designated as shop environments and can be of three different types: flow shop, open shop or job shop. In a flow shop scheme the processing alignment of the operations of a job, passing from one machine to the next, is the same for all the jobs. In an open shop system the ordering through what the jobs move from one machine to another is to be decided when solving the problem and may differ from job to job. In a job shop environment the order of processing the operations within the jobs and its correspondent machines are fixed apriority and are independent from job to job.

Speaking of the job characteristics, if there is only one machine or if they are identical parallel machines, the processing time of a job j is given by p_j; otherwise, p_{ij} is the processing time of job j on machine i (or also p_{ij} is the processing time of operation o_{ij}). The processing times are non-negative integer parameters of the problem. The jobs may all be available at the beginning of the process or they may have release dates (r_j for job j), specifying when a job becomes available to the system. Jobs may also have due dates (d_j for job j), indicating a limited date for their conclusion. There may be dependence relations between jobs and it may be allowed to interrupt the processing of an operation; resuming it at a latter moment.

In off-line scheduling systems, the classical models, all the information of an instance is known apriority. In on-line systems the information on the number of jobs, their release and/or due dates are made available during the course of the scheduling.

According to the optimality criteria, several measures related to the processing times p_j, the due dates d_j, or the weights w_j associated with the jobs j, may be used as the function to be optimised. Given a schedule the following measures may be computed: C_j - the completion time of job j; $F_j = C_j - r_j$ - the flow time of job j; $L_j = C_j - d_j$ - the lateness of job j; $E_j = \max\{d_j - C_j, 0\}$ - the earliness of job j; $T_j = \max\{C_j - d_j, 0\}$ - the tardiness of job j; $U_j = \begin{cases} 1 & \mathit{if}\ C_j > d_j \\ 0 & \mathit{otherwise} \end{cases}$ - the unity penalty of job j. Based on these measures many objective functions may be formulated. The table 4.1 shows some common ones.

Table 4.1 Common Objective Functions for Scheduling Problems

Objective Function	Description
$\min_s C_{\max} = \max_j C_j$	minimisation of the maximum completion time or makespan
$\min_s L_{\max} = \max_j L_j$	minimisation of the maximum lateness
$\min_s E_{\max} = \max_j E_j$	minimisation of the maximum earliness
$\min_s \sum_j (w_j) C_j$	minimisation of the total (weighted) completion time
$\min_s \sum_j (w_j) F_j$	minimisation of the total (weighted) flow time
$\min_s \sum_j (w_j) E_j$	minimisation of the total (weighted) earliness
$\min_s \sum_j (w_j) T_j$	minimisation of the total (weighted) tardiness
$\min_s \sum_j (w_j) U_j$	minimisation of the (weighted) number of late jobs

Graham, Lawer, Lenstra & Rinnooy Kan (Graham, Lawler et al. 1979) presented a three-field description $\alpha | \beta | \gamma$ for scheduling problems where α represents the machine environment, β the job characteristics and γ the optimality criteria. The field α indicates if the system is a one-machine, a parallel machine or a shop environment, and the number of stages and machines. The β field indicates if it is an on-line or off-line system, the existence of release and/or due dates, if pre-emption is or not allowed, if there are dependence relations between jobs and if processing times are unitary or arbitrary. For example $| r_j \; d_j \; pmnt | L_{\max}$ is the representation of a one-machine problem with release dates and due dates where pre-emption is allowed and the objective is to minimise the maximum lateness; $O3 | p_{ij} = 1 | C_{\max}$ is an open shop problem with three machines, all processing times are unitary and the objective is to minimise the makespan; $F2(P4, P3) || \sum C_j$ is a two-stage flow shop to minimise the total completion time, where the two stages have four and three identical parallel machines, respectively.

Many fundamental results on scheduling were stated in (Lenstra, Kan et al. 1977). For further details on scheduling problems and theory please refer to the surveys (Lawler, Lenstra et al. 1993) and (Chen, Potts et al. 1998).

In the following sections of this chapter we will address the study of the algebraic structure of scheduling problems.

4.2 Disjunctive Programming

The creation of a feasible schedule for a scheduling problem involves frequently the decision to allocate job i to be processed before job j, or vice versa, in a machine. Therefore, when formulating a scheduling problem and considering different jobs that must be processed by a common machine, we are frequently faced with constraints of the form $t_j - t_i \geq p_i \vee t_i - t_j \geq p_j$ for every jobs i and j that share a machine, where p_i represents the processing time of job i and t_i is the variable representing the starting instance of the processing of job i. This is a disjunctive constraint. A program containing disjunctive constraints is a disjunctive program and there is a whole area of mathematical programming addressing these programs, the disjunctive programming area.

When characterising the algebraic structure of an optimisation problem P, with a non-convex set of feasible solutions S, which is the case in scheduling problems, we are especially interested in the description of the convex hull of the feasible solutions $Conv(S)$, because optimal solutions are found in the vertices and extreme rays of the convex hull. A convex hull is characterised by its defining facets. (An equality $\pi x = \pi_o$ is a facet of a set T of dimension d when it is verified by exactly d affinely independent points x of T. An inequality $\pi x \geq \pi_o$ valid for all $x \in T$ is said to be a facet defining inequality if $\pi x = \pi_o$ is a facet of T.)

The description of the convex hull of disjunctive programs has been addressed by some authors, including Egon Balas. His article *Disjunctive programming: Properties of the convex hull of feasible points* (Balas 1998) congregates major results on the subject. There we can find the characterisation of the family of all the valid inequalities for a given disjunctive program and he gives necessary and sufficient conditions for an inequality to define a facet of the convex hull of the feasible points. The facets of the convex hull are computed solving a large linear program, with size proportional to the number of disjunctions of the original problem. As the number of disjunctions is often enormous solving this linear program may be impracticable; but, for some special disjunctive programs, it is possible to generate the convex hull by a sequence of “partial” convex hulls of relaxed problems, adding one disjunctive constraint at the time. Even so, the number of the facets of the partial convex hulls may be very large. A practical approach would be to generate only a few facets, if one can have information about which ones are likely to be binding in the region between the relaxed and the integer optimum.

Since scheduling problems are formulated as disjunctive programs, these results may be used to develop valid inequalities and cutting-plane methods to solve them. For instance, we can find valid inequalities to one-machine scheduling problems and query under which conditions these inequalities produce cuts to shop environment problems.

We are especially interested in the study of the algebraic structure of the job shop scheduling problem. As stated before, the knowledge of the properties of the one machine scheduling problem can be very useful to the study of the job shop scheduling problem.

If in a job shop system we relax the constraints which state that a set of operations must be processed by the same machine and that the machine can only process one operation at the time (the capacity constraints), for all the machines except one, we get a one machine scheduling problem. In this sense we say that a job shop problem with m machines has m one machine scheduling subproblems.

In the next section 4.2 we present the study of the one machine scheduling problem. Then in section 4.3 the job shop scheduling problem is studied and we show how the properties derived for the one machine problem are relevant to the job shop.

4.3 The One Machine Scheduling Problem

In the one machine scheduling problem n jobs have to be sequenced on one machine. Each job j is available to the system at instant r_j (release date), its processing takes p_j units of time on the machine, and it stays in the system q_j units of time (queue) after being processed by the machine. The goal is to minimise the maximum completion time of all jobs; the makespan. The one machine problem, represented in the three-field notation as $|r_j|C_{\max}$ is a strong NP–hard problem (Garey and Johnson 1979).

The problem is naturally formulated as a disjunctive program like presented below, where t_j is the variable representing the starting instant of processing job j.

$(OMSP)$

$$\min \max_j (t_j + q_j)$$

$$\begin{array}{llll} s.t. & t_j \geq r_j & j \in N = \{1, \ldots, n\} & (4.1) \\ & t_j - t_i \geq p_i \vee t_i - t_j \geq p_j & i, j \in N, i \neq j & (4.2) \end{array}$$

The one machine problem is usually represented by a disjunctive graph $G_{one} = (N \cup \{0, n+1\}, E, A_0 \cup A_{n+1})$. The set of nodes of graph G_{one} corresponds to the set of jobs N and two other fictitious nodes, the origin 0 representing the beginning of the system, and the sink $n+1$, representing the end of the system. The set of edges in G_{one} is $E=\{(i,j)\,\forall i,j\in N, i\neq j\}$ connecting all pairs of nodes representing jobs; and there are two sets of arcs $A_o=\{(0,j)\,\forall j\in N\}$, the arcs connecting the origin node to the nodes of all the jobs and $A_{n+1}=\{(j,n+1)\,\forall j\in N\}$, the arcs from each one of the nodes of the jobs to the sink node. An edge $(i,j)\in E$ will have weight p_i or p_j, depending on job i being processed before or after job j. Arcs in A_0 have weight r_j since the machine can not start to process job j before r_j. Arcs in A_{n+1} have weight p_j+q_j, since a job j has to spend an amount of time q_j in the system after being processed by the machine for p_j units of time. Finding a solution to the one machine scheduling problem means choosing an orientation for every edge in E, constructing an acyclic directed graph. The subgraph $C=(N,E)$ of G_{one} is a fully connected graph, named a clique, which means that for every possible pair of nodes in N, there is an edge in E connecting them.

As we will see in the following sections, many authors have worked on this problem; as the knowledge of their properties is crucial for addressing more complicated shop environment systems.

4.3.1 Algebraic Structure of the One Machine Problem

Based on the theory on disjunctive programming, Balas (Balas 1985) derives results on the characterisation of the facets of convex hull of the feasible solutions to the one machine scheduling problem. The facets are defined by valid inequalities in the variables t_j, $j\in N$, involving the parameters representing the release dates of the jobs r_j, the processing times p_j and the queue of the jobs q_j. Balas presents facet defining inequalities with one, two and three nonzero coefficients on the variables t_j.

Inequalities with one nonzero coefficient These inequalities with one nonzero coefficient are the ones related to the release date of the jobs: $t_j \geq r_j \quad \forall j \in N$. These are obvious inequalities present in the formulation.

Inequalities with two nonzero coefficients The facet defining inequalities having exactly two coefficients different from zero are of the form

$$\left(p_i + r_i - r_j\right)t_i + \left(p_j + r_j - r_i\right)t_j \geq p_i p_j + r_i p_j + r_j p_i \qquad (4.3)$$

for every pair of distinct jobs i and j of N such that $r_j < r_i + p_i \wedge r_i < r_j + p_j$. This condition implies that the coefficients of the variables t_i and t_j are non negative. To verify that these inequalities are valid please not that: if i is scheduled before j then the smallest possible values for t_i and t_j will be r_i and $r_i + p_i$ respectively; if j is scheduled before i then the smallest possible values for t_j and t_i will be r_j and $r_j + p_j$. Inequalities (4.3) are derived from the solution of the system solved to find the vertices of the polyhedron correspondent to the subproblem considering only jobs i and j.

Inequalities with three nonzero coefficients To present the facet defining inequalities with three nonzero coefficients we need first to introduce the following matrix V with three columns, each corresponding to a job (i, j and k), and each row corresponding to a permutation of the three jobs. V will have six rows. Let us assume the rows are ordered this way: row 1 corresponds to permutation (i, j, k), row 2 to (j, k, i); row 3 to (k, i, j); row 4 to (i, k, j); row 5 to (j, i, k) and row 6 to (k, j, i). Each element v_{rc} of matrix V will be the earliest possible date to start processing job c in the permutation correspondent to row r.

$$V = \begin{bmatrix} r_i & r_i + p_i & r_i + p_i + p_j \\ r_j + p_j + p_k & r_j & r_j + p_j \\ r_k + p_k & r_k + p_k + p_i & r_k \\ r_i & r_i + p_i + p_k & r_i + p_i \\ r_j + p_j & r_j & r_j + p_j + p_i \\ r_k + p_k + p_j & r_k + p_k & r_k \end{bmatrix}$$

Let $V_{\{a,b,c\}}$ denote a 3×3 matrix corresponding to rows a,b,c of V and let $V^{j}_{\{a,b,c\}}$ be the matrix obtained from $V_{\{a,b,c\}}$ by substituting 1 for every entry of column j.

The facet defining inequalities with three nonzero coefficients are of the form

$$\alpha_i t_i + \alpha_j t_j + \alpha_k t_k \geq 1 \text{ with } \alpha_l = \frac{\det\left(V^{l}_{\{a,b,c\}}\right)}{\det\left(V_{\{a,b,c\}}\right)} \text{ for } l = i, j, k \qquad (4.4)$$

for some triplets of rows (a,b,c). These inequalities are the generalisation of the ones with two nonzero coefficients. They are the solution to the system that defines the vertices of the polyhedron correspondent to the subproblem considering only jobs i, j and k. The expression defining the α is just the Cramer Rule to solve systems of linear equations.

Balas proves that there are at most four distinct inequalities (4.4) that define facets of the convex hull of the feasible solutions to the one machine problem.

For further details on the inequalities of Balas please refer to his paper (Balas 1985).

Another author that has studied the algebraic structure of the one machine scheduling problem and proved several properties is Jacques Carlier. In his work (Carlier 1982) he developed a branch-and-bound algorithm to solve the one machine problem, building an initial feasible solution with the priory rule algorithm of Schrage (Schrage 1970) and performing branching based on a proposition presented next. The algorithm of Schrage schedules available jobs giving priority to the one job j with bigger queue q_j. It builds the list schedule associated with the most work remaining priority dispatching rule of Jackson (Jackson 1955). Carlier proves that $h(J) = \min_{j\in J} r_j + \sum_{j\in J} p_j + \min_{j\in J} q_j$ is a lower bound on the optimal makespan for every subset of jobs $J \subseteq N$. He also shows that given a Jackson's scheduled with makespan mk one of two situations occurs. Or the schedule is optimal and there is a set of jobs J such that $h(J) = mk$. Either the schedule is not optimal and then there are a critical set of jobs J and a critical job c such that $h(J) > mk - p_c$. This means that in this case the distance to the optimum from the Jackson's schedule is less than p_c and implies that in an optimal schedule, either job c is processed before all the jobs in set J or job c is processed after all the jobs in set J. This property is used to define the branching scheme of the branch-and-bound algorithm.

Carlier also proves some other propositions that given an upper bound (UB) on the makespan enable the determination of the position of a job in the processing sequence of an optimal schedule.

Given two jobs i and j if

$$r_i + p_i + p_j + q_j > UB \qquad (4.5)$$

then job j is schedule before job i on every optimal schedule.

Given, a subset of jobs $J \subseteq N$ and a job $k \in J$; if

$$\min_{j \in J \setminus \{k\}} r_j + \sum_{j \in J} p_j + \min_{j \in J \setminus \{k\}} q_j > UB \qquad (4.6)$$

then on an optimal schedule k is sequenced either before or after all other jobs in set J; if

$$r_k + \sum_{j \in J} p_j + \min_{j \in J \setminus \{k\}} q_j > UB \qquad (4.7)$$

then on an optimal schedule k will not be the first job of set J to be processed; if

$$\min_{j \in J \setminus \{k\}} r_j + \sum_{j \in J} p_j + q_k > UB \qquad (4.8)$$

then on an optimal schedule k will not be the last job of set J to be processed; if both conditions (4.6) and (4.7) are verified, then on an optimal schedule k will be the last job of set J in the schedule and k is called the output of J; if both conditions (4.6) and (4.8) are verified, then on an optimal schedule k will be the first job of set J in the schedule and k is called the input of J.

4.4 The Job Shop Scheduling Problem

The job shop scheduling problem has been known to the operations research community since the early 50's (Jain and Meeran 1999). It is considered a particularly hard combinatorial optimisation problem of the NP-hard class (Garey and Johnson 1979) and it has numerous practical applications; which makes it an excellent test problem for the quality of new scheduling algorithms. These are main reasons for the vast bibliography on both exact and heuristic procedures applied to this scheduling problem. The paper *Deterministic job-shop scheduling: past, present and future* (Jain and Meeran 1999) includes an exhaustive survey not only of the evolution of the definition of the problem, but also of all the techniques applied to it: enumerative methods like branch-and-bound; constructive methods like priority dispatching rules; iterative methods like ant optimisation; local search methods and metaheuristics like GRASP, simulated annealing, genetic algorithms, large step optimisation or tabu search.

In the job shop scheduling problem each job is defined by an ordered set of operations and each operation is assigned to a machine with a predefined constant uninterrupted processing time. The order of the operations within the jobs and its correspondent machines are fixed apriority and independent from job to job. To solve the problem we need to find a sequence of operations in each machine respecting precedence constraints of operations of a job; it is assumed that two consecutive operations of the same job are assigned to different machines, each machine can only process one operation at a time and that different machines can not process the same job simultaneously. We will adopt the maximum of the completion time of all jobs – the makespan – as the objective function. Using the three fields notation of (Graham, Lawler et al. 1979) the job shop scheduling problem is represented by $Jm \| C_{\max}$.

The Table 4.2 presents an instance of the job shop scheduling problem with 4 jobs and 3 machines. Jobs 1 and 4 must be processed first by machine 1, then by machine 2 and finally by machine 3. The processing sequence for jobs 2 and 3 is first machine 1, then machine 3 and machine 2 at the end. The operations are numbered from 1 to

12, from job 1 to job 4. The processing times of each operation on the correspondent machine are given in the row named proc. times.

Table 4.2 An Instance for the Job Shop Scheduling Problem

	Job 1	Job 2	Job 3	Job 4
operations	$1 \rightarrow 2 \rightarrow 3$	$4 \rightarrow 5 \rightarrow 6$	$7 \rightarrow 8 \rightarrow 9$	$10 \rightarrow 11 \rightarrow 12$
machines	$1 \rightarrow 2 \rightarrow 3$	$1 \rightarrow 3 \rightarrow 2$	$1 \rightarrow 3 \rightarrow 2$	$1 \rightarrow 2 \rightarrow 3$
proc. times	1 1 2	4 2 2	1 1 2	4 2 2

The Fig. 4.2 shows the Gant Chart of a feasible solution of this instance, with makespan 13.

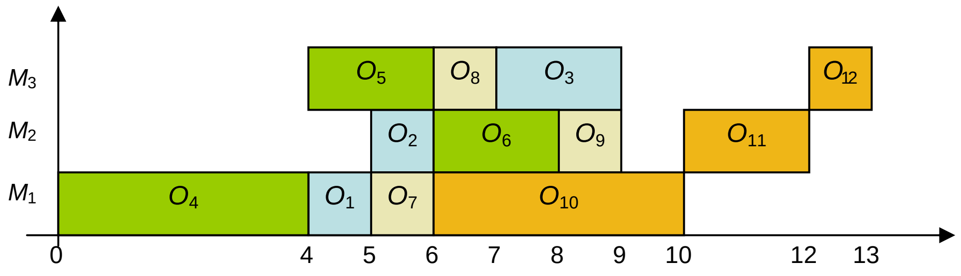


Fig. 4.1 Gant Chart of a Feasible Solution for Instance of Table 4.2

Formally let $O = \{0, \ldots, o+1\}$ be the set of operations with 0 and $o+1$ dummy operations representing the start and end of all jobs, respectively. Let M be the set of machines, A the set of pairs of consecutive operations of each job and E_k the set of all possible pairs of operations processed by machine k, with $k \in M$. We define $p_i > 0$ as the constant processing time of operation i and t_i is the variable representing the starting instant of operation i. The following mathematical formulation for the job shop scheduling problem is widely used by researchers:

$(JSSP)$

$$\min t_{o+1}$$

$$\begin{array}{llll} s.t. & t_j - t_i \geq p_i & (i,j) \in A & (4.9) \\ & t_i \geq 0 & i \in O & (4.10) \\ & t_j - t_i \geq p_i \vee t_i - t_j \geq p_j & (i,j) \in E_k, k \in M & (4.11) \end{array}$$

The constraints in (4.9) state the precedence relations of operations within jobs and also that no two operations of the same job can be processed simultaneously (because $p_i > 0$). Expressions (4.11) are named “capacity constraints” and assure there are no overlaps of operations at the machines.

A feasible solution for the problem is a schedule of operations respecting all these constraints.

The job shop scheduling problem is usually represented by a disjunctive graph $G = (O, A, E)$ (Roy and Sussman 1964). Where O is the node set, corresponding to the set of operations. A is the set of arcs between consecutive operations of the same job, and E is the set of edges between operations processed by the same machine. Each node i has weight p_i, with $p_0 = p_{o+1} = 0$. There is a subset of nodes O_k and a subset of edges E_k for each machine that together form the disjunctive clique $C_k = (O_k, E_k)$ of graph G. For every node j of $O/\{0, o+1\}$ there are unique nodes i and l such that arcs (i, j) and (j, l) are elements of A. Node i is called the job predecessor of node j - $jp(j)$ and l is the job successor of j - $js(j)$. Fig. 4.2 shows the disjunctive graph of the instance in Table 4.2.

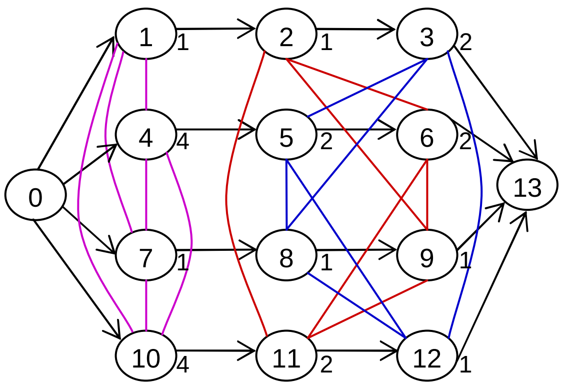


Fig. 4.2 Disjunctive Graph of the Instance in Table 4.2

Finding a solution to the job shop scheduling problem means replacing every edge of the respective graph with a directed arc, constructing an acyclic directed graph $D_S = (O, A \cup S)$. Graph $D = (O, A)$ is obtained from G removing all edges and $S = \bigcup_k S_k$ corresponds to an acyclic union of sequences of operations for each machine k (this implies that a solution can be built sequencing one machine at a time).

The optimal solution is the one represented by the graph D_S having the critical path from 0 to $o+1$ with the smallest length.

For any given solution, the operation processed immediately before operation i in the same machine is called the machine predecessor of i - $mp(i)$; analogously $ms(i)$ is the operation that immediately succeeds i at the same machine. Figure 4.3 shows the directed graph representing the solution for the instance of Table 4.2, shown earlier on a Gant chart in Fig. 4.1. The critical path is evidenced with thicker arrows.

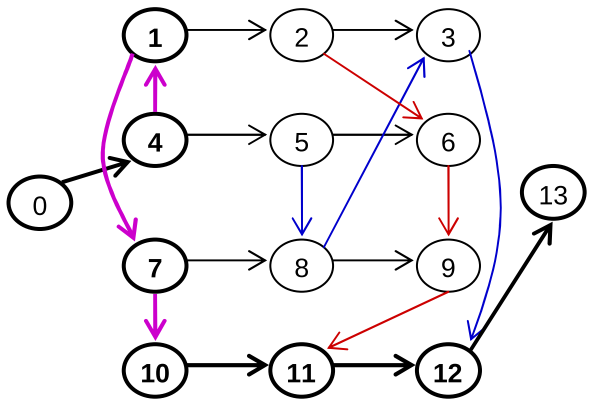


Fig. 4.3 Disjunctive Graph of the Solution in Fig. 4.1

4.4.1 Algebraic Structure of the Job Shop Scheduling Problem

The work *On the Facial Structure of Scheduling Polyhedra* (Balas 1985) shows that, under some conditions, the facet defining inequalities derived for the one machine problem also define the facets of the convex hull of the feasible solutions to the job shop scheduling problem. For example, the inequality $t_i \geq r_i \quad \forall i \in O$ defines a facet for the job shop problem whenever i is the first operation of the job. In the job shop problem the parameters r_i and q_i (present in the formulation of the one machine scheduling problem) are computed for each one machine subproblem, as lengths of paths in the disjunctive graph passing only through nodes of operations that belong to the same job. The r_i is the length of the path from node 0 to the node of operation i; the q_i is the length of the path from the node of operation i to the end node, without the processing time of operation i. In the job shop scheduling the notation r_i is changed to e_i - representing the earliest possible time for starting processing operation i. Analogously, q_i is replaced by f_i.

Let i be an operation processed on machine k; $jp_l(i)$ an operation of the same job as operation i, processed before i on machine l; $L_A(jp_l(i), i)$ the length of the path between $jp_l(i)$ and i, through the arcs in A (i.e., the path on the job between the two operations) and let $L_{E_k}(i, h)$ be the length of the path between two operations i and h (processed by the same machine k) through the edges in E_k.

A clique $C_k = (O_k, E_k)$ of graph $G = (O, A, E)$ is called a dominant clique if the condition $L_{E_l}(jp_l(i), jp_l(h)) + L_A(jp_l(h), h) < L_A(jp_l(i), i) + L_{E_k}(i, h)$ is verified for every two operations i and h (processed by machine k) that have job predecessors $jp_l(i)$ and $jp_l(h)$ processed by a same machine l (i.e. the jobs to which operations i and h belong are both processed by a common machine l before being processed by the common machine k).

Balas proves that the facet defining inequalities for the one machine problem also define facets for the job shop problem, whenever the clique of the machine is a dominant clique.

Similarly, all the inequalities derived by Carlier for the one machine problem are valid inequalities to the job shop scheduling problem (Carlier and Pinson 1989), (Carlier and Pinson 1994).

Before ending this section on the job shop scheduling problem and before following to the next chapters where we introduce the proposed new optimised search heuristics (illustrating them with and application to the job shop problem); we present here a literature review of optimised search heuristics applied to the job shop problem.

4.4.2 Solving The Job Shop Scheduling Problem with OSHs

In the literature we can find a few works applying Optimised Search Heuristics (OSH) to the job-shop scheduling problem (Fernandes and Lourenço 2007b).

Chen, Talukdar and Sadeh (Chen, Talukdar et al. 1993) and Denzinger and Offermann (Denzinger and Offermann 1999) design parallel algorithms that use asynchronous agents information to build solutions; some of these agents are genetic algorithms, others are branch-and-bound algorithms.

Tamura, Hirahara, Hatono and Umano (Tamura, Hirahara et al. 1994) design a genetic algorithm where the fitness of each individual, whose chromosomes represent each variable of the integer programming formulation, is the bound obtained solving lagrangean relaxations.

The works of Adams, Balas and Zawack (Adams, Balas et al. 1988), Applegate and Cook (Applegate and Cook 1991), Caseau and Laburthe (Caseau and Laburthe 1995), Balas and Vazacopoulos (Balas and Vazacopoulos 1998) and Pezzella and Merelli (Pezzella and Merelli 2000) all use an exact algorithm to solve a sub problem within a local search heuristic for the job-shop scheduling. Caseau and Laburthe (Caseau and Laburthe 1995) build a local search where the neighbourhood structure is defined by a subproblem that is exactly solved using constraint programming. Applegate and Cook (Applegate and Cook 1991) develop the shuffle heuristic. At each step of the local search the processing orders of the jobs on a small number of machines is fixed, and a branch-and-bound algorithm completes the schedule. The shifting bottleneck heuristic, due to Adams, Balas and Zawack (Adams, Balas et al. 1988), is an iterated local search with a construction heuristic that uses a branch-and-bound to solve the subproblems of one machine with release and due dates. Balas and Vazacopoulos (Balas and Vazacopoulos 1998) work with the shifting bottleneck heuristic and design a guided local search, over a tree search structure, that reconstructs partially destroyed solutions. The procedure of Pezzella and Merelli (Pezzella and Merelli 2000) is a tabu search that uses a branch-and-bound to solve one-machine subproblems; both at the construction of the initial solution and at a re-optimisation phase of the algorithm.

Lourenço (Lourenço 1995) and Lourenço and Zwijnenburg (Lourenço and Zwijnenburg 1996) use branch-and-bound algorithms to strategically guide an iterated local search and a tabu search algorithm. The diversification of the search is achieved by applying a branch-and-bound method to solve a one-machine scheduling subproblem obtained from the incumbent solution.

In the work of Schaal, Fadil, Silti and Tolla (Schaal, Fadil et al. 1999) an interior point method generates initial solutions of the linear relaxation. A genetic algorithm finds integer solutions. A cut is generated based on the integer solutions found and the interior point method is applied again to diversify the search. This procedure is defined for the generalised job-shop problem.

The interesting work of Danna, Rothberg and Le Pape (Danna, Rothberg et al. 2005) "applies the spirit of metaheuristics" in an exact algorithm. Within each node of a branch-and-cut tree, the solution of the linear relaxation is used to define the neighbourhood of the current best feasible solution. The local search consists in solving the restricted MIP problem defined by the neighbourhood.

We are especially interested in combinations of exact and heuristic methods where the exact procedures can be used to strategically guide the heuristic ones.

New Optimised Search Heuristics Proposed We propose two new optimised search heuristics and present both with an application to the job-shop scheduling problem. The first one GRASP_B&B combines a Branch-and-Bound algorithm with a GRASP procedure. The second Tabu_VVI uses the verification of Violated Valid Inequalities as a diversification strategy for a Tabu Search method. The new method GRASP_B&B is used within Tabu_VVI to build feasible solutions.

5. An OSH Combining GRASP with Branch-and-Bound

This chapter presents an algorithm for the job shop scheduling problem - the GRASP_B&B algorithm - that combines a heuristic local search procedure, GRASP, with an exact method of integer programming, branch-and-bound.

As stated earlier in section 2.2.1 GRASP (Feo and Resende 1995) is an iterative process where each iteration consists of two steps: a randomised building step of a greedy nature and a local search step. The proposed procedure GRASP_B&B starts with an empty solution and builds a complete solution sequencing one machine at a time. The branch-and-bound algorithm is used in the building step of a GRASP procedure to solve the one machine scheduling subproblems.

On the following sections each phase of GRASP_B&B, the building step and the local search phase, is described in detail. In section 5.4 the proposed method is compared with similar approaches and, as we will see, it leads to better results in terms of solution quality and computing times.

5.1 Building Step

At the building phase of a GRASP algorithm a feasible solution is constructed by joining one element at a time. Each element is evaluated by a heuristic function and incorporated (or not) in a restricted candidate list (RCL) according to its evaluation. Then the element to join the solution is chosen randomly from the RCL. We define the sequence of operations at each machine as the elements to join the solution, and the makespan of the one machine problem ($\max(t_i + p_i), i \in O_k, k \in M$) as the greedy function to evaluate them. In order to build the restricted candidate list we find the optimal solution for the one machine problems of all machines not yet scheduled ($K \subseteq M$ is the set of unscheduled machines), and identify the best $(\underline{f})$ and worst $(\overline{f})$

makespans. A machine k is included in the RCL if $f(x_k) \geq \overline{f} - \alpha\left(\overline{f} - \underline{f}\right)$, where $f(x_k)$ is the makespan of machine k and α is a uniform random number in $(0,1)$.

We explain how the one machine problems are defined and solved in the next section 5.1.1.

After solving the subproblems of all unscheduled machines and building the restricted candidate list, a semigreedy procedure chooses one machine to enter the solution in a semi-greedy randomised way.

```
Algorithm SemiGreedy $(K)$
(1) $\alpha := Random(0,1)$
(2) $\overline{f} := \max\{f(x_k), k \in K\}$
(3) $\underline{f} := \min\{f(x_k), k \in K\}$
(4) $RCL = \{\ \}$
(5) foreach $k \in K$
(6)         if $f(x_k) \geq \overline{f} - \alpha\left(\overline{f} - \underline{f}\right)$
(7)                 $RCL := RCL \cup \{k\}$
(8) return $RandomChoice(RCL)$
```

Fig. 5.1 Outline of Procedure SemiGreedy

This semi-greedy randomised procedure is biased towards the machine with the higher makespan, the bottleneck machine, in the sense that machines with low values of makespan have less probability of being included in the restricted candidate list.

The next chapter presents the definition of the one machine scheduling subproblems and the algorithm to solve them.

5.1.1 One Machine Problem

Defining the One Machine Subproblems Given a job shop scheduling problem and its representation on a disjunctive graph $G = (O, A, E)$, the one machine subproblems for each machine $k \in M$ are obtained considering only the nodes of the operations processed on k and the set of edges between them (E_k). The subproblems are represented by the clique $C_k = (O_k, E_k)$ with the objective function of minimising the completion times of all operations and considering three parameters associated with each operation i in C_k. One parameter is the processing time p_i. Let $K \subseteq M$ be the set of unscheduled machines. The two other measures associated with each operation of the one machine subproblems are computed from the graph $G_{M-K} = (O, A, S_{M-K})$, obtained from G replacing the edges of scheduled machines by the arcs corresponding to the sequence of processing their operations and removing the edges corresponding to all unscheduled machines. The two measures are the release dates r_i computed as the length of the longest path in G_{M-K} from the source node to the node of operation i and the queue values q_i computed as the length of the longest path in G_{M-K} from the node of operation i to the end node (minus the processing time of operation i). The release date of an operation i represents the time that the job to which operation i belongs has been in the system before the processing of the operation starts. The queue value, also called tail, represents the time that the job to which operation i belongs stays in the system after the processing of the operation ends. At the first iteration of the algorithm GRASP_B&B, release dates and tails are computed considering the graph $D = (O, A)$.

Solving the One Machine Problems To solve to optimality the one machine scheduling problems we use the branch-and-bound algorithm of Carlier (Carlier 1982) described earlier on section 4.3.1.

At each node of the branch-and-bound tree the upper bound is computed using the algorithm of Schrage (Schrage 1970). This algorithm gives priority to higher values of the tails (q_i) when scheduling released jobs. We break ties by preferring larger processing times.

The computation of the lower bound, computed like in (Carlier 1982) is based on the critical path with more jobs of the solution found by the algorithm of Schrage (Schrage 1970) and on a critical job, as shown in chapter 4. The value of the solution with pre-emption is used to strengthen this lower bound. We introduce a slight modification, forcing the lower bound of a node never to be smaller than the one of its father in the tree. (The makespans of the one machine scheduling subproblems are lower bounds to the makespan of the job shop scheduling problem.)

The algorithm of Carlier (Carlier 1982) uses some proven properties of the one machine scheduling problem, showed earlier in chapter 4, to define the branching strategy, and also to reduce the number of inspected nodes of the branch-and-bound tree.

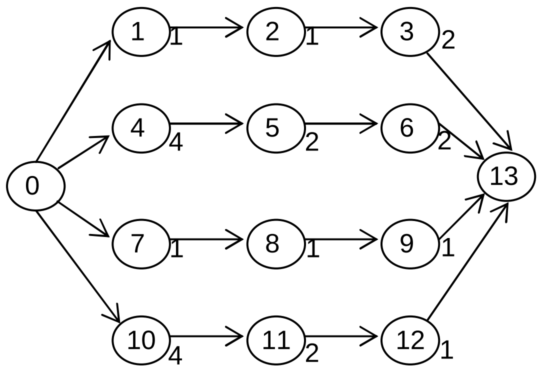
Incorporating the One Machine Solution Incorporating a new machine in the solution means adding to the set of arcs S_{M-K} of graph $G_{M-K} = (O, A, S_{M-K})$ the set of arcs corresponding to the optimal sequence for processing operations on machine k - S_k. In terms of the mathematical formulation, this means choosing one of the inequalities of the disjunctive constraints (4.11) corresponding to the machine k.

When a new machine is added to a partial solution the makespan of the solution and the release dates and tails of unscheduled operations are updated. In the proposed procedure GRASP_B&B these updates are accomplished using an algorithm similar to the one used by Taillard (Taillard 1994). This algorithm has a module that updates the release dates by building and maintaining a list of the operations which either do not have operations that precede them (both in the job and in the machine), or have the predecessors with the release dates already updated. The module is repeated with a modification to update the tails of the operations, building a list of operations without successors or with successors with the tails already updated. Finally for each operation the updated values of release dates and tails are added to the processing time and the makespan of the partial solution is computed.

Before proceeding to the section where the local search step of the algorithm GRASP_B&B is described, let us illustrate the building step with an example.

Illustrating the Building Step To exemplify how the building step of the procedure GRASP_B&B works let us illustrate one iteration considering the disjunctive graph of the instance of Table 4.2.

Deleting all the edges connecting operations that share a same machine in the graph of Fig. 4.2 we get the graph shown in Fig. 5.2. Computing the one machine problems for each of the machines, we get the problems present bellow the graph.



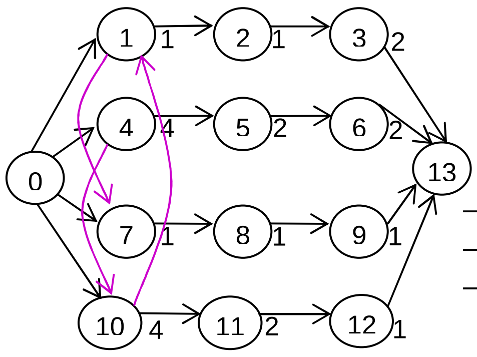
M_1	O_1	O_4	O_7	O_{10}
r_i	0	0	0	0
p_i	1	4	1	4
q_i	3	4	2	3

M_2	O_2	O_6	O_9	O_{11}
r_i	1	6	2	4
p_i	1	2	1	2
q_i	2	0	0	1

M_3	O_3	O_5	O_8	O_{12}
r_i	2	4	1	6
p_i	2	2	1	1
q_i	0	2	1	0

Fig. 5.2 Graph for instance of Table 4.2 without all edges and the respective one machine subproblems

The branch-and-bound algorithm finds the optimal solution $O_4 \to O_{10} \to O_1 \to O_7$ with makespan 12 for machine M_1, the optimal solution $O_2 \to O_9 \to O_{11} \to O_6$ with makespan 8 for machine M_2 and the optimal solution $O_8 \to O_3 \to O_5 \to O_{12}$ also with makespan 8 for machine M_3. Let us suppose the semigreedy procedure chooses machine 1 to be included in the solution. Then the partial solution would be the one represented in the graph of Fig. 5.3 and the new one machine subproblems for machines 2 and 3 the ones presented in the tables next to it.



M_2	O_2	O_6	O_9	O_{11}
r_i	9	6	11	8
p_i	1	2	1	2
q_i	2	0	0	1

M_3	O_3	O_5	O_8	O_{12}
r_i	10	4	10	10
p_i	2	2	1	1
q_i	0	2	1	0

Fig. 5.3 Graph of a partial solution to instance of Table 4.2, and the respective one machine problems for machines 2 and 3

5.2 The Local Search Module

In the algorithm GRASP_B&B, when the sequence of one machine is added to the solution in the building step, and if the solution already has more than one machine scheduled, a local search procedure is executed to get a local optimal (partial) solution. In this section we describe the local search module of the algorithm.

In order to build a local search algorithm we need to design a neighbourhood structure (defined by moves between solutions), the way to inspect the neighbourhood of a given solution, and a procedure to evaluate the quality of each solution. It is said that a solution B is a neighbour of a solution A if we can achieve B by performing a neighbourhood defining move in A.

We use a neighbourhood structure very similar to the NB neighbourhood of Dell’Amico and Trubian (Dell'Amico and Trubian 1993) and the one of Balas and Vazacopoulos (Balas and Vazacopoulos 1998). To describe the moves that define this neighbourhood we use the notion of blocks of critical operations. A block of critical operations is a maximal ordered set of consecutive operations of a critical path (in the disjunctive graph that represents the solution), sharing the same machine. Let $L(i, j)$ denote the length of the critical path from node i to node j. Borrowing the nomination of Balas and Vazacopoulos (Balas and Vazacopoulos 1998) we speak of forward and backward moves over forward and backward critical pairs of operations.

Two operations u and v form a forward critical pair (u,v) if:

a) they both belong to the same block;

b) v is the last operation of the block;

c) operation $js(v)$ also belongs to the same critical path;

d) the length of the critical path from v to $o+1$ is not less than the length of the critical path from $js(u)$ to $o+1$ ($L(v,o+1) \geq L(js(u),o+1)$).

Two operations u and v form a backward critical pair (u,v) if:

a) they both belong to the same block;

b) u is the first operation of the block;

c) operation $jp(u)$ also belongs to the same critical path;

d) the length of the critical path from 0 to u, including the processing time of u, is not less than the length of the critical path from 0 to $jp(v)$, including the processing time of $jp(v)$ ($L(0,u)+p_u \geq L(0, jp(v))+p_{jp(v)}$).

Conditions d) are included to guarantee that all moves lead to feasible solutions (Balas and Vazacopoulos 1998).

A forward move is executed by moving operation u to be processed immediately after operation v. A backward move is executed by moving operation v to be processed immediately before operation u.

For illustration purpose let us consider the feasible solution to the instance of Table 4.2 with makespan 14 represented by the graph in Fig. 5.4

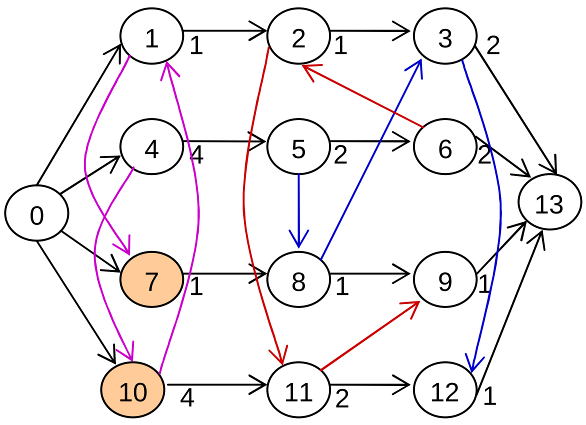


Fig. 5.4 Graph of a feasible solution with makespan 14 of instance in Table 4.2

The pair of operations $(10,7)$ is a forward critical pair since: a) they both belong to the critical path $O_4 \rightarrow O_{10} \rightarrow O_1 \rightarrow O_7 \rightarrow O_8 \rightarrow O_3 \rightarrow O_{12}$; b) operation 7 is the last operation of that block of critical operations; c) operation 8 (the job successor of operation 7) also belongs to the critical path and d) the length of the critical path from operation 7 to the end, which is 5, is not less than the length of the critical path from operation 11 (the job successor of operation 10) to the end, which is 3. The forward move would be to process operation 10 immediately after operation 7, generating the solution with makespan 13 represented by the graph of Fig. 5.5.

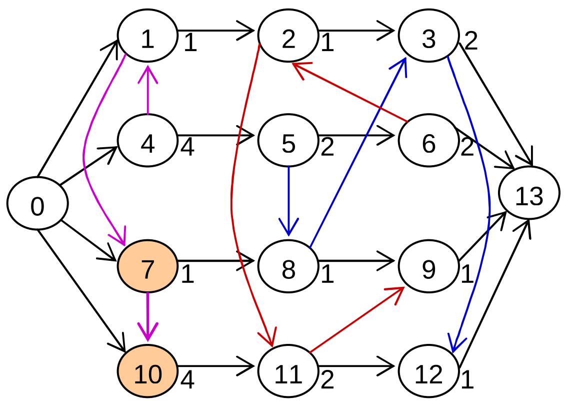


Fig 5.5 Graph obtained from the graph of Fig. 5.4 by the forward move on operations (7,10)

When inspecting the neighbourhood ($N(x, M_k)$) of a given solution x with M_k machines already scheduled, we stop whenever we find a neighbour with a best evaluation value than the makespan of x.

To evaluate the quality of a neighbour of a solution x, produced by a move over a critical pair (u,v), we need only to compute the length of all the longest paths through the operations that were between u and v in the critical path of solution x. This evaluation is computed using the same algorithm as Balas and Vazacopoulos (Balas and Vazacopoulos 1998), which is a variation of the one of Taillard (Taillard 1994) for a subset of arcs.

```
Algorithm LocalSearch $(x, f(x), M_0)$
(1)    $s := neighbour(x, f(x), M_0)$
(2)    while $s \neq x$
(3)        $x := s$
(4)        $s := neighbour(x, f(x), M_0)$
(5)    return( $s$ )
```

```
Algorithm Neighbour $(x, f(x), M_0)$
(1)    foreach $s \in N(x, M_0)$
(2)        $f(s) := evaluation\ (move(x \to s))$
(3)        if ( $f(s) < f(x)$ )
(4)            return( $s$ )
(5)    return( $x$ )
```

Fig. 5.6 Pseudo-code of Module Local Search

The Fig. 5.6 presents the pseudo-code of the module Local Search of the method GRASP_B&B.

The next section presents the whole metaheuristic GRASP_B&B.

5.3 GRASP_B&B

Let $runs$ be the total number of runs, M the set of machines of the instance and $f(x)$ the makespan of a solution x. The procedure GRASP_B&B can be generally described by the pseudo-code in the following Fig. 5.7.

```
Algorithm GRASP_B&B $(runs)$
(1)   $M := \{1, \cdots, m\}$
(2)   for $r = 1$ to $runs$
(3)       $x := \{\ \}$
(4)       $K := M$
(5)       while $K \neq \{\ \}$
(6)           foreach $k \in K$
(7)               $x_k := CARLIER\_B \& B(k)$
(8)           $k^* := SEMIGREEDY(K)$
(9)           $x := x \cup x_{k^*}$
(10)          $f(x) := TAILLARD(x)$
(11)          $K := K \setminus \{k^*\}$
(12)          if $|K| < |M| - 1$
(13)              $x := LOCALSEARCH(x, M \setminus K)$
(14)      if ( $x^*$ not initialised or $f(x) < f^*$)
(15)          $x^* := x$
(16)          $f^* := f(x)$
(17)  return( $x^*$)
```

Fig. 5.7 Outline of Procedure GRASP_B&B

The number of iterations of the method corresponds to the number of machines in the problem, and the GRASP_B&B ends with a complete solution. At each iteration the method defines and solves the one machine subproblems for all unscheduled machines using algorithm Carlier_B&B described in section 5.1.1. Procedure SemiGreedy chooses the one machine solution to add to the partial solution of the job shop and procedure Taillard, described in section 5.1.1, computes its makespan.

Procedure LocalSearch is applied to the current partial solution, if there are more than one scheduled machines, and this concludes the iteration. The all process is repeated for several runs, keeping track of the best complete solution found.

This metaheuristic has only one parameter to be defined: the number of runs to perform (line (2)). The step of line (8) is the only one using randomness. When applied to an instance with m machines, in each run of the metaheuristic, the branch-and-bound algorithm is called $m \times (m+1)/2$ times (line (7)); the local search is executed $m-1$ times (lines (12) and (13)); the procedure semigreedy (line (8)) and the algorithm of Taillard (line (10)) are executed m times.

5.4 Computational Experiment

We have tested the algorithm GRASP_B&B on the benchmark instances abz5-9 (Adams, Balas et al. 1988), ft6, ft10, ft20 (Fisher and Thompson 1963), la01-40 (Lawrence 1984), orb01-10 (Applegate and Cook 1991), swv01-20 (Storer, Wu et al. 1992), ta01-70 (Taillard 1993) and yn1-4 (Yamada and Nakano 1992).

When applying the branch-and-bound algorithm, used in the building step of GRASP_B&B to solve the one machine scheduling problems, to instances of the job shop problem with 50 or more jobs, we observed that a lot of time was spent inspecting nodes of the tree, after having already found the optimal solution. So we introduced a condition restricting the number of nodes of the tree: the algorithm is stopped if there have been inspected more then n^3 nodes after the last reduction of the difference between the upper and lower bound of the tree (n is the number of jobs).

In Annex B the reader can find tables where we present computational results having the following structure: in each line it is presented the name of the instance, the number of jobs and the number of machines of the instance $(n \times m)$, and the best lower and upper bound values (LB, UB) of the makespan. If the lower bound is omitted, the upper bound is optimal. We gathered the values of these bounds from the papers (Jain and Meeran 1999), (Nowicki and Smutniki 1996), (Nowicki and Smutnicki 2002) and (Nowicki and Smutnicki 2005).

The algorithm has been run 100 times for each instance on a Pentium 4 CPU 2.80 GHz and coded in C. The tables also present some statistical values concerning the makespan of the solutions found in the 100 runs, as well as the total time of all runs $(ttime)$ and the time to the best solution found $(btime)$, in seconds. The statistics of the makespan computed over the 100 runs are the minimum $(\min)$, the first quartile (Q_1), the median (Q_2), the third quartile (Q_3) and the maximum $(\max)$. We chose this measures because they allow us to see how disperse are the values obtained by different runs, which give us an idea of the robustness of the algorithm. Within brackets, next to each value, is the correspondent percentage of relative error to the upper bound.

$$RE_{UB}(x) = 100\% \times \frac{f(x) - UB}{UB}$$

Whenever the values are not worse than the best known upper bound, we present them in bold. Although this is a very simple (and fast) algorithm, it happens in 23 of the 152 instances used in this study.

The information of these tables can be visualised using boxplots. They show that the quality achieved is more dependent on the ratio n/m than on the absolute numbers of jobs and machines. There is no big dispersion of the solution values achieved by the algorithm in the 100 runs executed, so we say the algorithm is steady. The number of times the algorithm achieves the best values reported is high enough, so these values are not considered outliers of the distribution of the results. On the other end, the worse values occur very seldom and are outliers for the majority of the instances.

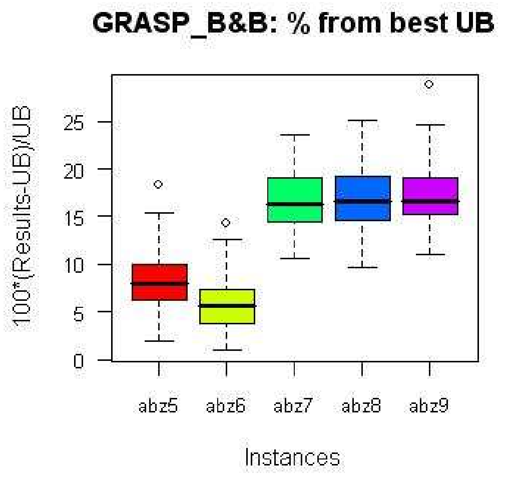


Fig. 5.8 **D**istribution of results of GRASP_B&B for instances abz

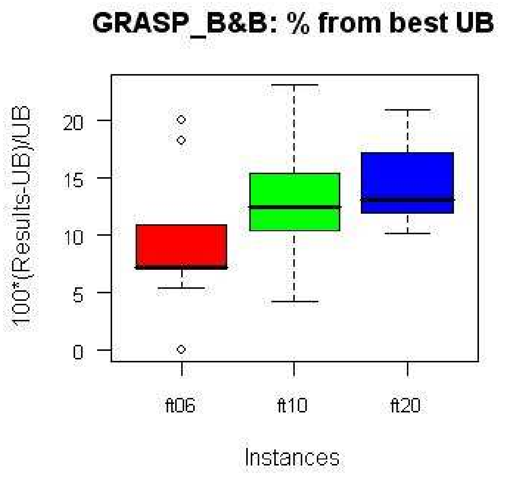


Fig. 5.9 Distribution of results of GRASP_B&B for instances ft

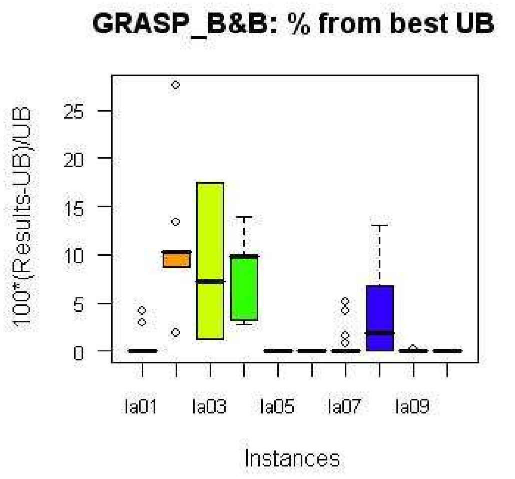


Fig. 5.10 Distribution of results of GRASP_B&B for instances la01-10

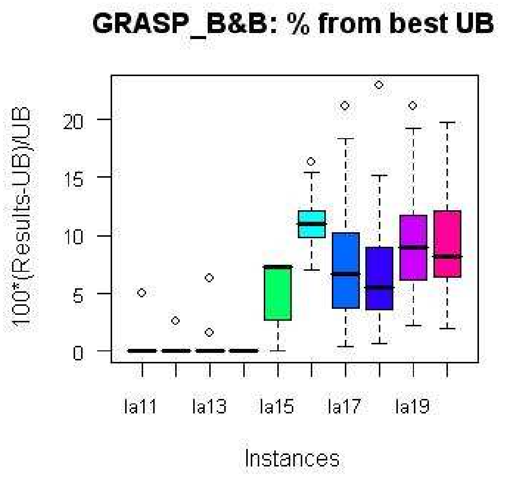


Fig. 5.11 Distribution of results of GRASP_B&B for instances la11-20

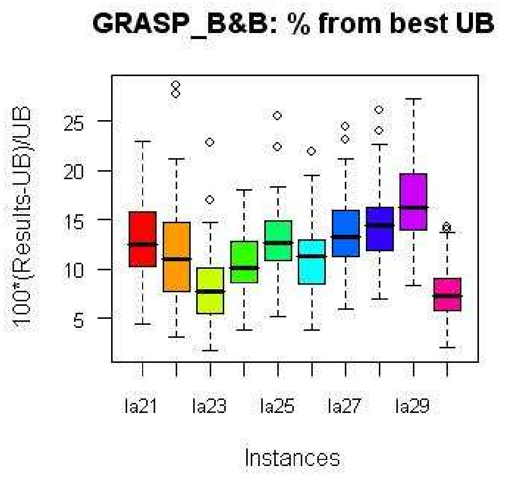


Fig. 5.12 Distribution of results of GRASP_B&B for instances la21-30

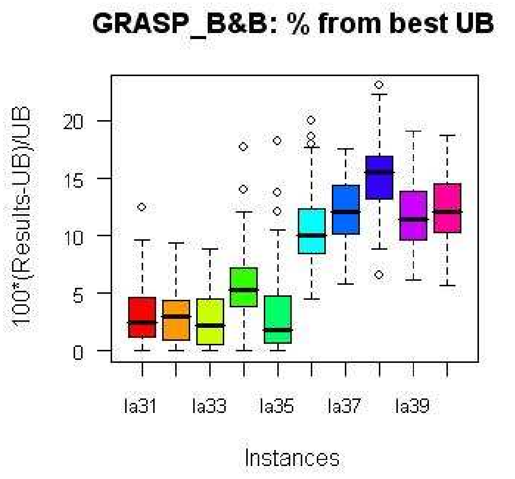


Fig. 5.13 Distribution of results of GRASP_B&B for instances la31-40

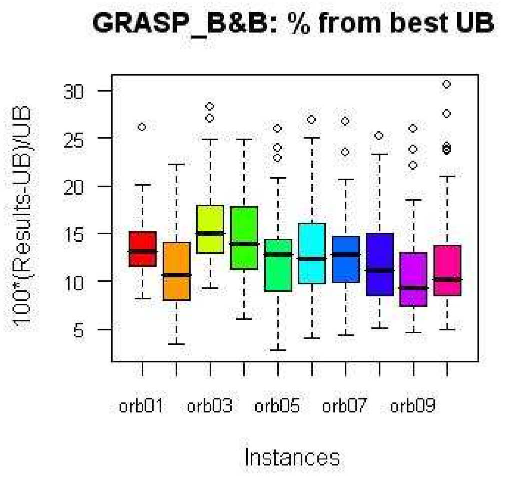


Fig. 5.14 Distribution of results of GRASP_B&B for instances orb

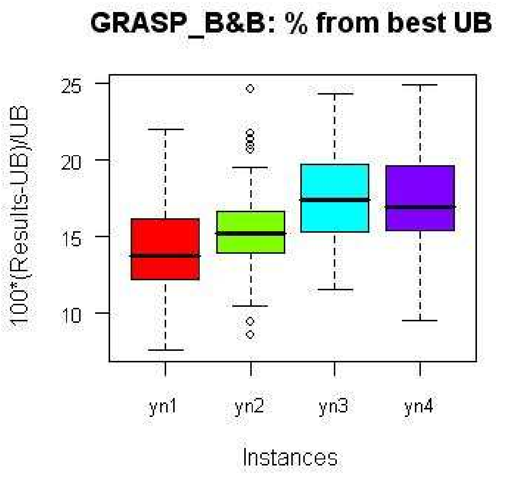


Fig. 5.15 Distribution of results of GRASP_B&B for instances yn

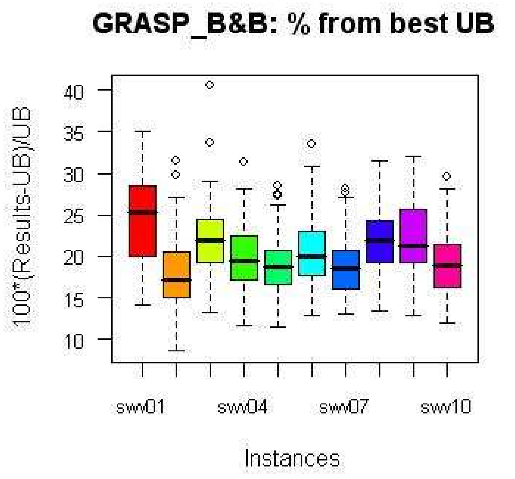


Fig. 5.16 Distribution of results of GRASP_B&B for instances swv01-10

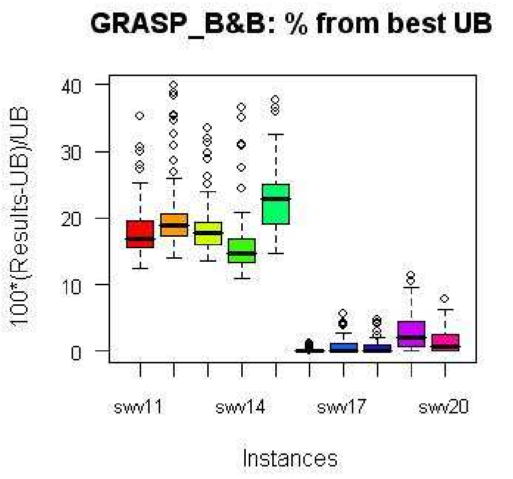


Fig. 5.17 Distribution of results of GRASP_B&B for instances swv11-20

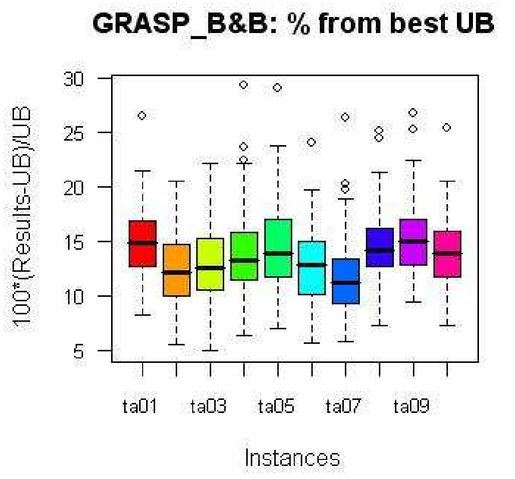


Fig. 5.18 Distribution of results of GRASP_B&B for instances ta01-10

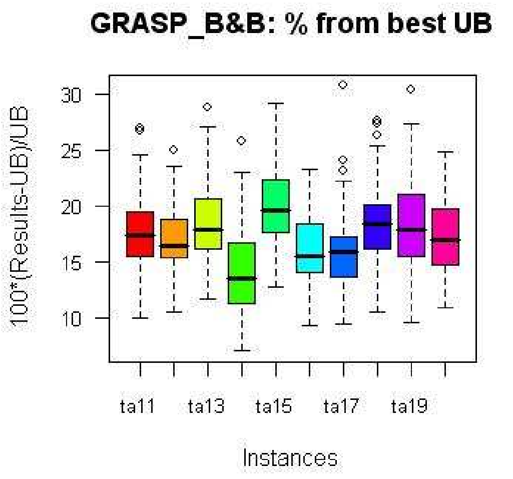


Fig. 5.19 Distribution of results of GRASP_B&B for instances ta11-20

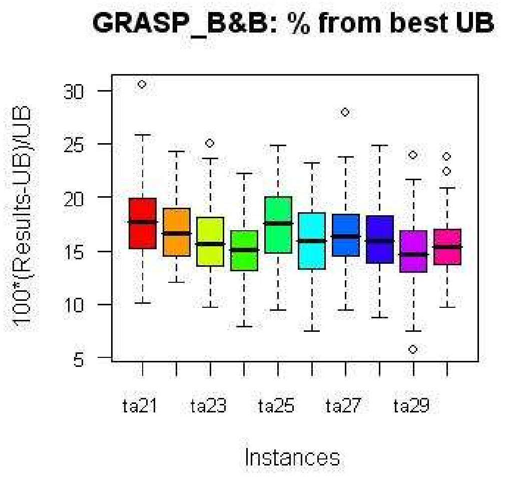


Fig. 5.20 Distribution of results of GRASP_B&B for instances ta21-30

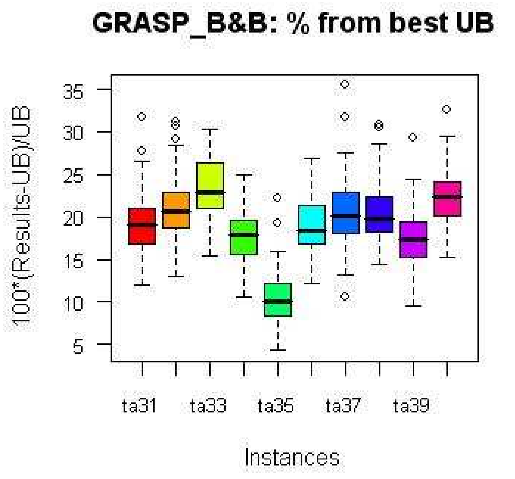


Fig. 5.21 Distribution of results of GRASP_B&B for instances ta31-40

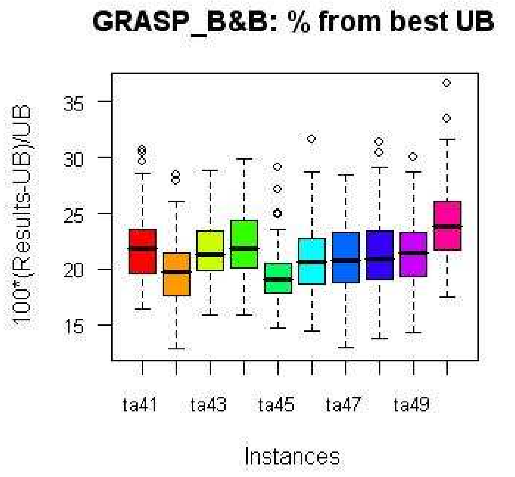


Fig. 5.22 Distribution of results of GRASP_B&B for instances ta41-50

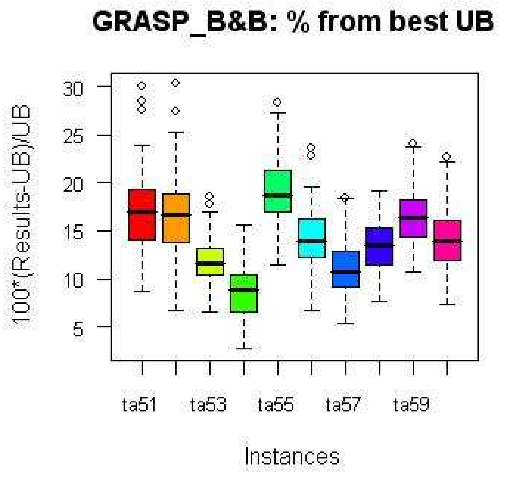


Fig. 5.23 Distribution of results of GRASP_B&B for instances ta51-60

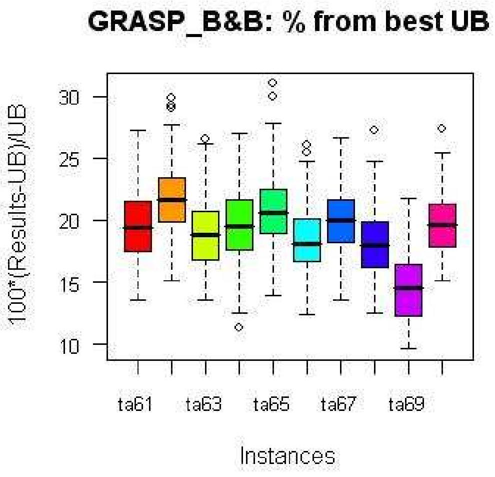


Fig. 5.24 Distribution of results of GRASP_B&B for instances ta61-70

5.4.1 Comparison to Other Procedures

GRASP_B&B is a very simple GRASP algorithm with a construction phase very similar to the one of the shifting bottleneck. Therefore we show comparative results to two other procedures; a simple GRASP procedure of Binato, Hery, Loewenstern and Resende (Binato, Hery et al. 2002), and the shifting bottleneck procedure of Adams, Balas and Zawack (Adams, Balas et al. 1988).

Comparison to the GRASP of Binato, Hery, Loewenstern and Resende The building step of the construction phase of the GRASP in (Binato, Hery et al. 2002) is a single operation of a job. In their computational results, they present the time in seconds per thousand iterations (an iteration is one building phase followed by a local search) and the thousands of iterations. For a comparison purpose we multiply these values to get the total computation time. For GRASP_B&B we present the total time of all runs $(ttime)$, in seconds. As the tables show, our algorithm is much faster. Whenever our GRASP achieves a solution not worse than theirs, we present the respective value in bold. This happens for 26 of the 58 instances whose results where compared.

Table 5.1 Comparison to GRASP for Instances abz

name	GRASP_B&B	ttime (s)	GRASP	time (s)
abz5	1258	0.7650	1238	6030
abz6	952	0.7660	947	62310
abz7	725	10.9070	667	349740
abz8	734	10.5160	729	365820
abz9	**754**	10.4690	758	343710

Table 5.2 Comparison to GRASP for Instances ft

name	GRASP_B&B	ttime (s)	GRASP	time (s)
ft06	**55**	0.1400	55	70
ft10	970	1.0000	938	261290
ft20	1283	0.4690	1169	387430

Table 5.3 Comparison to GRASP for Instances orb

name	GRASP_B&B	ttime (s)	GRASP	time (s)
orb01	1145	0.9850	1070	116290
orb02	918	0.9530	889	152380
orb03	1098	1.0150	1021	124310
orb04	1066	1.1250	1031	124310
orb05	911	0.8750	891	112280
orb06	1050	1.0460	1013	124310
orb07	414	1.0630	397	128320
orb08	945	1.0310	909	124310
orb09	978	0.9060	945	124310
orb10	991	0.8430	953	116290

Table 5.4 Comparison to GRASP for Instances la01-20

name	GRASP_B&B	ttime (s)	GRASP	time (s)
la01	**666**	0.1720	666	140
la02	667	0.1560	655	140
la03	605	0.2190	604	65130
la04	607	0.1710	590	130
la05	**593**	0.1100	593	130
la06	**926**	0.1710	926	240
la07	**890**	0.2030	890	250
la08	**863**	0.2970	863	240
la09	**951**	0.2810	951	290
la10	**958**	0.1410	958	250
la11	**1222**	0.2660	1222	410
la12	**1039**	0.2650	1039	390
la13	**1150**	0.3750	1150	430
la14	**1292**	0.2180	1292	390
la15	**1207**	0.9060	1207	410
la16	1012	0.7350	946	155310
la17	787	0.7660	784	60300
la18	854	0.7500	848	58290
la19	861	0.9690	842	31310
la20	920	0.8130	907	160320

Table 5.5 Comparison to GRASP for Instances la21-40

name	GRASP_B&B	ttime (s)	GRASP	time (s)
la21	1092	2.0460	1091	325650
la22	**955**	1.7970	960	315630
la23	1049	1.8900	1032	65650
la24	**971**	1.8440	978	64640
la25	**1027**	1.7960	1028	64640
la26	**1265**	3.3750	1271	109080
la27	**1308**	3.5620	1320	110090
la28	1301	3.0000	1293	110090
la29	**1248**	3.2960	1293	112110
la30	1382	3.3280	1368	106050
la31	**1784**	7.0160	1784	231290
la32	**1850**	6.2350	1850	241390
la33	**1719**	7.9060	1719	241390
la34	**1721**	8.2810	1753	240380
la35	**1888**	5.6880	1888	222200
la36	**1325**	4.2650	1334	115360
la37	1479	4.7970	1457	115360
la38	1274	5.1090	1267	118720
la39	1309	4.4530	1290	115360
la40	1291	5.3910	1259	123200

Comparison to the Shifting Bottleneck of Adams, Balas and Zawack The comparison between the shifting bottleneck procedure (Adams, Balas et al. 1988) and the GRASP_B&B is also presented in tables. Comparing the computation times of both procedures, our GRASP is slightly faster than the shifting bottleneck for smaller instances. Given the distinct computers used in the experiments we would say that this is not meaningful, but the difference does get accentuated as the dimensions grow. Whenever GRASP_B&B achieves a solution better than the shifting bottleneck procedure, we present its value in bold. This happens in 29 of the 48 instances whose results where compared, and in 16 of the remaining 19 instances the best value found was the same.

Table 5.6 Comparison to Shifting Bottleneck for Instances abz

name	GRASP_B&B	ttime (s)	Shifting Bottleneck	time (s)
abz5	**1258**	0.7650	1306	5.7
abz6	**952**	0.7660	962	12.67
abz7	**725**	10.9070	730	118.87
abz8	**734**	10.5160	774	125.02
abz9	754	10.4690	751	94.32

Table 5.7 Comparison to Shifting Bottleneck for Instances ft

name	GRASP_B&B	ttime (s)	Shifting Bottleneck	time (s)
ft06	55	0.1400	55	1.5
ft10	**970**	1.0000	1015	10.1
ft20	**1283**	0.4690	1290	3.5

Table 5.8 Comparison to Shifting Bottleneck for Instances la01-10

name	GRASP_B&B	ttime (s)	Shifting Bottleneck	time (s)
la01	**666**	0.1720	666	1.26
la02	**667**	0.1560	720	1.69
la03	**605**	0.2190	623	2.46
la04	607	0.1710	597	2.79
la05	593	0.1100	593	0.52
la06	926	0.1710	926	1.28
la07	890	0.2030	890	1.51
la08	**863**	0.2970	868	2.41
la09	951	0.2810	951	0.85
la10	**958**	0.1410	959	0.81

Table 5.9 Comparison to Shifting Bottleneck for Instances la11-20

name	GRASP_B&B	ttime (s)	Shifting Bottleneck	time (s)
la11	1222	0.2660	1222	2.03
la12	1039	0.2650	1039	0.87
la13	1150	0.3750	1150	1.23
la14	1292	0.2180	1292	0.94
la15	1207	0.9060	1207	3.09
la16	**1012**	0.7350	1021	6.48
la17	**787**	0.7660	796	4.58
la18	**854**	0.7500	891	10.2
la19	**861**	0.9690	875	7.4
la20	**920**	0.8130	924	10.2

Table 5.10 Comparison to Shifting Bottleneck for Instances la21-40

name	GRASP_B&B	ttime (s)	Shifting Bottleneck	time (s)
la21	**1092**	2.0460	1172	21.9
la22	**955**	1.7970	1040	19.2
la23	**1049**	1.8900	1061	24.6
la24	**971**	1.8440	1000	25.5
la25	**1027**	1.7960	1048	27.9
la26	**1265**	3.3750	1304	48.5
la27	**1308**	3.5620	1325	45.5
la28	1301	3.0000	1256	28.5
la29	**1248**	3.2960	1294	48
la30	**1382**	3.3280	1403	37.8
la31	1784	7.0160	1784	38.3
la32	1850	6.2350	1850	29.1
la33	1719	7.9060	1719	25.6
la34	1721	8.2810	1721	27.6
la35	1888	5.6880	1888	21.3
la36	**1325**	4.2650	1351	46.9
la37	**1479**	4.7970	1485	6104
la38	**1274**	5.1090	1280	57.5
la39	**1309**	4.4530	1321	71.8
la40	**1291**	5.3910	1326	76.7

5.5 Discussion on GRASP_B&B

This very simple optimised search heuristic, the GRASP_B&B, is intended to be a starting point for a more elaborated metaheuristic. We have compared it to other base procedures used within more complex algorithms; namely a GRASP of Binato, Hery, Loewenstern and Resende (Binato, Hery et al. 2002), which is the base for a GRASP with path-relinking procedure of Aiex, Binato and Resende (Aiex, Binato et al. 2001), and the shifting bottleneck procedure of Adams, Balas and Zawack (Adams, Balas et al. 1988), incorporated in the successful guided local search of Balas and Vazacopoulos (Balas and Vazacopoulos 1998). The comparison to the GRASP in (Binato, Hery et al. 2002) shows that our procedure is much faster than theirs. The quality of their best solution is slightly better than ours in 60% of the instances tested. When comparing GRASP_B&B with the Shifting Bottleneck (Binato, Hery et al. 2002), ours is still faster, and it achieves better solutions, except for 3 of the comparable instances.

The description of this new method GRASP_B&B was published in the short paper *A GRASP and Branch-and-Bound Metaheuristic for the Job-Shop Scheduling* (Fernandes and Lourenço 2007), and in an extended version, the paper *A Simple Optimised Search Heuristic for the Job Shop Scheduling Problem* (Fernandes and Lourenço 2008b).

6. An OSH Combining Tabu Search with the Verification of Violated Valid Inequalities

In this section we present an OSH procedure that uses valid inequalities to reconstruct a local optimal solution that has been partially destroyed. We named this procedure Tabu_VVI because it combines a Tabu Search heuristic with Valid Inequalities properties. The algorithm Tabu_VVI has two main stages. The first stage consists of building a feasible solution, and executing the tabu search procedure starting from it. The second stage consists of a large step followed by the tabu search, and it is repeated for a predefined number of iterations. The large step partially destroys the solution delivered by the tabu search (using a greedy randomised method to choose which elements to "delete"), looks for violated valid inequalities that enforce some order between unscheduled operations, and then rebuilds a complete solution respecting those established orders. The information about the algebraic structure of the problem within the valid inequalities is used to guide the search. The idea is to perturb the current complete solution achieving diversification and leading the search method to new unexplored regions of the solution space.

The main loop of the algorithm is stopped either when the lower bound of the instance is achieved (LB), or a predefined maximum number of iterations are executed without improving the upper bound (UB).

Fig. 6.1 shows a not detailed and simplified pseudo-code of algorithm Tabu_VVI. The main procedures of this algorithm are the following ones: building a feasible solution – line (1), the tabu search heuristic – lines (2) and (9), and the large step – lines (6) to (8). These procedures will be explained in detail in the following sections.

Algorithm Tabu_VVI

```
(1)   $x_i = GRASP\_B\&B(runs)$
(2)   $x = TabuSearch(x_i)$
(3)   $UB = makespan(x)$
(4)   $x_b = x$
(5)   while( $(UB > LB)$ and $(\#iterations\ without\ improvement < \max \#iterations)$ )
(6)       $x_d = Destroy(x)$
(7)       $x_d = FindValidInequalities(x_d)$
(8)       $x = \mathrm{Re}build(x_d)$
(9)       $x = TabuSearch(x)$
(10)      if( $makespan(x) < UB$ )
(11)          $update\ UB$
(12)          $x_b = x$
(13)  return( $x_b$ )
```

Fig. 6.1 Outline of Tabu_VVI: (x_i) - initial feasible solution, (x) - current complete solution, (x_d) - partially destroyed solution, (x_b) – best solution

6.1 Building a Feasible Solution

The algorithm Tabu_VVI first builds a feasible solution using the GRASP_B&B algorithm, described earlier on the previous chapter.

6.2 The Tabu Search Module

A tabu search procedure (Glover 1989), (Glover 1990) is a local search procedure that inspects the all neighbourhood of a current solution x and executes the move that produces the best not-tabu neighbour $ybest$. The move that goes back from $ybest$ to x becomes tabu, there is, forbidden. The objective value of $ybest$ may be worse than the one of x. The procedure stops after a predefined number of iterations have been performed without improving the best solution found.

In order to implement a simple tabu search procedure we need to define the neighbourhood structure, the characterisation of a tabu move or neighbour, the tabu length that defines how long will a move remain tabu, and an aspiration criterion, to be able to execute moves abusively considered tabu. (this abuse happens because we do not keep track of the pair of solutions before and after a move, but only of some features of the move).

The neighbourhood structure of the tabu search implemented is the same used in the local search of the GRASP_B&B, with forward and backward moves defined on critical pairs of operations (please refer to section 5.3 for the definition of the moves and the way to evaluate their value). But this time we keep track of those moves rejected by conditions d) because they can not guarantee that a cycle is not produced in the disjunctive graph, there is they can lead to an infeasible solution.

The tabu list stores for each move performed on a solution x, the critical pair of operations (u,v) involved, the type of move – forward or backward, and the number of neighbours of solution x.

The number of iterations a move (performed on solution x) stays tabu – the tabu length – is defined so it depends on the size of the neighbourhood of solution x. If a solution x has many neighbours, the reverse move of the one executed to leave from it stays tabu for a longer number of iterations than the reverse move of the one executed to leave from a solution y with a smaller neighbourhood. This way we state that the possibility of returning to a previously visited solution is not equal for every solution but depends on the number of neighbours it has.

The aspiration criterion allows a tabu move to be executed if the value of the resulting solution is better than the best one found so far.

When inspecting the solution space, tabu search inspects the whole neighbourhood of the current solution looking for its best not tabu neighbour. If the neighbourhood of a solution is empty, i.e., if the solution has no valid neighbours, it looks in the excluded moves; moves that do not verify conditions d), for feasibility and executes the one that generates the best feasible solution. If none of the excluded moves produces a feasible solution it then executes the tabu move that would remain tabu for the shortest number of iterations.

The tabu list actually stores for each move, besides the critical pair of operations involved and the type of move, a length field with the number of the iteration when the move was executed plus the number of neighbours of the solution where the move was perform on. To verify if a move is tabu is just to compare the number of the current iteration with the number stored in the length field of the tabu list in the position corresponding to the move. To update the tabu list is to add a new item in the list for the last executed move and to delete all items which have a value less than the number of the current iteration in the field length. The tabu list is implemented in a heap structure allowing efficient implementations of procedures update_tabu and verify_tabu.

Every time the tabu search improves the best known solution an intensification scheme is performed that consists in repeating the tabu search, this time duplicating the number of allowed iterations without improvement.

Algorithm TabuSearch (x)

```
(1)   $y_{best} = x$
(2)   while $(\#iterations\ without\ improvement < \max \#tabu\ iterations)$
(3)       $tabu\_UB = UB$
(4)       $y^* = none$
(5)       $mk^* = \infty$
(6)       for( $y \in N(x)$ )
(7)           $y^* = InspectNeighbour(y, tabu\_UB, mk^*, y^*)$
(8)       if( $y^*\ not\ found$ )
(9)           for( $y \in R_j N(x)$ )
(10)              if( $y\ feasible$ )
(11)                  $y^* = InspectNeighbour(y, tabu\_UB, mk^*, y^*)$
(12)          if( $y^*\ not\ found$ )
(13)              $y^* = tabu\_move(x)$
(14)      $execute\_move(x, y^*)$
(15)      $update\_tabu\_list$
(16)      if( $tabu\_UB < UB$ )
(17)          $y_{best} = y^*$
(18)          $UB = tabu\_UB$
(19)  return( $y_{best}$ )
```

```
Algorithm InspectNeighbour $(y, tabu\_UB, mk^*, y^*)$
(1)   $mk = estimate(y)$
(2)   if( $mk < tabu\_UB$ )
(3)       $y^* = y$
(4)       $mk^* = mk$
(5)       $tabu\_UB = mk^*$
(6)   else
(7)       $verify\_tabu(y)$
(8)       if(( $y$ not tabu ) and ( $mk < mk^*$ ))
(9)           $y^* = y$
(10)          $mk^* = mk$
(11)  return( $y^*$ )
```

Fig. 6.2 Pseudo-code of module Tabu Search: (x) - current complete solution, (y) - neighbour solution, y^* - best neighbour solution, y_{best} – best solution found, $N(x)$– neighbourhood of solution x, $R_j N(x)$ – rejected moves of the neighbourhood of solution x

Fig. 6.2 shows a not detailed pseudo-code of the tabu search module of the algorithm. Lines (6) and (7) of algorithm TabuSearch implement the neighbourhood search; lines (9) to (11) refer to the inspection of moves not verifying conditions d) (which happens whenever there are no moves verifying these conditions); and line (13) represents the decision to execute a tabu move, whenever the rejected moves do not produce a feasible solution. Lines (2) to (5) of algorithm InspectNeighbour implement the aspiration criterion.

6.3 Large Step

The main objective of the large step is to force a large modification in the local optimal solution achieved by the tabu search module, redirecting the search path to a different and preferably unexplored region with better quality of the solution's space. This large step has three main procedures: partially destroying a solution; finding violated valid inequalities and rebuilding a complete solution.

In the module that partially destroys the current solution, some of the sequences of processing operations in the machines are removed from the solution, i.e. in the graph that represents the solution, the disjunctive arcs between operations that define the processing sequence on the machines are eliminated for some of the machines. When the disjunctive arcs defining the processing sequence of a machine k are eliminated we say that machine k is deleted from the solution. The process of choosing the machines to delete is done using a greedy randomised heuristic, proposed in the next section 6.3.1.

The finding violated valid inequalities module is executed after partially destroying the solution. It looks for valid inequalities violated by the current partial solution. These inequalities are used to set the relative position (in the processing sequence) of some operations on a “deleted” machine. In the disjunctive graph representing the partial solution, setting the relative position (in the processing sequence) of two operations means adding to the graph one specific disjunctive arc between the two operations. The finding violated valid inequalities module, which will be presented in detail in section 6.3.2, is the one responsible for forcing a change in the direction of the search path in the solution’s space.

The rebuild the solution module finally reconstructs a complete solution including for one “deleted” machine at a time the sequences of processing operations. These sequences are forced to respect the relative positions determined by the violated valid inequalities. The rebuild the solution module is proposed in section 6.3.3.

6.3.1 Partially Destroying a Solution

The tabu search module of the algorithm provides a local optimal solution (and its makespan is an upper bound for the optimal value). This solution is then submitted to a perturbation which eliminates the processing sequence of operations on some machines. A greedy randomised method is used to choose which machines will have their processing sequence deleted. This method is biased either towards machines that, when their processing sequence is deleted, lead to a bigger reduction on the makespan of the solution – greedy_max; or towards machines that lead to the smallest reduction on the makespan – greedy_min.

When perturbating a complete solution we keep “deleting” machines (destroying the sequence for processing their operations) until the makespan of the resulting partial solution is less than the upper bound.

At the beginning of the Tabu_VVI algorithm the Destroy Module uses the greedy_max criterion to choose which machines are “deleted”. After a predefined maximum number of global iterations of Tabu_VVI are executed without improving the best solution found, the criterion for choosing the machines to “delete” changes to greedy_min. The method changes again to the criterion greedy_max after the same amount of iterations, provided that the solution has been improved at least once while using greedy_min. While the best solution found is updated at least once for each criterion, we keep running the algorithm, alternating the criterion for “deleting” the machines from the solution.

Fig. 6.3 shows a not detailed pseudo-code of the destroy module of the algorithm.

```
Algorithm Destroy $(x)$
(1)    $x_d = x$
(2)    while( $makespan(x_d) > UB - 1$)
(3)        $x_d = delete1machine(x_d)$
(4)        if ( $x_d$ empty )
(5)            return $(x)$
(6)    return $(x_d)$
```

Fig. 6.3 Pseudo-code of module Destroy: (x) - current complete solution, (x_d) - partially destroyed solution

6.3.2 Finding Violated Valid Inequalities

Having a partial solution and an upper bound (UB) for the optimal value, we then test the existence of violated valid inequalities. These allow us to establish some relative positions between operations of each unscheduled machine.

The procedure looks for violated valid inequalities for every machine whose sequence of operations is not present on the current partial solution. The process cycles through all the “deleted” machines and is repeated until no more orders (relative positions) between operations are set.

We use the same inequalities that were used in the branch-and-bound algorithms of Carlier and Pinson (Carlier and Pinson 1989) and Applegate and Cook (Applegate and Cook 1991).

Let α be a machine of the instance whose sequence of processing the operations was deleted from the solution, and S_α any given sub-set of the operations processed by α. Every operation i has an earliest possible starting time - e_i, a processing time - p_i and a minimum completion time after it is processed - f_i.

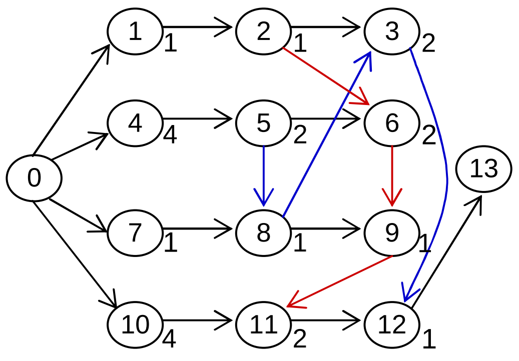
If for any given set S_α and any given operation $i \in S_\alpha$, $\min_{j \in S_\alpha \setminus \{i\}} \{e_j\} + \sum_{j \in S_\alpha} p_j + \min_{j \in S_\alpha} \{f_j\} \geq UB$ then, to be possible to reduce the upper bound, operation i must be processed on α before any other operation in S_α. The inverse inequality $\min_{j \in S_\alpha} \{e_j\} + \sum_{j \in S_\alpha} p_j + \min_{j \in S_\alpha \setminus \{i\}} \{f_j\} \geq UB$ states that operation i must be processed on α after any other operation in S_α.

Let C_α be the set of operations not yet ordered for machine α, $E_\alpha \subseteq C_\alpha$ the sub-set of operations that could be scheduled first, and $F_\alpha \subseteq C_\alpha$ the subset of operations that could be scheduled last. If there is an operation $i \in E_\alpha$ such that $e_i + \sum_{j \in C_\alpha} p_j + \min_{j \in F_\alpha} \{f_j\} \geq UB$, then i can be removed from E_α. If E_α contains only one operation, then it must be processed on α before any other operation in C_α. The reverse inequality, $\min_{j \in E_\alpha} \{e_j\} + \sum_{j \in C_\alpha} p_j + f_i \geq UB$, states that i cannot be scheduled after all the other operations in C_α, and should be removed from F_α.

Not all the sub-sets S_α are inspected when looking for violated valid inequalities that allow us to fix orders between operations of one machine. The number of subsets of a set with n elements is 2^n, an exponential number on the size of the problem, which poses an implementation problem.

We have decided to compromise and instead of looking for valid inequalities in all the possible subsets of operations on one machine, we generate only a few of them. The subsets to be inspected are built adding to each of them the operations one by one, being the operations ordered by decreasing values of starting and completion times. This way the process of generating subsets to inspect is biased to subsets with more possibilities of concealing a violation of a valid inequality.

For illustration purpose let us consider the instance in Table 4.2 and the feasible solution with makespan 13 represented by the graph shown in Fig. 4.3. The earliest possible time for starting processing an operation i (e_i) is given by the r_i parameter computed for the respective one machine problem. Analogously, f_i is given by q_i. Let us further assume that the processing sequence of machine 1 is deleted from the solution, since it is the bottleneck machine, i.e., the one that produces the biggest change in the value of the makespan of the solution. We then get the partial solution represented by the graph in Fig. 6.4, and the one machine problem for machine 1 presented in the table next to it.

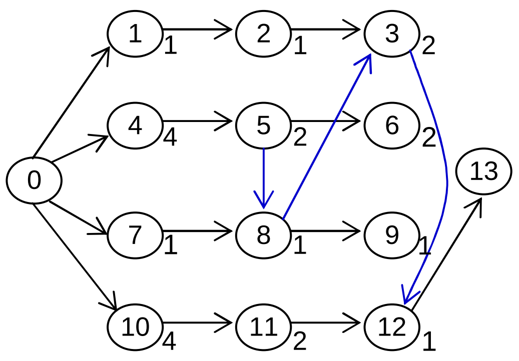


M_1	O_1	O_4	O_7	O_{10}
r_i	0	0	0	0
p_i	1	4	1	4
q_i	7	8	5	3

Fig 6.4 Graph of the partial solution removing the processing sequence of machine 1 from the solution in Fig. 4.3 and the respective one machine problem for machine 1

Since $r_i = 0 \forall i \in \{1,4,7,10\}$ and $q_4 > q_1 > q_7 > q_{10}$ the algorithm first includes operation O_4 in the set and then includes the other operations by decreasing values of its queues. The first set to be inspected is $S = \{O_4, O_1\}$ looking at the inequalities $r_1 + p_4 + p_1 + q_4 \geq UB$, that being verified implies arc $O_4 \rightarrow O_1$, and $r_4 + p_4 + p_1 + q_1 \geq UB$, that is not verified. Set $S = \{O_4, O_7\}$ is the next to be inspected with inequality $r_7 + p_4 + p_7 + q_4 \geq UB$ leading to arc $O_4 \rightarrow O_7$ and inequality $r_4 + p_4 + p_7 + q_7 \geq UB$ not being verified. Finally the set $S = \{O_4, O_{10}\}$ is considered checking the inequalities $r_{10} + p_4 + p_{10} + q_4 \geq UB$, that leads to arc $O_4 \rightarrow O_{10}$, and $r_4 + p_4 + p_{10} + q_{10} \geq UB$, that is not true. The next sets to be considered are sets with three operations including O_4 and O_1. $S = \{O_4, O_1, O_7\}$ is inspected first, looking at inequality $r_7 + p_4 + p_1 + p_7 + \min_{i \in \{4,1\}} q_i \geq UB$, which being verified states that operation O_7 must be processed after all other operations in S in order to be able to reduce the makespan and leads to arcs $O_4 \rightarrow O_7$ and $O_1 \rightarrow O_7$; and inspecting inequality $\min\limits_{i\in\{4,1\}} r_i + p_4 + p_1 + p_7 + q_7 \geq UB$, that is not true. The next set considered is $S = \{O_4, O_1, O_{10}\}$ and the inequalities inspected are $r_{10} + p_4 + p_1 + p_{10} + \min\limits_{i\in\{4,1\}} q_i \geq UB$, which leads to arcs $O_4 \rightarrow O_{10}$ and $O_1 \rightarrow O_{10}$, and $\min\limits_{i\in\{4,1\}} r_i + p_4 + p_1 + p_{10} + q_{10} \geq UB$, which is not verified. In the end the set with all the operations is considered $S = \{O_1, O_4, O_7, O_{10}\}$ inspecting the inequality $r_{10} + p_4 + p_1 + p_7 + p_{10} + \min\limits_{i\in\{4,1,7\}} q_i \geq UB$, that leads to arcs $O_4 \rightarrow O_{10}$, $O_1 \rightarrow O_{10}$ and $O_7 \rightarrow O_{10}$, and the inequality $\min\limits_{i\in\{4,1,7\}} r_i + p_4 + p_1 + p_7 + p_{10} + q_{10} \geq UB$ that implies arcs $O_{10} \rightarrow O_4$, $O_{10} \rightarrow O_1$ and $O_{10} \rightarrow O_7$. It is impossible to have both arcs $i \rightarrow j$ and $j \rightarrow i$ in a solution, for any given operations i and j, which happens for pairs of operations (O_4, O_{10}), (O_1, O_{10}) and (O_7, O_{10}). This means that we can not produce a solution with makespan less than 13 (i.e. reduce the upper bound) starting from this partial solution. The solution must be further destroyed.

The next machine whose sequence of processing operations is deleted from the solution is machine 2. Fig 6.5 shows the partial solution obtained and the corresponding one machine problem for machine 1.



M_1	O_1	O_4	O_7	O_{10}
r_i	0	0	0	0
p_i	1	4	1	4
q_i	4	6	4	3

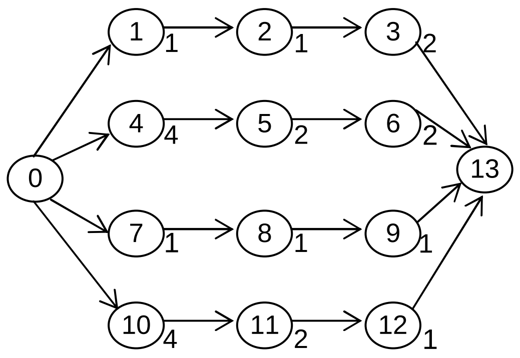
Fig 6.5 Graph of the partial solution removing the processing sequence of machines 1 and 2 from the solution in Fig. 4.3 and the respective one machine problem for machine 1

Notice that $r_i = 0 \forall i \in \{1,4,7,10\}$ and $q_4 > q_1 = q_7 > q_{10}$ and remember that $UB = 13$. The table 6.1 presents the sets considered when looking for violated valid inequalities, the inequalities inspected and the arcs implied by them.

Table 6.1 Valid inequalities inspected and corresponding arcs for partial solution of Fig. 6.5.

sets	valid inequalities		arcs
$S = \{O_4, O_1\}$	$r_1 + p_4 + p_1 + q_4 \geq UB$	false	
	$r_4 + p_4 + p_1 + q_1 \geq UB$	false	
$S = \{O_4, O_7\}$	$r_7 + p_4 + p_7 + q_4 \geq UB$	false	
	$r_4 + p_4 + p_7 + q_7 \geq UB$	false	
$S = \{O_4, O_{10}\}$	$r_{10} + p_4 + p_{10} + q_4 \geq UB$	true	$O_4 \to O_{10}$
	$r_4 + p_4 + p_{10} + q_{10} \geq UB$	false	
$S = \{O_4, O_1, O_7, O_{10}\}$	$r_{10} + p_4 + p_1 + p_7 + p_{10} + \min_{i \in \{4,1,7\}} q_i \geq UB$	true	$O_4 \to O_{10}$ $O_1 \to O_{10}$ $O_7 \to O_{10}$
	$\min_{i \in \{4,1,7\}} r_i + p_4 + p_1 + p_7 + p_{10} + q_{10} \geq UB$	true	$O_{10} \to O_4$ $O_{10} \to O_1$ $O_{10} \to O_7$

Again there are incompatible arcs deduced from the violated valid inequalities, so the solution must be further destroyed. The only machine whose sequence of processing operations is present in the solution is machine 3. Fig 6.6 shows the partial solution obtained from deleting the processing sequence of machine 3 (the empty solution) and the corresponding one machine problem for machine 1.



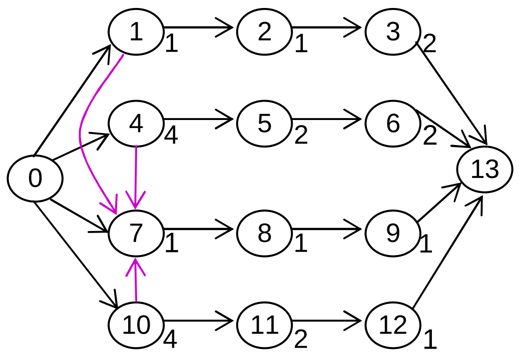
M_1	O_1	O_4	O_7	O_{10}
r_i	0	0	0	0
p_i	1	4	1	4
q_i	3	4	2	3

Fig 6.6 Graph of the partial solution removing the processing sequence of machines 1, 2 and 3 from the solution in Fig. 4.3 and the respective one machine problem for machine 1

Notice that $r_i = 0 \forall i \in \{1,4,7,10\}$ and $q_4 > q_{10} = q_1 > q_7$. The algorithm does not look for violated valid inequalities for sets with two operations because operation O_4 is the first to be included and $r_4 + p_4 + q_4 + \min_{i \in \{1,7,10\}} p_i < UB$. Since $q_{10} = q_1$ ($p_{10} > p_1$) operations O_{10} and O_1 are included at once in the set. So the only set considered for inspecting violated valid inequalities is $S = \{O_4, O_{10}, O_1, O_7\}$ and the algorithm looks at inequality $r_7 + p_4 + p_{10} + p_1 + p_7 + \min_{i \in \{4,10,1\}} q_i \geq UB$, which implies arcs $O_4 \to O_7$, $O_{10} \to O_7$ and $O_1 \to O_7$, and inequality $\min_{i \in \{4,10,1\}} r_i + p_4 + p_{10} + p_1 + p_7 + q_7 \geq UB$, which is not true.

At this moment the algorithm updates the sets C_α, E_α and F_α for machine 1. The position in the processing sequence for operation O_7 is completely determined, it is processed after all the others, so it is removed from the set of operations not yet scheduled, i.e., $C_1 = \{O_1, O_4, O_{10}\}$. The set of operations that can be scheduled first E_1 and the set of operations that can be scheduled last F_1 on machine 1 are equal to C_1. The algorithm then inspects all the inequalities of the type $r_i + \sum_{j \in C_\alpha} p_j + \min_{j \in F_\alpha} \{q_j\} \geq UB$ and $\min_{j \in E_\alpha} \{r_j\} + \sum_{j \in C_\alpha} p_j + q_i \geq UB$. The only one that is true is $\min_{i \in C_1} r_i + p_1 + p_4 + p_{10} + q_4 \geq UB$, meaning that operation O_4 can not be the last one on set C_1 to be processed and so F_1 is updated to $F_1 = \{O_1, O_{10}\}$.

Including all new arcs in the current partial solution we get the partial solution corresponding to the graph presented in Fig 6.7. Next to it is the updated one machine problem for machine 2.



M_2	O_2	O_6	O_9	O_{11}
r_i	1	6	11	4
p_i	1	2	1	2
q_i	2	0	0	1

Fig 6.7 Graph of the partial solution including all new arcs generated by valid inequalities for machine 1 after having removed the processing sequence of machines 1, 2 and 3 from the solution in Fig. 4.3 and the respective one machine problem for machine 2

The algorithm proceeds verifying valid inequalities for sets of operations processed on machine 2. It adds the new arcs obtain from the violated valid inequalities, updates the one machine problem for machine 3 and proceeds looking for valid inequalities in sets of operations processed by machine 3. Again the new arcs are added to the graph of the partial solution and the process cycles through all the unscheduled machines until no new arcs are included for none of the machines. Only then the algorithm moves to the next step; rebuilding a complete feasible solution.

If when looking for violated valid inequalities we find none, then we reintroduce a deleted processing sequence of a machine into the current partial solution and we look again for violated valid inequalities. The processing sequence to add to the solution is chosen randomly from the machines present in the complete solution from which the partial solution derives. We say that the violated valid inequalities lead to incompatible sequences of operations, when while adding the disjunctive arcs (corresponding to the relative positions between operations in the processing sequence) determined by the valid inequalities to the graph representing the partial solution we get a cycle (thus an infeasible partial solution). If the violated valid inequalities lead to incompatible sequences of operations this means we cannot improve the upper bound (UB) with the set of sequenced machines, and another machine is deleted from the solution. If this happens repeatedly, the solution becomes empty (without any processing sequence on the machines) and still violated valid inequalities lead to incompatible arcs, then the current complete solution is optimal. Fig. 6.8 shows a not detailed pseudo-code of the find violated valid inequalities module of the algorithm.

```
Algorithm FindValidInequalities $(x_d)$
(1)  repeat
(2)      $fixedordersflag = 1$
(3)      while( $fixedordersflag > 0$ )
(4)          $fixedordersflag = 0$
(5)          for( $every\ deleted\ machine$ )
(6)              if( $valid\ inequalities\ found$ )
(7)                  $fixedordersflag = 1$
(8)                  $set\ fixed\ orders$
(9)                  if( $incompatible\ orders$ )
(10)                     $fixedordersflag = -1$
(11)                     break for
(12)     if( $no\ valid\ inequalities\ found$ )
(13)         $x_d = add1machine(x_d)$
(14)     elseif( $fixedordersflag = -1$ )
(15)         if( $x_d$ $empty$ )
(16)             return $(x)$
(17)         $x_d = delete1machine(x_d)$
(18)     else
(19)         return $(x_d)$
```

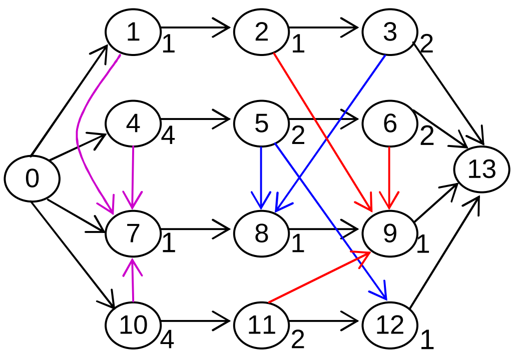
Fig. 6.8 Pseudo-code of module Find Valid Inequalities: (x) - current complete solution, (x_d) - partially destroyed solution

6.3.3 Rebuilding a Complete Solution

The solution is reconstructed including the sequence of operations of one machine at a time. The order of adding the sequences in the machines to the solution is the same of the elimination. The first machine to be re-included in the solution is the one that was first removed, and so on. The schedule of operations for each machine is determined using a modified version of the Schrage algorithm (Schrage 1970) that considers pre-defined orders between operations, i.e., it starts with a partial solution. Each time the sequence of operations of a machine is re-included in the solution, a restricted local search is executed, where it is forbidden to change orders fixed by the valid inequalities. This allows to immediately improve the solution. When a new sequence of operations is included, we look for new violated valid inequalities in all remaining unscheduled machines, trying to fix more orders between operations.

After the solution is complete, again, the local search is executed.

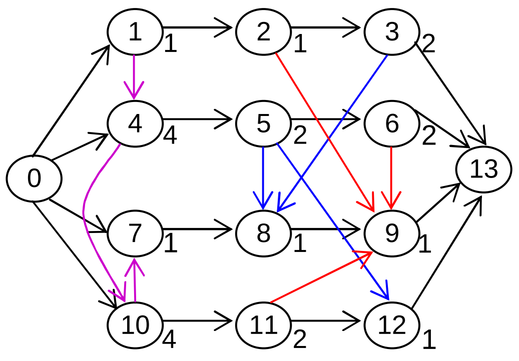
Let us use the instance of table 4.2 to illustrate this module of the algorithm. After the previous module (find valid inequalities) ends, the partial solution obtained is the one represented by the graph of Fig. 6.9.



M_1	O_1	O_4	O_7	O_{10}
r_i	0	0	9	0
p_i	1	4	1	4
q_i	5	5	2	3

Fig 6.9 Graph of the partial solution obtained at the end of module find violated valid inequalities when applying Tabu_VVI to the solution of Fig. 4.3

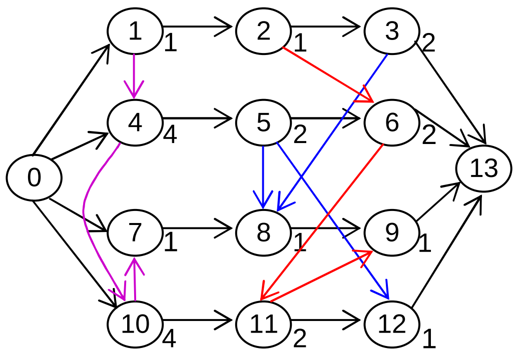
The module rebuilding a complete solution starts by building the optimal solution for the one machine problem for machine 1, with predefined arcs. It builds the solution $O_1 \rightarrow O_4 \rightarrow O_{10} \rightarrow O_7$, that is added to the graph of the partial solution for the job shop. Fig. 6.10 shows the graph of this partial solution, and the updated one machine problem for machine 2.



M_2	O_2	O_6	O_9	O_{11}
r_i	1	7	11	9
p_i	1	2	1	2
q_i	4	1	0	1

Fig 6.10 Graph of the partial solution with the complete processing sequence for machine 1 obtained at the first iteration of the module rebuilding a complete solution when applying Tabu_VVI to the solution of Fig. 4.3

The algorithm runs again the find violated valid inequalities module. It generates the arcs $O_6 \rightarrow O_{11}$, $O_2 \rightarrow O_{11}$ and $O_2 \rightarrow O_6$. This completely defines the sequence of processing operations on machine 2 $O_2 \rightarrow O_6 \rightarrow O_{11} \rightarrow O_9$, which is added to the partial solution. The local search module is run on the new partial solution but, on this small instance, it produces no changes. Fig. 6.11 shows the graph of the partial solution, and the updated one machine problem for machine 3.



M_3	O_3	O_5	O_8	O_{12}
r_i	2	5	10	11
p_i	2	2	1	1
q_i	2	5	1	0

Fig 6.11 Graph of the partial solution with the complete processing sequence for machines 1 and 2 obtained at the second iteration of the module rebuilding a complete solution when applying Tabu_VVI to the solution of Fig. 4.3

The algorithm runs again the find violated valid inequalities module. It generates the new arcs $O_8 \rightarrow O_{12}$, $O_3 \rightarrow O_{12}$ and $O_3 \rightarrow O_5$. This, once again for this small instance, completely defines the sequence of processing operations on machine 3.

$O_3 \rightarrow O_5 \rightarrow O_8 \rightarrow O_{12}$, which is added to the partial solution leading to a complete solution for this instance of the job shop scheduling problem. The makespan of the produced complete solution, represented by the graph in Fig. 6.12, is 12. The makespan is equal to the lower bound of the problem, so the solution is optimal.

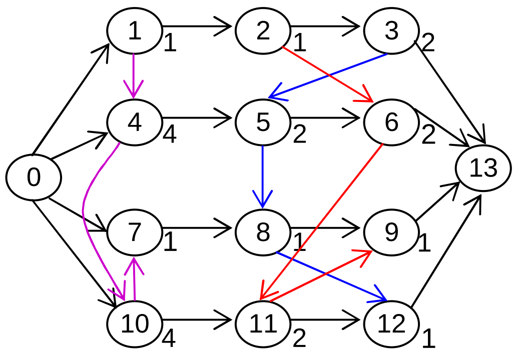


Fig 6.12 Optimal solution to instance of Fig. 4.2 achieved by Tabu_VVI

A not detailed pseudo-code of the rebuild module of the algorithm Tabu_VVI is presented in Fig. 6.13.

```
Algorithm Rebuild $(x_d)$
(1)    for( every deleted machine m )
(2)        $x_d = modified\_schrage(x_d, m)$
(3)        $x_d = LocalSearch(x_d)$
(4)        $x_d = FindValidInequalities(x_d)$
(5)    return $(x_d)$
```

Fig. 6.13 Pseudo-code of module Rebuild: (x_d) - partially destroyed solution

6.4 Computational Experiment

We have tested the performance of the method Tabu_VVI using once again the 132 benchmark instances abz5-9 (Adams, Balas et al. 1988), ft6, ft10, ft20 (Fisher and Thompson 1963), la01-40 (Lawrence 1984), orb01-10 (Applegate and Cook 1991), swv01-20 (Storer, Wu et al. 1992), ta01-50 (Taillard 1993) and yn1-4 (Yamada and Nakano 1992).

The size of the instances is measure by the number of operations (equal to the number of jobs times the number of machines). The instances have different sizes: ft6 is the smaller one with 6×6 operations; la01-05 have 10×5; la06-10 have 15×5; ft20 and la11-15 have 20×5; abz5-6, ft10, la16-20 and orb01-10 have 10×10; la21-25 have 15×10; la26-30 and swv01-05 have 20×10; la31-35 have 30×10; la36-40 and ta01-10 have 15×15; abz7-9, swv06-10 and ta11-20 have 20×15; ta31-40 and yn1-4 have 20×20; swv11-20 have 50×10; the bigger ones are ta41-50 with 30×20 operations.

An optimal solution has already been found for 83 of these instances; namely abz5-7, ft6, ft10, ft20, la01-40, orb01-10, swv01-02, swv05, swv13-14, swv16-20, ta01-10, ta14, ta17, ta31, ta35-36 and ta38-39.

We have tested slightly different versions of the method Tabu_VVI:

1) A larger neighbourhood, not forcing moves to respect conditions d), was implemented (notation ls2).

2) We have compared the results of performing (notation tabuls) and not performing (notation tabu) an unrestricted local search after the rebuild phase and before the tabu search.

3) Within the tabu search module, different values of the tabu length parameter were tested: equal to the number of neighbours (notation mv); half of it (notation mv-2), the double of it (notation mv2), etc.

4) Also inside the tabu search module, we have tested not to look for those moves rejected by conditions d), so when a neighbourhood is empty the eligible tabu move is always the one executed (notation without inf).

5) We have implemented versions where instead of only keeping the move with the best evaluation, we store them all (or just some of them) in a heap structure by increasing values of their evaluations (notation hp). Whenever, after executing the move, the real value of the solution is different from the evaluation of the move, if the value of the solution is bigger than the evaluation of the next move in the heap structure, then this next move is executed. This was also tried for the moves rejected by conditions d) (notation infhp).

The number of tabu iterations allowed without improving the best solution was set to the number of operations of each instance.

6) Within the rebuild module, we have also tested to build the sequence of processing operations in one machine using a modified branch-and-bound method (notation bb) instead of just the priority rule of the Schrage algorithm. The orders between operations that were fixed by the find violated valid inequalities module are always respected.

The described base version of the algorithm Tabu_VVI is represented by the notation tabu_mvinf. We have tested a total of 35 different versions of the algorithm. The different versions have names that respect the following denominations presented in Table 6.2.

Table 6.2 Notations for the different Variants of Algorithm Tabu_VVI

notation	description
bb	branch-and-bound at the rebuilding module
ls2	extended local search to include moves rejected by conditions d)
tabu	tabu search
tabuls	local search after the rebuild and before the tabu search
mv	tabu tenure equal to the size of the neighbourhood
mv#	tabu tenure equal to # times the size of the neighbourhood
mv-2	tabu tenure equal to half the size of the neighbourhood
mvfct	there is a factor that extends or reduces the tabu length according to improving or not improving cycles
mvinf	moves rejected by conditions d), are executed when there are no feasible ones
hp	all moves are stored in a heap – use them when the value of the solution is different from its evaluation and bigger than the evaluation of the next move in the heap
mv_infhp	heap only for moves rejected by conditions d)
mvinfhp	a heap for each type of moves (rejected and not rejected)

At the first stage of the method Tabu_VVI, the GRASP_B&B algorithm was executed with parameter *runs* equal to 10, to generate the initial feasible solution and tabu search was run for 100 iterations without improvement.

The algorithm has been run on a Pentium 4 CPU 2.80 GHz and coded in C.

In order to measure the performance of the algorithm we use again the percentage of relative error (but this time to the lower bound) - RE_{LB} (or to the optimum if the problem is closed). $f(x)$ stands for the makespan of the best solution found.

$$RE_{LB}(x) = 100\% \times \frac{f(x) - LB}{LB}$$

In section 1 of Annex C the reader can find tables that present the performance of fifteen variants of the algorithm showing the average values, over each class of instances, of the RE_{LB} and the time in seconds to the best solution. There are also shown for each variant the number of instances for which it achieved the best known solution $(best)$; the number of instances for which it achieved the minimum value of all variants $(\min)$; the number of instances for which the variant was the only one to achieve the minimum value $(only\ \min)$; the sum of the error and the sum of time over all instances. The fifteen variants chosen to present are the ones that are the only one to find the minimum value for at least one instance. The best performance measures are in bold. For the 42 instances abz6, ft06, la01, la03-18, la23, la26, la30-35, orb07, orb10, swv16-20 and ta35, all the variants find the optimal solution. Here we present Table 6.3 with the best results over all the variants tested.

Table 6.3 Best results by all variants of Tabu_VVI, in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

sets of instances	best of all variants of Tabu_VVI	
	$avg(RE_{LB})$	$avg(time)$
abz	1.71	81.46
ft	0	0.15
la01-05	0	0.03
la06-10	0	0.02
la11-15	0	0.04
la16-20	0	0.44
la21-25	0.02	7.94
la26-30	0.17	83.39
la31-35	0	0.27
la36-40	0.05	57.08
orb	0	4.30

swv01-05	2.33	127.91
swv06-10	8.21	281.64
swv11-15	1.41	1854.58
swv16-20	0	1.58
yn	7.00	163.95
ta01-10	0.24	49.52
ta11-20	3.12	177.24
ta21-30	5.96	319.02
ta31-40	1.26	220.62
ta41-50	5.47	1016.21
# best	42+22	
sum RE_{LB}	258.08	
sum time	31007	

We have verified that when a heap of the moves respecting conditions d) was kept, it was never used, that is, the evaluation of the move always corresponded to the makespan of the resulting solution. For moves not respecting conditions d), the heap was only used when the move produced an infeasible solution (a cycle in the graph is created). This shows that the evaluation function, which for determining the value of the solution obtained by a move, re-computes only the values of the paths through operations between the ones of the critical pair, is very accurate.

In section 2 of Annex C the reader can find tables showing for each instance, its size in jobs times machines, the best known upper bound (bk_UB), the best value achieved by Tabu_VVI $(Tabu\_VVI)$, the average time in seconds to achieve it $(time)$ and the number of variants of the algorithm that reach this minimum $(\#variants)$. When the best known upper bound is not the optimum value for the instance it appears in *italic*.

We have found a new upper bound, 1765, for instance swv10 in 101 seconds.

The values of best known lower and upper bounds were gathered from the papers (Jain and Meeran 1999), (Nowicki and Smutniki 1996), (Nowicki and Smutnicki 2002) and (Nowicki and Smutnicki 2005).

The following boxplots show the distribution of the makespans of the solutions achieved by the 35 variants of algorithm Tabu_VVI. The measure used to present the results is the percentage of relative error to the upper bound - RE_{UB} (or to the optimum if the problem is closed).

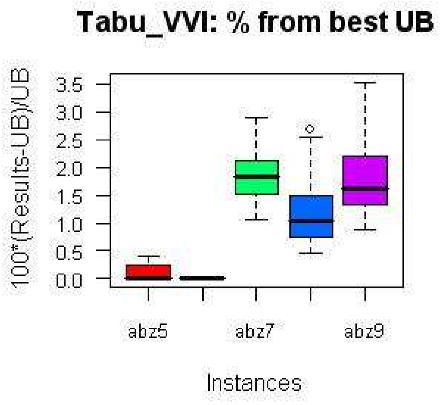


Fig. 6.14 Distribution of results of all variants of Tabu_VVI for instances abz

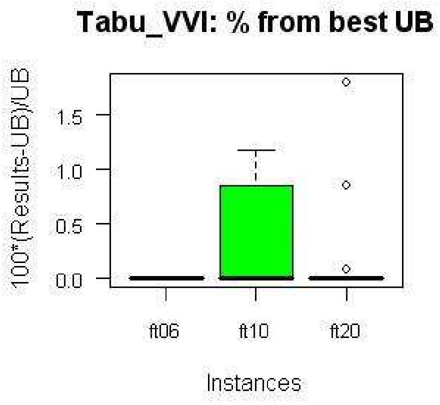


Fig. 6.15 Distribution of results of all variants of Tabu_VVI for instances ft

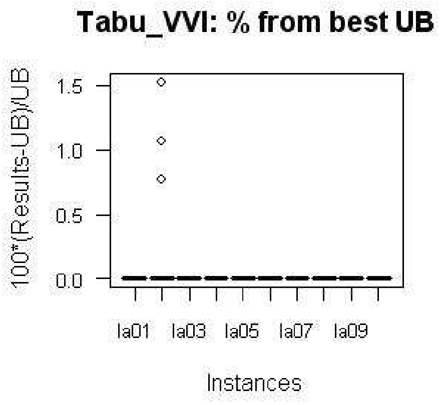


Fig. 6.16 Distribution of results of all variants of Tabu_VVI for instances la01-10

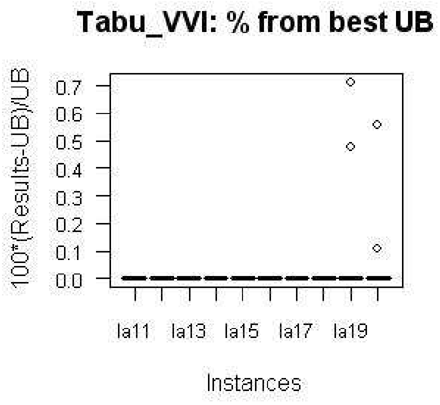


Fig. 6.17 Distribution of results of all variants of Tabu_VVI for instances la11-20

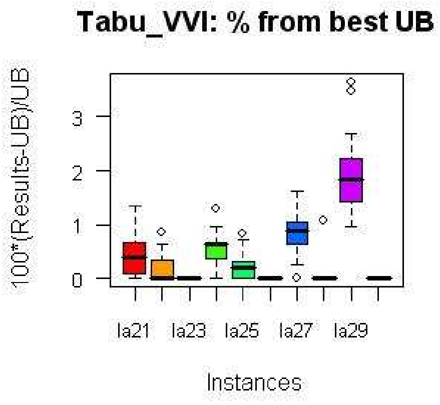


Fig. 6.18 Distribution of results of all variants of Tabu_VVI for instances la21-30

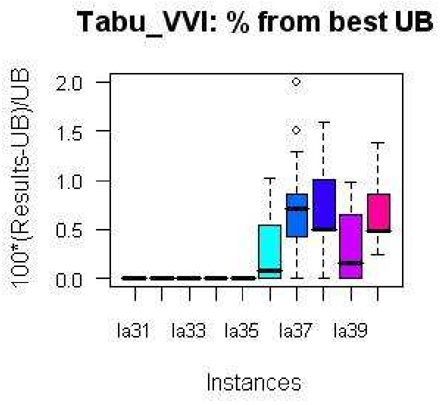


Fig. 6.19 Distribution of results of all variants of Tabu_VVI for instances la31-40

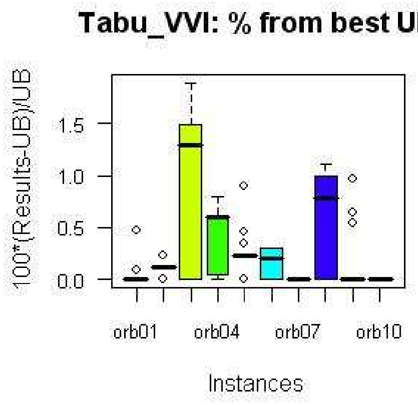


Fig. 6.20 Distribution of results of all variants of Tabu_VVI for instances orb

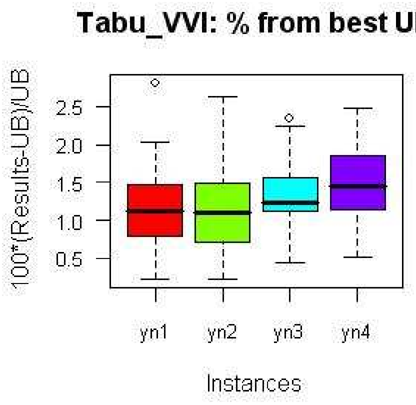


Fig. 6.21 Distribution of results of all variants of Tabu_VVI for instances yn

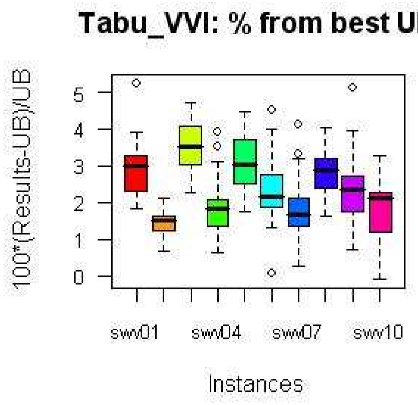


Fig. 6.22 Distribution of results of all variants of Tabu_VVI for instances swv01-10

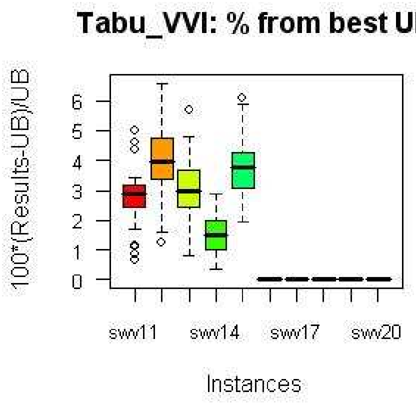


Fig. 6.23 Distribution of results of all variants of Tabu_VVI for instances swv11-20

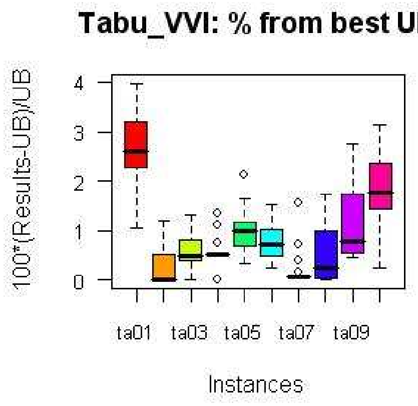


Fig. 6.24 Distribution of results of all variants of Tabu_VVI for instances ta01-10

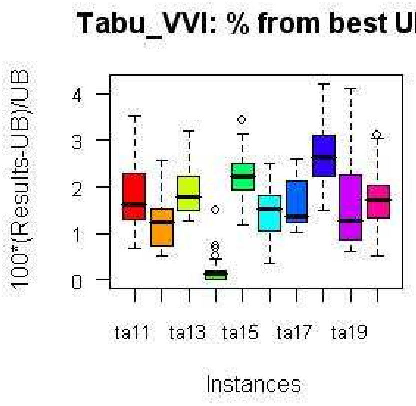


Fig. 6.25 Distribution of results of all variants of Tabu_VVI for instances ta11-20

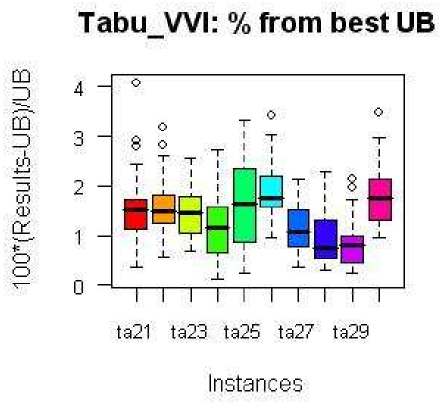


Fig. 6.26 Distribution of results of all variants of Tabu_VVI for instances ta21-30

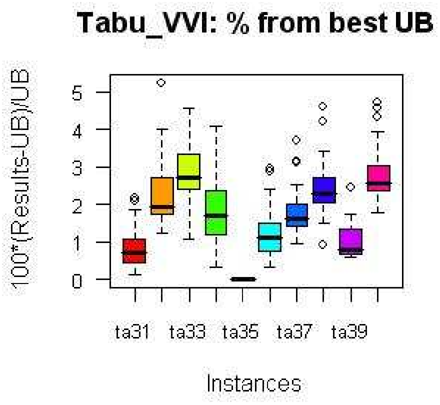


Fig. 6.27 Distribution of results of all variants of Tabu_VVI for instances ta31-40

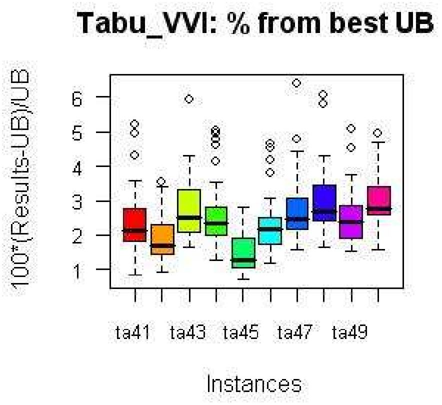


Fig. 6.28 Distribution of results of all variants of Tabu_VVI for instances ta41-50

6.4.1 Comparison to Other OSH Methods

To compare the results of Tabu_VVI to other methods we choose the two variants with the smallest sum over all instances of the percentage of the relative error to the lower bound, or the best over all variants.

The optimised search methods applied to the job-shop scheduling problem, that we know of and have mentioned in the literature review, are only applied to the older and easier instances of the problem, except for the works of Balas and Vazacopoulos (Balas and Vazacopoulos 1998) and Pezzella and Merelli (Pezzella and Merelli 2000), that will be treated separately.

The method of Danna, Rothberg and Le Pape (Danna, Rothberg et al. 2005) is applied to instances of the weighted-tardiness version of the problem, and the work of

Schaal, Fadil, Silti and Tolla (Schaal, Fadil et al. 1999) is applied to the generalised scheduling problem.

Our method, Tabu_VVI is always better, in quality of the solutions and in computational time, than the works (Chen, Talukdar et al. 1993), (Denzinger and Offermann 1999), (Tamura, Hirahara et al. 1994), (Adams, Balas et al. 1988), (Applegate and Cook 1991), (Caseau and Laburthe 1995), (Lourenço 1995), and (Lourenço and Zwijnenburg 1996). In Table 6.4 we show the comparison results to the work of Caseau and Laburthe (Caseau and Laburthe 1995) (named CL), because it is the best of these methods and also because it is the one that presents results for more instances. Their algorithm was run on a SunSparc 10 machine.

Table 6.4 Results by variants tabu_mvinf and tabu_mv_bb of Tabu_VVI, and the algorithm of Caseau and Laburthe, in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

instances	Tabu_VVI				CL	
	tabu_mvinf		tabu_mv_bb			
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
abz	2.11	63.77	1.93	61.02	2.57	112.67
ft	0	11.72	0	0.58	0	112
la01-05	0	0.12	0	0.12	0	3.80
la06-10	0	0.02	0	0.03	0	0.75
la11-15	0	0.04	0	0.05	0	27
la16-20	0	1.79	0	1.67	0	25.08
la21-25	0.11	23.13	0.06	14.80	0.11	551.40
la26-30	0.29	54.12	0.26	40.88	0.47	4322.25
la31-35	0	0.38	0	0.39	0	2108.40
la36-40	0.47	22.68	0.22	33.50	0.37	2476.40
orb	0.23	7	0.09	14.13	1.66	111.11

Comparison to the Guided Local Search The guided local search procedure of Balas and Vazacopoulos (Balas and Vazacopoulos 1998) designs a search procedure based on local improvements and accepting nonimproving moves, using structures of neighbourhood trees. Each neighbourhood tree corresponds to a cycle of the guided local search procedure. Each node of the tree stores a solution and each edge connects neighbour solutions. Feasible solutions are built solving to optimality by branch-and-bound all one-machine subproblems (like the shifting bottleneck heuristic (Adams, Balas et al. 1988)). After a few cycles of neighbourhood trees, the procedure randomly destroys the best solution found; deleting the sequence of operations for some machines, and then reconstructs the partially destroyed solution repeating the all process.

Here we compare our best results to their best reported version SB-RGSL10, which stands for shifting bottleneck with randomised guided local search. The 10 means the number of times the all process is repeated. We call it BZ. Their algorithm was run on a SunSparc 30 machine. The comparison results between algorithms Tabu_VVI and BZ are shown in Table 6.5. Although we used different computers, we can still say that our method is always much faster than BZ. Quality values that win the comparison are shown in bold.

Table 6.5 Results by the best of all variants of Tabu_VVI and the best variant of the algorithm of Balas and Vazacopoulos; in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

instances	Tabu_VVI		BZ	
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
la01-05	0	0.03	0	5.9
la16-20	0	0.44	0	47
la21-25	0	7.94	0	139.6
la26-30	**0.17**	83.4	0.19	121.6
la36-40	0.05	57.1	**0.03**	278
orb	**0**	4.30	0.10	80.18
swv01-05	2.33	128	**2.02**	1290
swv06-10	**8.06**	282	9.64	2917
swv11-15	**1.41**	1855	2.12	9173
yn	7	164	**5.96**	5938
ta01-10	**0.24**	49.5	0.25	1182
ta11-20	**3.12**	177	3.34	3383
ta21-30	**5.96**	319	6.57	4377
ta31-40	1.26	221	**1.13**	5069
ta41-50	**5.47**	1016	5.71	10726

Comparison to the Tabu Search with Shifting Bottleneck The procedure of Pezzella and Merelli (Pezzella and Merelli 2000) combines tabu search with the shifting bottleneck heuristic. The later is used to build the initial solution, and also at the re-optimisation phase of the algorithm. Whenever the tabu search cycle improves the best known solution, the procedure deletes the sequence of operations of all critical machines (machines with operations in the critical path). After shifting bottleneck rebuilds the solution, the tabu search is repeated. The tabu search module uses a dynamic management of three different neighbourhood structures and a tabu list of variable size, dependent of how many tabu iterations have been executed. The algorithm, that we name PM, was run on a Pentium 133MHz. Table 6.6 shows the comparison results between algorithms Tabu_VVI and PM. Quality values that win the comparison are shown in bold.

Table 6.6 Results by the best of all variants of Tabu_VVI and the algorithm of Pezzella and Merelli; in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

instances	Tabu_VVI		PM	
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
abz	**1.71**	81.5	2.23	151
ft	0	0.15	0	65
la01-05	0	0.03	0	9.8
la06-10	0	0.02	0	-
la11-15	0	0.04	0	-
la16-20	0	0.44	0	61.5
la21-25	**0**	7.94	0.1	115
la26-30	**0.17**	83.4	0.46	105
la31-35	0	0.27	0	-
la36-40	**0.05**	57.1	0.58	141
ta01-10	**0.24**	49.5	0.45	2175
ta11-20	**3.12**	177	3.47	2526
ta21-30	**5.96**	319	6.52	34910
ta31-40	**1.26**	221	1.92	14133
ta41-50	**5.47**	1016	6.04	11512

Comparison to the Tabu Search with Path-Relinking Along with the guided local search procedure of Balas and Vazacopoulos, and the tabu search with shifting bottleneck of Pezzella and Merelli, one other procedure, due to Nowicki and Smutnicki (Nowicki and Smutnicki 2005), forms the group of three procedures that are the best up to date methods applied to the job-shop scheduling problem. This last one being the state of the art for the job shop scheduling problem

The procedure of Nowicki and Smutnicki performs path-relinking between elite solutions found by a tabu search module. The solutions achieved by the path-relinking are then used as starting points for new cycles of the tabu search; the set of elite solutions is updated and the all process is repeated. We can say that the path-relinking works as the diversification strategy of the tabu search.

The algorithm uses a data structure specially designed for the application of this method to the job-shop scheduling problem. The instances of Taillard (Taillard 1993) were used to study the distribution of the local optima solutions in the solution space; and this study supported the design of this method. The algorithm, that we name NS, was run on a Pentium 900MHz. Unlike all other procedures, the computational times reported by the authors do not include the time needed to build the initial solutions. Table 6.7 shows the comparison results between algorithms Tabu_VVI and NS.

Table 6.7 Results by the best of all variants of Tabu_VVI and the algorithm of Nowicki and Smutnicki; in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

instances	Tabu_VVI		NS	
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
swv01-05	2.33	128	1.01	462
swv06-10	8.06	282	7.49	514
swv11-15	1.41	1855	0.51	360
yn	7	164	5.18	510
ta01-10	0.24	50	0.11	26
ta11-20	3.12	177	2.81	108
ta21-30	5.96	319	5.68	328
ta31-40	1.26	221	0.78	341
ta41-50	5.47	1016	4.7	975

6.5 Discussion on Tabu_VVI

We have developed a powerful, fast and innovative optimised search heuristic to solve combinatorial optimisation problems. It uses an exact technique from the operations research field to guide the search process of a metaheuristic. The procedure, named Tabu_VVI, uses the verification of violated valid inequalities as a diversification strategy of a tabu search procedure. The idea of this new method is to mimic the cuts in integer programming, letting the violated valid inequalities discard unattractive regions of the solution space and guide the search from a local optimal solution to a more quality region of the search space.

The procedure was illustrated with an application to the job-shop scheduling problem.

When developing this algorithm we were confronted with a challenge, related to the implementation of the verification of valid inequalities violated by partial solutions.

The valid inequalities are defined for every subset of operations processed on a machine. A problem with n jobs will have 2^n subsets of operations to each machine. We have decided not to inspect all the subsets but only the ones with more possibilities of concealing a violation of a valid inequality. The inspected subsets were built adding one by one the operations, according to its release dates and completion values.

We presented some computational results for a large set of benchmark instances, along with comparisons to other similar and successful works. Our new method, Tabu_VVI, always performs better than other methods that combine exact and heuristic procedures. It compares most favourably to two other leading methods for solving the job-shop scheduling problem; the guided local search of Balas and Vazacopoulos and the tabu search with shifting bottleneck of Pezzella and Merelli. When compared to the state of the art tabu search of Nowicki and Smutnicki, after running for approximately the same amount of time, Tabu_VVI achieves solutions with quality very close to theirs.

The description of this new method Tabu_VVI is presented in the paper *Optimised Search Heuristic Combining Valid Inequalities and Tabu Search* (Fernandes and Lourenço 2008).

7. Conclusions

Combinatorial optimisation problems are the subject of study of many practitioners with different scientific backgrounds, like operations research, artificial intelligence and computation sciences. While the work of researchers with an education on operations research has been mainly devoted to the study of the mathematical properties of the problems and the development of exact optimisation algorithms, researchers from artificial intelligence and computer science have developed metaheuristic methods especially focused on solving real-life applications of these problems. Practitioners of artificial intelligence and computer sciences do not invest in exact algorithms mainly because of assumptions like exact methods are too slow to be of any use to solve real life instances. Practitioners of operations research usually do not work on metaheuristics because of the assumption that these methods depend heavily on computational experiments to define their parameters values, lacking of any theoretical fundaments on their design and there by of no mathematical interest. It may also be true that while some may lack advanced programming skills others may have insufficient knowledge of advanced mathematical techniques.

Some practitioners have recently overcome the gap between exact and metaheuristic methods and developed new procedures that combine the two in order to solve combinatorial optimisation problems.

We have studied these new methods that combine metaheuristics and exact algorithms of combinatorial optimisation highlighting which procedures are combined; the specific way they work together and the problems to which they have been applied. Chapter 3 of this thesis is concerned with this subject. There we propose a designation for these methods – we call them Optimised Search Heuristics (OSH), and present a summary of the different ways of combining exact algorithms and metaheuristics versus the problems to which the methods are applied.

This mapping shows that there is plenty of room for new developments in this area. We are particularly interested in using exact techniques to guide the local search procedure of metaheuristics.

A paper entitled *Optimised Search Heuristics* which presented this study was produced and published (Fernandes and Lourenço 2007b).

To be able to capture the interest of the scientific community working on combinatorial optimisation problems, any new method must be tested on a problem of the NP-hard complexity class.

We have chosen to address scheduling problems and especially the job shop scheduling problem, famous for its difficulty both in theory and practice. Another reason for focusing on this problem to introduce the new method was the fact that its algebraic structure is well studied. Many properties have been proven that allow the description of valid inequalities, some of them defining facets of the convex hull of the set of feasible solutions.

Chapter 4 is dedicated to the presentation of scheduling problems; their definition, formulation and proven properties of the algebraic structure.

Chapters 5 and 6 present two proposed optimised search heuristic methods and describe the computational results for a large set of benchmark instances, along with comparisons to other similar and successful works.

We first developed a method which combines GRASP and branch-and-bound that we called GRASP_B&B and used it to solve the job shop scheduling problem. The method is a very fast procedure to find solutions of acceptable quality, ideal to work as initial solutions to other more elaborated methods. We have compared our GRASP_B&B to other procedures applied to the same problem, also used as producers of initial solutions. Namely, we compared it to another GRASP procedure (Binato, Hery et al. 2002) and to a procedure that uses branch-and-bound in the same way as we did – the shifting bottleneck procedure (Adams, Balas et al. 1988). GRASP_B&B compared most favourably to these methods, producing solutions of higher quality in less time.

A paper describing the new method GRASP_B&B entitled *A GRASP and Branch-and-Bound Metaheuristic for the Job Shop Scheduling*, was produced and published (Fernandes and Lourenço 2007).

Chapter 6 of this thesis contains the main contribution of this research work – the Tabu_VVI procedure. It is an optimised search heuristic that combines the verification of violated valid inequalities with a tabu search procedure. The method starts with a feasible solution produced with the method GRASP_B&B to which tabu search is applied, producing a “good” local optimum. This local optimal solution is then perturbed in order to continue search in the solution space. The solution is partially destroyed using a greedy randomised procedure to delete some of its elements. Then the method verifies the existence of valid inequalities violated by the partial solution. The reconstruction of a complete feasible solution is restricted by these violated valid inequalities, in the sense that they force some elements present on the partial solution out of the new complete solution. This way, the search path of the method is forced to jump to a different region of the solutions space. Hopefully to a more attractive region. This change in the search path direction is guided by the information of the algebraic structure of the problem present in the valid inequalities. And this is why we state that the search procedure of the tabu search metaheuristic is guided by the exact technique of verifying the existence of violated valid inequalities to discard some regions of the solution space.

The new method Tabu_VVI was applied to the job shop scheduling problem and compared to other methods that address it. Namely we compared Tabu_VVI to other methods that combine exact algorithms and metaheuristics and to the three leading procedures applied to the job shop scheduling problem: the Guided Local Search of Balas and Vazacopoulos (Balas and Vazacopoulos 1998), the Tabu Search combined with Shifting Bottleneck of Pezzella and Merelli (Pezzella and Merelli 2000) and the Tabu Search with Path Relinking of Nowicki and Smutnicki (Nowicki and Smutnicki 2005).

Tabu_VVI wins the comparison to other methods that combine exact algorithms and metaheuristics, always producing solutions with better quality and in less time. When compared to the procedure of Balas and Vazacopoulos and to the procedure of Pezzella and Merelli, our method achieves results very competitive with theirs. In the comparison to the state of the art procedure to solve job shop scheduling, the method of Nowicki and Smutnicki, our Tabu_VVI gets very close to their results.

This new method Tabu_VVI is described in our published paper *Optimised Search Heuristic Combining Valid Inequalities and Tabu Search* (Fernandes and Lourenço 2008).

We hope that these good results will encourage other researchers to close the gap between the areas of exact combinatorial optimisation methods and metaheuristics, taking advantage of the good characteristics of each of them.

When developing this research work we encountered one major challenge, related to the implementation of the verification of valid inequalities to the job shop scheduling problem violated by partial solutions.

The valid inequalities are derived from the subproblems of one machine scheduling and are defined for every subset of operations processed on a machine. As we all know the number of subsets of a set with say n elements is 2^n, an exponential number on the size of the problem, which poses an implementation problem. Instead of looking for all valid inequalities in all the possible subsets of operations on one machine, we have generated only a few of them. The subsets to be inspected were built adding to each of them the operations, one by one, according to some measures. The process of generating subsets to inspect is biased to subsets with more possibilities of concealing a violation of a valid inequality.

(Péridy and Rivreau 2005) proposes a new efficient enumerative method based on local adjustments that may be useful for inspecting all the subsets when looking for violated valid inequalities. A possible line of future work would be to discover how viable it is to implement such a method and to test if it would improve the efficiency of our Tabu_VVI method.

Different directions for proceeding with the line of research conducted in this thesis are:

a) Apply the new method Tabu_VVI to other hard scheduling problems, like the total weighted tardiness job shop problem or the generalised job shop problem. It would also be very interesting to apply the method to a real world instance.

b) Study the theoretical structure of the Tabu_VVI method, to design a general method that can be applied to other combinatorial optimisation problems.

c) Study the relationship between the different combinations of metaheuristics and exact procedures in OSH methods to evaluate the contribution of each one of them in the success of the OSH approach.

Annex A – Abstracts of Optimised Search Heuristics

Here we present a short abstract for each of the OSH procedures referenced in chapter 3, ordered by type of combination.

1.1 Sequential execution

Mixed Integer Problems

Hybrid Enumeration Strategies for Mixed Integer Programming (Pedroso 2004) The procedure solves the linear relaxation of the mixed integer problem and sets the integer values by random enumeration. It ends with a local search.

p-Median

Heuristic Concentration (Rosing and ReVelle 1997), (Rosing and ReVelle 1998), (Rosing 2000) This procedure is named heuristic concentration and is applied to the p-median problem. The first phase consists of doing multi random starts of a local search procedure and to choose a set of the best solutions found. In the second phase a branch-and-bound method is used to solve a p-median problem, where the possible facility locations are restricted to the ones chosen in the best local search solutions.

Steiner Tree

Combining a Memetic Algorithm with Integer Programming to Solve the Prize-Collecting Steiner Tree Problem (Klau, Ljubíc et al. 2004) This procedure is developed for the prize-collecting Steiner tree problem. A preprocessing phase reduces the graph. A memetic algorithm with problem-dependent operators and an exact local search procedure is applied to the reduced graph. Solving the integer programming problem of a minimum Steiner arborescence optimises the solutions found by the memetic algorithm. When solving the integer problem, not all the complicating constraints are included in the model, only the ones violated by the current solution.

Also not all variables are included, only the ones needed. So the procedure uses cutting and pricing. This is a very complicated algorithm.

Traveling Salesman

Finding Tours in the TSP (Applegate, Bixby et al. 1999) In a first phase of this procedure several tours for the traveling salesman problem are generated using an iterated Lin-Kernighan algorithm. The second phase uses a branch-and-cut algorithm to solve the problem defined over the subgraph with only the edges used by the tours found in the first phase.

Tour merging via branch-decomposition (Cook and Seymour 2003) In a first phase of this procedure several tours for the traveling salesman problem are generated using an iterated Lin-Kernighan algorithm. The second phase uses a dynamic programming algorithm to solve the problem defined over the subgraph with only the edges used by the tours found in the first phase.

Vehicle Routing

Effective Local search Algorithms for the Vehicle Routing Problem with General Time Window Constraints (Ibaraki, Kubo et al. 2001) The procedure is applied to the vehicle routing problem. Iterated local search determines the number of routes and the order in it, dynamic programming optimises the times of the routes.

Cutting Stock

Hybridizing Tabu Search with Optimization Techniques for Irregular Stock Cutting (Bennell and Dowsland 2001) The procedure is applied to the irregular cutting stock problem. Tabu search finds local optima of incomplete neighbourhoods. These solutions are improved by solving a linear program that uses the geometric concept of no fit polygon.

One Dimensional Cutting Stock Problem to Minimize the Number of Different Patterns (Umetani, Yagiura et al. 2003) Initial solutions for the iterated local search are built by heuristics based on the linear relaxation solution of the one-dimensional cutting stock problem.

Flow-Shop Scheduling

A meta-heuristic algorithm for a bi-criteria scheduling problem (Nagar, Heragu et al. 1995) This procedure is applied to the 2-machine flow shop scheduling problem. In a first phase the algorithm executes an incomplete branch-and-bound, and the partial solutions are stored along with their respective bounds. The second phase is a genetic algorithm that uses the information of the bounds to decide upon the mutation operator.

Parallel Machine Scheduling

Heuristic Optimization: A hybrid AI/OR approach (Clements, Crawford et al. 1997) Local search is used to find initial pre-solutions for the multi-job, parallel machine scheduling problem, with lateness and changeover costs. The priority heuristic with local search schedules blocks of jobs in each line of production. The integer programming problem is a set partitioning, where groups of schedules have to be chosen. Dantzig-Wolf solves the linear relaxation of the IP and then branch-and-bound finds integer solutions.

Knapsack

A Hybrid Approach for the 0-1 Multidimensional Knapsack problem (Vasquez and Hao 2001) The procedure is applied to the multidimensional knapsack problem. Linear relaxation is solved with the extra constraint of the sum of the variables being an integer k. Upper and lower limits for k are determined. Tabu search is executed for each one of the linear relaxation solutions sk. The neighbourhood is restricted to a small radius around sk.

A hybrid search combining interior point method and metaheuristics for 0-1 programming (Plateau, Tachat et al. 2002) This procedure is applied to the multiconstrained knapsack problem. It starts by executing an interior point method with early termination. Feasible solutions built by rounding and applying different ascendant heuristics will be the initial population for a scatter search method, with path-relinking.

Generalised Assignment

An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem (Feltl and Raidl 2004) This procedure is defined for the generalised assignment problem and is based on the algorithm of Chu and Beasley (1996). Initial solutions for the genetic algorithm are generated by randomly rounding the linear relaxation solution. The mutation operator consists of a heuristic procedure that preserves feasibility.

Markov Decision Processes

A Hybrid Genetic/ Optimization Algorithm for Finite-Horizon, Partially Observed Markov Decision Processes (Lin, Bean et al. 2004) This procedure is designed for the partially observed markov decision processes problem. The genetic algorithm generates an initial sub-set of witness points. A mixed integer program is solved to find the remaining ones.

2.1.1 Exactly solving relaxed problems

Job-Shop Scheduling

An approximate solution method for combinatorial optimisation (Tamura, Hirahara et al. 1994) This procedure is a genetic algorithm applied to the job-shop scheduling. The fitness of each individual, whose chromosomes represent each variable of the integer programming formulation, is the bound obtained solving lagrangean relaxations.

Knapsack

A Genetic Algorithm for the Multidimensional Knapsack Problem (Chu and Beasley 1998) This procedure is designed for the multiconstrained knapsack problem. Some elements of the population of the genetic algorithm are infeasible solutions generated by crossover and mutation operator. To recover feasibility of these solutions, the dual variables of the linear relaxation are used as weights in the surrogate relaxation of the multidimensional knapsack problem. A greedy heuristic based on the surrogate relaxation produces feasible solutions.

An improved genetic algorithm for the multiconstrained 0-1 knapsack problem (Raidl 1998) This procedure is designed for the multiconstrained knapsack problem. The initial population of the genetic algorithm is generated randomly setting the 0 /1 variables to one, with a probability given by its values on the linear relaxation solution. The repair operator to regain feasibility after crossover and mutation is also based on the solution values of the linear relaxation.

2.1.2 Exactly searching large neighbourhoods

Partitioning

Very large-scale neighbourhood search (Ahuja, Orlin et al. 2000), (Ahuja, Ergun et al. 2002) Very large neighbourhoods are exactly searched by network flow techniques, dynamic programming or by polynomial time solvable restrictions of the original problem. An application to the minimum spanning tree problem is described.

Steiner Tree

Combining a Memetic Algorithm with Integer Programming to Solve the Prize-Collecting Steiner Tree Problem (Klau, Ljubíc et al. 2004) This procedure is developed for the prize-collecting Steiner tree problem. A preprocessing phase reduces the graph. A memetic algorithm with problem-dependent operators and an exact local search procedure is applied to the reduced graph. Solving the integer programming problem of a minimum Steiner arborescence optimises the solutions found by the memetic algorithm. When solving the integer problem, not all the complicating constraints are included in the model, only the violated by the current solution. Also not all variables are included, only the ones needed. So the procedure uses cutting and pricing. This is a fairly complicated algorithm.

Traveling Salesman

A constraint programming framework for local search methods (Pesant and Gendreau 1996), (Pesant and Gendreau 1999) This procedure is developed within a constraint programming framework and applied to the traveling salesman with time windows. Each neighbourhood exploration is performed by branch-and-bound, defining elaborate local search moves.

Guided Local Search for Combinatorial Optimization Problems (Voudouris 1997), (Voudouris and Tsang 1999) Dynasearch[5] is used as a local search routine within a guided local search procedure applied to the traveling salesman problem.

Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation (Congram 2000) Dynamic programming finds the best neighbour in a local search neighbourhood of exponential size. A perturbation is performed on the solution and dynasearch is iterated. The procedure is applied to the traveling salesman problem.

Effective local and guided variable neighborhood search methods for the asymmetric traveling salesman problem (Burke, Cowling et al. 2001) This procedure is designed to the traveling salesman problem. The local search routine is based on splitting the original problem into small subproblems of connecting fixed subtours, which are solved to optimality by dynamic programming. The local search is embedded in a variable neighbourhood procedure.

Embedded local search approaches for routing optimization (Cowling and Keuthen 2005) The procedure is applied to the asymmetric traveling salesman problem. Local search uses a neighbourhood of 2 unconnected segments of the tour. Dynamic programming is used to optimally reconnect cities within the segments. The variable neighbourhood search version uses several k-opt neighbourhoods.

Vehicle Routing

Cycle transfers (Thompson and Orlin 1989), (Thompson and Psaraftis 1993) This procedure defines a local search procedure with a neighbourhood structure based on the cyclic transfer concept. The exponential sized neighbourhood is exactly explored

[5] Dynasearch – dynamic programming searchs an exponencial sized neighbourhood in polynomial time

defining appropriated auxiliary graphs and using dynamic programming. A variable depth search technique is employed. The procedure is applied to the vehicle routing problem.

One Machine Scheduling

An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem (Congram, Potts et al. 2002) Dynamic programming finds the best neighbour in a local search neighbourhood of exponential size. A perturbation is performed on the solution and dynasearch is iterated. The procedure is applied to the problem of scheduling a single machine with total weighted tardiness.

Iterated local search (Lourenço, Martin et al. 2002) A dynasearch is used as a local search routine inside an iterated local search procedure, applied to the single machine weighted tardiness problem.

2.1.3 Exactly solving subproblems

Mixed Integer

Tabu Search for Mixed Integer programming (Pedroso 2004b) The procedure is applied to mixed integer programming problems. Tabu search sets the values of integer variables and then a linear program is solved. Tabu search uses branch-and-bound as an intensification strategy.

Graph Colouring

Improving graph coloring with linear programming and genetic algorithms (Marino, Prugel-Bennett et al. 1999) The crossover of the genetic algorithm uses the optimal solution of the linear assignment formulation for the maximal sub-graph with zero clashes of the graph colouring problem.

Maximum Independent Set

An optimized crossover for the maximum independent set (Aggarwal, Orlin et al. 1997) This is a genetic procedure designed to the maximum independent set problem. In the recombination phase, the union of the features of two parent solutions defines an integer programming subproblem, which is solved to optimality.

Maximum Clique

Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems (Balas and Niehaus 1998) This is a genetic procedure designed to the maximum clique problem. In the crossover operator, a subproblem is defined by the union of the features of two parent solutions, which is then solved exactly by integer programming.

Network Design

A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem (Büdenbender, Grünert et al. 2000) This procedure is built for the network design problem. Each neighbouring solution of the local search is generated fixing the value of some variables and leaving the others free. The subproblem defined on the free subset of variables is solved to optimality by integer programming techniques.

Quadratic Assignment

Intensification neighbourhoods for local search methods (Mautor and Michelon 1997), (Mautor and Michelon 2001), (Mautor 2002) The MIMAUSA algorithm is designed to the quadratic assignment problem. The local search neighbourhood is defined deleting the value of some k variables. The correspondent subproblem of assigning values to those variables is exactly solved by integer programming.

Vehicle Routing

Using constraint programming and local search methods to solve vehicle routing problems (Shaw 1998) This tabu search procedure is designed for a vehicle routing problem. Branch-and-bound is used to exactly explore a partial neighbourhood structure, defined by a subproblem.

Job-Shop Scheduling

The Shifting Bottleneck Procedure for Job Shop Scheduling (Adams, Balas et al. 1988) The shifting bottleneck procedure is an iterated local search applied to the job shop scheduling problem, with a construction heuristic that uses a branch-and-bound to solve the subproblems of one machine with release and due dates.

A Computational Study of the Job-Shop Scheduling Problem (Applegate and Cook 1991) The local search type shuffle heuristic was built for the job shop scheduling problem. At each step the processing orders of the jobs on a small number of machines is fixed, and a branch-and-bound algorithm completes the schedule.

Disjunctive scheduling with task intervals (Caseau and Laburthe 1995) The local search procedure is applied to the job shop scheduling problem. The neighbourhood structure is defined by a subproblem that is exactly solved using constraint programming.

Guided Local Search with Shifting Bottleneck for Job Shop Scheduling (Balas and Vazacopoulos 1998) This is a guided local search, over a tree search structure, that reconstructs partially destroyed solutions for the job shop problem, using a branch-and-bound algorithm to exactly solve one machine subproblems.

One Machine Scheduling

The use of dynamic programming in genetic algorithms for permutation problems (Yagiura and Ibaraki 1996) This is a genetic algorithm for permutation problems. In the crossover operator, common chromosomes of two parent solutions are kept fixed and the free ones are optimised using dynamic programming.

Generalised Schwefel Function

Embedding Branch and Bound within Evolutionary Algorithms (Cotta and Troya 2003) Branch-and-bound is used to optimally complete a partial solution built with the recombination operator of the genetic algorithm. This procedure is applied to the following problems: generalised schwefel function, rulebase learning in mobile agents, design of a brachystochrone, k-epistatic minimal permutation.

2.1.4 Exact algorithms as decoders

Packing

Local search algorithms for the rectangle packing problem with general spatial costs (Imahori, Yagiura et al. 2003) Dynamic programming evaluates codified solutions found by local search by determining the optimal real solution that corresponds to the codified one. The codified solution is perturbed and local search is iterated. The procedure is applied to the rectangle packing problem.

Solving a Real-World Glass Cutting Problem (Puchinger, Raidl et al. 2004) The individuals of the genetic algorithms are coded solutions of the two-dimensional bin-packing problem with scheduling. Branch-and-bound is used to decode coded solutions. This procedure is developed for a real cutting glass problem.

Lot-sizing

A hybrid genetic algorithm to solve a lot-sizing and scheduling problem (Staggemeier, Clark et al. 2002) The procedure is applied to the lot-sizing problem. The genetic algorithm schedules products and linear programming optimises sizes of lots for a given schedule, determining the fitness value of each element of the population. A heuristic of the asymmetric TSP type is used within the genetic algorithm to re-optimise all changes produced by crossover or mutation.

2.1.5 Exact algorithms for strategic guidance of metaheuristics

Frequency Assignment - Quadratic Assignment

An ANTS heuristic for the frequency assignment problem (Maniezzo 1999), (Maniezzo and Carbonaro 2000) The procedure is an Ant Colony Optimisation metaheuristic that uses information from the linear relaxation and the values of the dual variables to determine the pheromones, which guide the construction of solutions. The procedure has been applied to problems like quadratic assignment (1999) and frequency assignment (2000).

Packing

Using tree bounds to enhance a genetic algorithm approach to two rectangle packing problems (Dowsland, Herbert et al. 2004) The representation of the individuals in the genetic algorithm is related to the search tree, as each position in the string corresponds to the choice of the branch at that level. Each individual corresponds to a path from the top of the tree to a terminal node. This way, bounds can be calculated to partial solutions, guiding crossover and mutation operators. This procedure is applied to the rectangle packing problem.

Job-Shop Scheduling

Combining large-step optimization with tabu-search: Application to the job-shop scheduling problem (Lourenço 1995), (Lourenço and Zwijnenburg 1996) The iterated local search procedure is applied to the job shop scheduling problem. In the perturbation phase, subproblems of one or two machines are solved by a branch-and-bound algorithm.

Optimisation of Continuous Problems

Tabu Search directed by direct local methods for nonlinear global optimization (Hedar and Fukushima 2004) The procedure is developed for the optimisation of continuous problems. The neighbourhood of the tabu search is generated according to extreme directions using the Nelder-Mead method and the pattern search strategy. The tabu list is managed by anti-cycling rules.

2.2.1 Metaheuristics for obtaining incumbent solutions and bounds

Packing

A hybrid improvement heuristic for the one-dimensional bin packing problem (Alvim, Ribeiro et al. 2003) The procedure is applied to the one-dimensional bin-packing problem. The related min-max problem is solved by greedy heuristics to find the number of bins. Upper and lower bounds are calculated using the algebraic structure of the problem. Solutions are determined solving the dual bin-packing problem heuristically. Tabu search transforms remaining infeasible solutions into feasible ones.

Lot-sizing

An integrated lagrangean relaxation - simulated annealing approach to the multi-level multi-item capacitated lot sizing problem (Ozdamar and Barbarosoglu 2000) Subproblems of the multi-level, multi-item, capacitated lot-sizing problem are derived by lagrangean relaxation. Solutions of these subproblems update lower bounds and lagrangean multipliers. A recursive heuristic is applied in order to restore capacity feasibility of the subproblems solutions and then simulated annealing is applied to find complete solutions, providing upper bounds. The procedure is repeated with the updated lagrangean multipliers.

Job-Shop Scheduling

Meta heuristics diversification of generalized job shop scheduling based upon mathematical programming techniques (Schaal, Fadil et al. 1999) Interior point method generates initial solutions of the linear relaxation. The genetic algorithm finds integer solutions. A cut is generated based on the integer solutions found and the interior point method is applied again to diversify the search. This procedure is defined for the generalised job shop problem.

Binary (0-1) Programming

A chunking based selection strategy for integrating meta-heuristics with branch and bound (Woodruff 1999) This is a branch-and-bound procedure that uses a chunking-based selection strategy to decide at each node of the tree whether or not a reactive tabu search is run to improve the incumbent solution.

2.2.2 Metaheuristics for column and cut generation

Graph Colouring

Constructive Genetic Algorithm and Column Generation: an Application to Graph Coloring (Filho and Lorena 2000) This procedure is applied to the graph colouring problem. The genetic algorithm is used with a given number of columns to approximately solve a weighted maximum independent set problem; which generates the initial pool of columns needed for the column generation process. Each column forms an independent set. Column generation solves the set covering formulation. The whole procedure is repeated with the number of columns minus one, until no improvement is found.

Packing

An Evolutionary Algorithm for Column Generation in Integer Programming: an Effective Approach for 2D Bin Packing (Puchinger and Raidl 2004b), (Puchinger and Raidl 2004c) The genetic algorithm is used within the branch-and-price procedure to solve the column generation. This procedure is applied to the 2D bin-packing problem.

2.2.3 Metaheuristics for strategic guidance of exact methods

Mixed Integer

Using a Hybrid Genetic-Algorithm/Branch and Bound Approach to Solve Feasibility and Optimization Integer Programming Problems (French, Robinson et al. 2001) This procedure is used to solve feasibility and optimisation integer programming problems and is inspired on the algorithm of Beasley and Chu (1996). Bounds of the branch-and-bound tree are found by relaxing integrality. The genetic algorithm builds integer solutions by relaxing constraint satisfaction and using information from the tree nodes to generate chromosomes. The solutions found by the genetic algorithm determine the new nodes of the tree to be examined. The algorithm is exact and was incorporated in commercial software XPRESS-MP.

Genetic Programming for Guiding Branch and Bound Search (Kostikas and Fragakis 2004) The genetic programming is used for evolving the best branching heuristic to each instance. Genetic programming "trains" during a first phase of branch-and-bound, finds the best branching heuristic and then branch-and-bound starts again with the learned strategy for branching. This procedure is applied to mixed integer programming problems.

p-Median -- Flow-Shop Scheduling

Recovering Beam Search: Enhancing the Beam Search Approach for Combinatorial Optimization Problems (Della-Croce, Ghirardi et al. 2004) Lagrangean relaxation is used to derive lower and upper bounds to the nodes of the limited branch-and-bound tree. The number of nodes per level is limited heuristically using valid and pseudo dominance conditions. The recovery step consists of performing a local search at the current node and determines the next node to be examined. This procedure is applied to the two machine flow shop scheduling and the uncapacitated p-median problems.

Parallel Machine Scheduling

Makespan minimization on unrelated parallel machines: a Recovering Beam Search approach (Ghirardi and Potts 2005) The Beam Search[6] procedure is applied to the scheduling problem with parallel machines. The neighbourhood of partial solutions is inspected by local search, recovering pruned solutions of a limited branch-and-bound tree.

[6] Beam search – a branch-and-bound procedure where the number of nodes per level (the beam) is limited heuristically

2.2.4 Applying the spirit of metaheuristics

Mixed Integer

Local Branching (Fischetti and Lodi 2003) This procedure, design to mixed integer problems, called local branching, is a branch-and-bound method with a branching strategy that determines the number of variables to remain unchanged, instead of specifying which variables to change. At each node of the branch-and-bound tree the commercial software Cplex is used to solve the sub MIP integer model.

Exploring relaxation induced neighborhoods to improve MIP solutions (Danna, Rothberg et al. 2005) Within each node of a branch-and-cut tree, the solution of the linear relaxation is used to define the neighbourhood of the current best feasible solution. The local search consists in solving the restricted MIP problem defined by the neighbourhood. The procedure is applied to mixed integer problems like job shop, network design and multicommodity routing.

Generalised Assignment

Stabilized Branch-and-cut-and-price for the Generalized Assignment Problem (Pigatti, Aragão et al. 2005) This procedure is developed for the general assignment problem. Upper bounds for the nodes of the search tree are obtained by solving a linear program that inspects a k-opt neighbourhood in polynomial time. Ellipsoidal cuts that define the neighbourhood are added to the linear problem. Ellipsoidal cuts are inspired in the path relinking idea.

Annex B – Computational Results for GRASP_B&B

Table B.1 Results by GRASP_B&B for Instances abz (Adams, Balas et al. 1988)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
abz5	10× 10		1234	1258 (1.94)	1312 (6.32)	1332 (7.94)	1358 (10.05)	1460 (18.31)	0.77	0.10
abz6	10× 10		943	952 (0.95)	978.75 (3.79)	997 (5.73)	1012.5 (7.37)	1078 (14.32)	0.77	0.31
abz7	15× 20		656	725 (10.52)	750.75 (14.44)	763 (16.31)	781 (19.05)	810 (23.48)	10.91	3.49
abz8	15× 20	647	669	734 (9.72)	767 (14.65)	780 (16.59)	797.25 (19.17)	837 (25.11)	10.52	1.89
abz9	15× 20	661	679	754 (11.05)	782.5 (15.24)	792 (16.64)	809 (19.15)	874 (28.72)	10.47	1.36

Table B.2 Results by GRASP_B&B for Instances ft (Fisher and Thompson 1963)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ft06	6× 6		55	**55** (0.00)	59 (7.27)	59 (7.27)	61 (10.91)	66 (20.00)	0.14	0.13
ft10	10× 10		930	970 (4.30)	1026.75 (10.40)	1046 (12.47)	1073.25 (15.40)	1144 (23.01)	1.00	0.58
ft20	20× 5		1165	1283 (10.13)	1304 (11.93)	1318 (13.13)	1365 (17.17)	1409 (20.94)	0.47	0.01

Table B.3 Results by GRASP_B&B for Instances la01-la10 (Lawrence 1984)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
la01	10× 5		666	**666** (0.00)	**666** (0.00)	**666** (0.00)	**666** (0.00)	694 (4.20)	0.17	0.002
la02	10× 5		655	667 (1.83)	712 (8.70)	722 (10.23)	722 (10.23)	835 (27.48)	0.16	0.04
la03	10× 5		597	605 (1.34)	605 (1.34)	640 (7.20)	701 (17.42)	701 (17.42)	0.22	0.01
la04	10× 5		590	607 (2.88)	610 (3.39)	648 (9.83)	648 (9.83)	672 (13.90)	0.17	0.01
la05	10× 5		593	**593** (0.00)	**593** (0.00)	**593** (0.00)	**593** (0.00)	**593** (0.00)	0.11	0.001
la06	15× 5		926	**926** (0.00)	**926** (0.00)	**926** (0.00)	**926** (0.00)	**926** (0.00)	0.17	0.002
la07	15× 5		890	**890** (0.00)	**890** (0.00)	**890** (0.00)	**890** (0.00)	936 (5.17)	0.20	0.002
la08	15× 5		863	**863** (0.00)	**863** (0.00)	880 (1.97)	921 (6.72)	976 (13.09)	0.30	0.01
la09	15× 5		951	**951** (0.00)	**951** (0.00)	**951** (0.00)	**951** (0.00)	953 (0.21)	0.28	0.003
la10	15× 5		958	**958** (0.00)	**958** (0.00)	**958** (0.00)	**958** (0.00)	**958** (0.00)	0.14	0.001

Table B.4 Results by GRASP_B&B for Instances la11-la20 (Lawrence 1984)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
la11	20× 5		1222	**1222** (0.00)	**1222** (0.00)	**1222** (0.00)	**1222** (0.00)	1284 (5.07)	0.27	0.003
la12	20× 5		1039	**1039** (0.00)	**1039** (0.00)	**1039** (0.00)	**1039** (0.00)	**1039** (0.00)	0.27	0.003
la13	20× 5		1150	**1150** (0.00)	**1150** (0.00)	**1150** (0.00)	**1150** (0.00)	1223 (6.35)	0.38	0.004
la14	20× 5		1292	**1292** (0.00)	**1292** (0.00)	**1292** (0.00)	**1292** (0.00)	**1292** (0.00)	0.22	0.002
la15	20× 5		1207	**1207** (0.00)	1240 (2.73)	1295 (7.29)	1295 (7.29)	1295 (7.29)	0.91	0.05
la16	10× 10		945	1012 (7.09)	1038.5 (9.89)	1049 (11.01)	1060 (12.17)	1099 (16.30)	0.74	0.02
la17	10× 10		784	787 (0.38)	813.75 (3.79)	836.5 (6.70)	864.25 (10.24)	950 (21.17)	0.77	0.08
la18	10× 10		848	854 (0.71)	879.25 (3.69)	895 (5.54)	924 (8.96)	1042 (22.88)	0.75	0.30
la19	10× 10		842	861 (2.26)	893.75 (6.15)	917 (8.91)	940.5 (11.70)	1020 (21.14)	0.97	0.46
la20	10× 10		902	920 (2.00)	960 (6.43)	976 (8.20)	1011.5 (12.14)	1080 (19.73)	0.81	0.08

Table B.5 Results by GRASP_B&B for Instances la21-la30 (Lawrence 1984)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
la21	15× 10		1046	1092 (4.40)	1154 (10.33)	1177.5 (12.57)	1210.25 (15.70)	1286 (22.94)	2.05	0.10
la22	15× 10		927	955 (3.02)	999 (7.77)	1029.5 (11.06)	1063.5 (14.72)	1192 (28.59)	1.80	0.99
la23	15× 10		1032	1049 (1.65)	1089.25 (5.55)	1111 (7.66)	1136 (10.08)	1268 (22.87)	1.89	1.74
la24	15× 10		935	971 (3.85)	1016 (8.66)	1030 (10.16)	1054.25 (12.75)	1104 (18.07)	1.84	0.63
la25	15× 10		977	1027 (5.12)	1082.75 (10.82)	1100 (12.59)	1122.25 (14.87)	1226 (25.49)	1.80	0.54
la26	20× 10		1218	1265 (3.86)	1321.75 (8.52)	1355 (11.25)	1376 (12.97)	1485 (21.92)	3.38	3.04
la27	20× 10		1235	1308 (5.91)	1375 (11.34)	1399 (13.28)	1431.25 (15.89)	1538 (24.53)	3.56	0.18
la28	20× 10		1216	1301 (6.99)	1360.75 (11.90)	1391 (14.39)	1413.25 (16.22)	1533 (26.07)	3.00	0.15
la29	20× 10		1152	1248 (8.33)	1312.75 (13.95)	1339 (16.23)	1379 (19.70)	1466 (27.26)	3.30	0.86
la30	20× 10		1355	1382 (1.99)	1432.75 (5.74)	1452.5 (7.20)	1477 (9.00)	1548 (14.24)	3.33	0.87

Table B.6 Results by GRASP_B&B for Instances la31-la40 (Lawrence 1984)

name	n×m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
la31	30×10		1784	**1784** (0.00)	1806.75 (1.28)	1829.5 (2.55)	1866.25 (4.61)	2006 (12.44)	7.02	0.07
la32	30×10		1850	**1850** (0.00)	1868.75 (1.01)	1906 (3.03)	1931 (4.38)	2024 (9.41)	6.24	0.56
la33	30×10		1719	**1719** (0.00)	1729.75 (0.63)	1756.5 (2.18)	1797 (4.54)	1872 (8.90)	7.91	1.27
la34	30×10		1721	**1721** (0.00)	1787 (3.83)	1812 (5.29)	1845.25 (7.22)	2025 (17.66)	8.28	3.81
la35	30×10		1888	**1888** (0.00)	1901 (0.69)	1923 (1.85)	1978.25 (4.78)	2232 (18.22)	5.69	0.28
la36	15×15		1268	1325 (4.50)	1375.75 (8.50)	1395.5 (10.06)	1423.25 (12.24)	1521 (19.95)	4.27	0.09
la37	15×15		1397	1479 (5.87)	1538.75 (10.15)	1565.5 (12.06)	1597.25 (14.33)	1642 (17.54)	4.80	4.03
la38	15×15		1196	1274 (6.52)	1354.75 (13.27)	1381.5 (15.51)	1397.75 (16.87)	1471 (22.99)	5.11	0.72
la39	15×15		1233	1309 (6.16)	1352.75 (9.71)	1374 (11.44)	1404.25 (13.89)	1468 (19.06)	4.45	2.98
la40	15×15		1222	1291 (5.65)	1347 (10.23)	1369 (12.03)	1398.5 (14.44)	1451 (18.74)	5.39	3.56

Table B.7 Results by GRASP_B&B for Instances orb01-orb10 (Applegate and Cook 1991)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
orb01	10× 10		1059	1145 (8.12)	1181.75 (11.59)	1198 (13.13)	1219.25 (15.13)	1335 (26.06)	0.99	0.03
orb02	10× 10		888	918 (3.38)	959.75 (8.08)	983 (10.70)	1013.25 (14.10)	1085 (22.18)	0.95	0.10
orb03	10× 10		1005	1098 (9.25)	1135.5 (12.99)	1155.5 (14.98)	1184.25 (17.84)	1289 (28.26)	1.02	0.34
orb04	10× 10		1005	1066 (6.07)	1120 (11.44)	1144.5 (13.88)	1183 (17.71)	1255 (24.88)	1.13	0.82
orb05	10× 10		887	911 (2.71)	966.75 (8.99)	1001 (12.85)	1014.25 (14.35)	1117 (25.93)	0.88	0.11
orb06	10× 10		1010	1050 (3.96)	1108 (9.70)	1134.5 (12.33)	1172 (16.04)	1282 (26.93)	1.05	0.48
orb07	10× 10		397	414 (4.28)	436.5 (9.95)	448 (12.85)	455 (14.61)	503 (26.70)	1.06	0.28
orb08	10× 10		899	945 (5.12)	975 (8.45)	999 (11.12)	1032.75 (14.88)	1125 (25.14)	1.03	0.31
orb09	10× 10		934	978 (4.71)	1003.75 (7.47)	1021 (9.31)	1053.75 (12.82)	1177 (26.02)	0.91	0.28
orb10	10× 10		944	991 (4.98)	1024.75 (8.55)	1040 (10.17)	1073 (13.67)	1232 (30.51)	0.84	0.23

Table B.8 Results by GRASP_B&B for Instances swv01-swv10 (Storer, Wu et al. 1992)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
swv01	20× 10		1407	1605 (14.07)	1688 (19.97)	1762 (25.23)	1806.75 (28.41)	1900 (35.04)	3.67	3.67
swv02	20× 10		1475	1601 (8.54)	1696 (14.98)	1729 (17.22)	1776.5 (20.44)	1940 (31.53)	3.27	0.29
swv03	20× 10	1369	1398	1582 (13.16)	1666.75 (19.22)	1704.5 (21.92)	1738.5 (24.36)	1964 (40.49)	3.49	1.50
swv04	20× 10	1450	1483	1655 (11.60)	1737.5 (17.16)	1772.5 (19.52)	1816.25 (22.47)	1949 (31.42)	4.00	2.72
swv05	20× 10		1424	1587 (11.45)	1660.75 (16.63)	1690 (18.68)	1718.25 (20.66)	1829 (28.44)	3.67	3.60
swv06	20× 15	1591	1678	1895 (12.93)	1975 (17.70)	2012.5 (19.93)	2064.25 (23.02)	2240 (33.49)	10.78	8.09
swv07	20× 15	1446	1620	1833 (13.15)	1881.75 (16.16)	1921 (18.58)	1953.75 (20.60)	2076 (28.15)	11.55	2.31
swv08	20× 15	1640	1763	2001 (13.50)	2103.25 (19.30)	2150 (21.95)	2190 (24.22)	2318 (31.48)	11.03	9.05
swv09	20× 15	1604	1663	1877 (12.87)	1984.75 (19.35)	2017.5 (21.32)	2088 (25.56)	2197 (32.11)	11.39	10.02
swv10	20× 15	1631	1767	1978 (11.94)	2053.5 (16.21)	2102 (18.96)	2145 (21.39)	2288 (29.49)	10.06	4.43

Table B.9 Results by GRASP_B&B for Instances swv11-swv20 (Storer, Wu et al. 1992)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
swv11	50× 10	2983	2991	3366 (12.54)	3454.25 (15.49)	3498.5 (16.97)	3574 (19.49)	4047 (35.31)	62.36	8.73
swv12	50× 10	2972	3003	3422 (13.95)	3520.75 (17.24)	3569 (18.85)	3621 (20.58)	4196 (39.73)	141.92	5.68
swv13	50× 10		3104	3527 (13.63)	3601.25 (16.02)	3654 (17.72)	3698.25 (19.14)	4143 (33.47)	54.98	20.34
swv14	50× 10		2968	3295 (11.02)	3362.25 (13.28)	3402.5 (14.64)	3469.5 (16.90)	4052 (36.52)	180.84	159.14
swv15	50× 10	2885	2904	3329 (14.63)	3458.5 (19.09)	3565 (22.76)	3634.25 (25.15)	3994 (37.53)	113.17	73.56
swv16	50× 10		2924	**2924** (0.00)	**2924** (0.00)	**2924** (0.00)	**2924** (0.00)	2962 (1.30)	9.67	0.10
swv17	50× 10		2794	**2794** (0.00)	**2794** (0.00)	2798 (0.14)	2828 (1.22)	2949 (5.55)	16.97	0.68
swv18	50× 10		2852	**2852** (0.00)	**2852** (0.00)	**2852** (0.00)	2879 (0.95)	2985 (4.66)	15.61	0.16
swv19	50× 10		2843	**2843** (0.00)	2864 (0.74)	2904 (2.15)	2972.5 (4.56)	3168 (11.43)	30.27	2.12
swv20	50× 10		2823	**2823** (0.00)	**2823** (0.00)	2846.5 (0.83)	2894.25 (2.52)	3045 (7.86)	17.39	0.87

Table B.10 Results by GRASP_B&B for Instances yn (Yamada and Nakano 1992)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
yn1	20× 20	826	888	955 (7.55)	996.75 (12.25)	1010.5 (13.80)	1031.25 (16.13)	1084 (22.07)	23.45	4.22
yn2	20× 20	861	909	987 (8.58)	1035.75 (13.94)	1047 (15.18)	1060 (16.61)	1133 (24.64)	25.38	12.43
yn3	20× 20	827	893	996 (11.53)	1029.75 (15.31)	1049 (17.47)	1068.5 (19.65)	1111 (24.41)	25.34	11.91
yn4	20× 20	918	968	1060 (9.50)	1117.75 (15.47)	1132 (16.94)	1158 (19.63)	1209 (24.90)	23.89	20.07

Table B.11 Results by GRASP_B&B for Instances ta01-ta10 (Taillard 1993)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta01	15× 15		1231	1332 (8.20)	1387 (12.67)	1413 (14.78)	1438.25 (16.84)	1556 (26.40)	5.56	0.28
ta02	15× 15		1244	1313 (5.55)	1368.75 (10.03)	1394.5 (12.10)	1426.75 (14.69)	1499 (20.50)	5.52	4.96
ta03	15× 15		1218	1278 (4.93)	1346.75 (10.57)	1370 (12.48)	1403.25 (15.21)	1488 (22.17)	5.41	4.11
ta04	15× 15		1175	1249 (6.30)	1309 (11.40)	1330.5 (13.23)	1360.25 (15.77)	1518 (29.19)	6.16	3.45
ta05	15× 15		1224	1310 (7.03)	1369 (11.85)	1393.5 (13.85)	1432 (16.99)	1579 (29.00)	5.81	0.87
ta06	15× 15		1238	1308 (5.65)	1362.75 (10.08)	1396 (12.76)	1422.5 (14.90)	1535 (23.99)	5.94	0.77
ta07	15× 15		1227	1299 (5.87)	1342 (9.37)	1364 (11.17)	1390.25 (13.30)	1549 (26.24)	5.14	2.31
ta08	15× 15		1217	1306 (7.31)	1371 (12.65)	1389.5 (14.17)	1414.25 (16.21)	1523 (25.14)	5.95	2.80
ta09	15× 15		1274	1395 (9.50)	1438 (12.87)	1465 (14.99)	1491 (17.03)	1614 (26.69)	6.11	1.59
ta10	15× 15		1241	1332 (7.33)	1387 (11.76)	1413 (13.86)	1438.25 (15.89)	1556 (25.38)	5.52	0.28

Table B.12 Results by GRASP_B&B for Instances ta11-ta20 (Taillard 1993)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta11	20× 15	1323	1361	1497 (9.99)	1571 (15.43)	1597.5 (17.38)	1626.25 (19.49)	1727 (26.89)	11.11	8.00
ta12	20× 15	1351	1367	1511 (10.53)	1576.75 (15.34)	1590.5 (16.35)	1623.25 (18.75)	1709 (25.02)	11.77	3.41
ta13	20× 15	1282	1342	1498 (11.62)	1559.5 (16.21)	1581.5 (17.85)	1618.25 (20.58)	1728 (28.76)	10.69	5.02
ta14	20× 15		1345	1439 (6.99)	1496.75 (11.28)	1527.5 (13.57)	1569 (16.65)	1692 (25.80)	11.59	2.32
ta15	20× 15	1304	1340	1511 (12.76)	1576.25 (17.63)	1602 (19.55)	1639 (22.31)	1732 (29.25)	12.44	4.35
ta16	20× 15	1302	1360	1486 (9.26)	1551.75 (14.10)	1571.5 (15.55)	1609.25 (18.33)	1677 (23.31)	11.20	5.94
ta17	20× 15		1462	1600 (9.44)	1661 (13.61)	1693.5 (15.83)	1713 (17.17)	1911 (30.71)	9.39	7.70
ta18	20× 15	1369	1396	1543 (10.53)	1623 (16.26)	1652 (18.34)	1677.25 (20.15)	1782 (27.65)	12.20	7.57
ta19	20× 15	1297	1335	1463 (9.59)	1542 (15.51)	1574 (17.90)	1616 (21.05)	1740 (30.34)	11.53	7.26
ta20	20× 15	1318	1351	1498 (10.88)	1549 (14.66)	1580 (16.95)	1617 (19.69)	1686 (24.80)	11.56	6.24

Table B.13 Results by GRASP_B&B for Instances ta21-ta20 (Taillard 1993)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta21	20× 20	1539	1644	1810 (10.10)	1894.5 (15.24)	1936 (17.76)	1970.25 (19.84)	2144 (30.41)	20.97	11.11
ta22	20× 20	1511	1600	1792 (12.00)	1832.75 (14.55)	1865 (16.56)	1903 (18.94)	1989 (24.31)	22.48	6.07
ta23	20× 20	1472	1557	1708 (9.70)	1768.75 (13.60)	1801 (15.67)	1839.25 (18.13)	1947 (25.05)	22.08	16.12
ta24	20× 20	1602	1647	1778 (7.95)	1864.75 (13.22)	1894.5 (15.03)	1925.25 (16.89)	2014 (22.28)	19.19	17.08
ta25	20× 20	1504	1595	1746 (9.47)	1830.75 (14.78)	1876 (17.62)	1913.5 (19.97)	1992 (24.89)	20.41	16.12
ta26	20× 20	1539	1645	1768 (7.48)	1863.75 (13.30)	1907 (15.93)	1950.25 (18.56)	2027 (23.22)	17.84	2.68
ta27	20× 20	1616	1680	1839 (9.46)	1923.75 (14.51)	1954 (16.31)	1988.25 (18.35)	2149 (27.92)	19.84	17.86
ta28	20× 20	1591	1614	1755 (8.74)	1837.5 (13.85)	1871 (15.92)	1908.75 (18.26)	2016 (24.91)	22.31	19.19
ta29	20× 20	1514	1625	1717 (5.66)	1835.75 (12.97)	1864 (14.71)	1898.25 (16.82)	2012 (23.82)	20.16	6.25
ta30	20× 20	1473	1584	1737 (9.66)	1800.75 (13.68)	1827.5 (15.37)	1852.5 (16.95)	1960 (23.74)	17.55	6.49

Table B.14 Results by GRASP_B&B for Instances ta31-ta40 (Taillard 1993)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta31	30× 15		1764	1976 (12.02)	2059.75 (16.77)	2099.5 (19.02)	2135 (21.03)	2322 (31.63)	30.61	14.39
ta32	30× 15	1774	1796	2029 (12.97)	2132.5 (18.74)	2165.5 (20.57)	2205 (22.77)	2356 (31.18)	30.61	23.88
ta33	30× 15	1778	1793	2070 (15.45)	2171.25 (21.10)	2204 (22.92)	2265 (26.32)	2336 (30.28)	31.92	18.20
ta34	30× 15	1828	1829	2024 (10.66)	2114.25 (15.60)	2156 (17.88)	2186 (19.52)	2287 (25.04)	33.67	32.33
ta35	30× 15		2007	2093 (4.29)	2174.25 (8.33)	2208 (10.01)	2250.25 (12.12)	2454 (22.27)	35.30	13.41
ta36	30× 15		1819	2040 (12.15)	2124.5 (16.79)	2153.5 (18.39)	2204.25 (21.18)	2307 (26.83)	30.53	3.97
ta37	30× 15	1771	1778	1967 (10.63)	2099.75 (18.10)	2136 (20.13)	2184.5 (22.86)	2408 (35.43)	29.03	15.39
ta38	30× 15		1673	1913 (14.35)	1976.75 (18.16)	2003.5 (19.75)	2046.25 (22.31)	2188 (30.78)	34.00	12.58
ta39	30× 15		1795	1966 (9.53)	2069.25 (15.28)	2107.5 (17.41)	2144 (19.44)	2321 (29.30)	31.38	13.49
ta40	30× 15	1631	1674	1931 (15.35)	2012 (20.19)	2047.5 (22.31)	2077.25 (24.09)	2218 (32.50)	33.99	0.34

Table B.15 Results by GRASP_B&B for Instances ta41-ta50 (Taillard 1993)

name	$n \times m$	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta41	30×20	1859	2018	2348 (16.35)	2413.75 (19.61)	2458 (21.80)	2494 (23.59)	2638 (30.72)	61.11	0.61
ta42	30×20	1867	1956	2206 (12.78)	2301.75 (17.68)	2341 (19.68)	2374.25 (21.38)	2513 (28.48)	62.33	49.24
ta43	30×20	1809	1859	2155 (15.92)	2228.5 (19.88)	2254 (21.25)	2294.5 (23.43)	2395 (28.83)	72.70	50.89
ta44	30×20	1927	1984	2300 (15.93)	2382.5 (20.09)	2418 (21.88)	2466 (24.29)	2577 (29.89)	64.80	11.66
ta45	30×20	1997	2000	2295 (14.75)	2358 (17.90)	2380 (19.00)	2410.25 (20.51)	2581 (29.05)	70.83	34.71
ta46	30×20	1940	2021	2314 (14.50)	2399.5 (18.73)	2438 (20.63)	2481 (22.76)	2660 (31.62)	64.11	25.64
ta47	30×20	1789	1903	2151 (13.03)	2260.5 (18.79)	2299.5 (20.84)	2345.75 (23.27)	2443 (28.38)	63.99	63.35
ta48	30×20	1912	1952	2222 (13.83)	2325.5 (19.13)	2360 (20.90)	2407.5 (23.34)	2565 (31.40)	63.22	60.06
ta49	30×20	1915	1968	2250 (14.33)	2349.5 (19.39)	2390 (21.44)	2425 (23.22)	2560 (30.08)	65.27	11.75
ta50	30×20	1807	1928	2264 (17.43)	2347.5 (21.76)	2387 (23.81)	2431 (26.09)	2633 (36.57)	63.30	13.29

Table B.16 Results by GRASP_B&B for Instances ta51-ta60 (Taillard 1993)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta51	50× 15		2760	3001 (8.73)	3149 (14.09)	3227 (16.92)	3294 (19.35)	3587 (29.96)	139.88	65.74
ta52	50× 15		2756	2940 (6.68)	3136 (13.79)	3216 (16.69)	3276.5 (18.89)	3590 (30.26)	128.36	119.37
ta53	50× 15		2717	2895 (6.55)	2998 (10.34)	3031.5 (11.58)	3072.5 (13.08)	3219 (18.48)	116.94	87.70
ta54	50× 15		2839	2914 (2.64)	3024.75 (6.54)	3092.5 (8.93)	3133.75 (10.38)	3280 (15.53)	112.55	15.76
ta55	50× 15		2679	2988 (11.53)	3133 (16.95)	3178 (18.63)	3247.5 (21.22)	3440 (28.41)	144.66	56.42
ta56	50× 15		2781	2966 (6.65)	3122.75 (12.29)	3167 (13.88)	3230.25 (16.15)	3437 (23.59)	131.38	3.94
ta57	50× 15		2943	3101 (5.37)	3213.5 (9.19)	3256.5 (10.65)	3320.5 (12.83)	3485 (18.42)	106.42	27.67
ta58	50× 15		2885	3103 (7.56)	3214.5 (11.42)	3273 (13.45)	3326.25 (15.29)	3438 (19.17)	146.91	104.30
ta59	50× 15		2655	2940 (10.73)	3038.25 (14.44)	3090 (16.38)	3139.5 (18.25)	3294 (24.07)	124.30	119.33
ta60	50× 15		2723	2921 (7.27)	3048.5 (11.95)	3104 (13.99)	3160 (16.05)	3339 (22.62)	121.80	86.48

Table B.17 Results by GRASP_B&B for Instances ta61-ta70 (Taillard 1993)

name	n× m	LB	UB	min	Q1	Q2	Q3	max	ttime (s)	btime (s)
ta61	50× 20		2868	3258 (13.60)	3369.5 (17.49)	3424 (19.39)	3484 (21.48)	3649 (27.23)	285.50	216.98
ta62	50× 20	2869	2872	3306 (15.11)	3444 (19.92)	3493.5 (21.64)	3544 (23.40)	3730 (29.87)	311.99	293.27
ta63	50× 20		2755	3130 (13.61)	3218.75 (16.83)	3273 (18.80)	3324.25 (20.66)	3487 (26.57)	315.13	302.52
ta64	50× 20		2702	3008 (11.32)	3180.25 (17.70)	3230 (19.54)	3287.75 (21.68)	3431 (26.98)	289.84	153.62
ta65	50× 20		2725	3104 (13.91)	3242.5 (18.99)	3286 (20.59)	3339 (22.53)	3569 (30.97)	323.23	42.02
ta66	50× 20		2845	3198 (12.41)	3320 (16.70)	3361 (18.14)	3415.5 (20.05)	3585 (26.01)	317.16	63.43
ta67	50× 20		2825	3209 (13.59)	3339 (18.19)	3389.5 (19.98)	3435.25 (21.60)	3578 (26.65)	273.34	46.47
ta68	50× 20		2784	3133 (12.54)	3235.75 (16.23)	3283 (17.92)	3337.5 (19.88)	3542 (27.23)	268.78	223.09
ta69	50× 20		3071	3366 (9.61)	3447.5 (12.26)	3517 (14.52)	3576 (16.44)	3739 (21.75)	259.17	33.69
ta70	50× 20		2995	3449 (15.16)	3528.75 (17.82)	3582 (19.60)	3633.25 (21.31)	3815 (27.38)	279.52	58.70

Annex C – Computational Results for Tabu_VVI

C.1 Computational Results per Variant of Tabu_VVI

Table C.1 Results by Tabu_VVI: variants tabu_mvfct, tabu_mv_bb and tabu_mv2_bb, for all groups of instances, in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

sets of	tabu_mvfct		tabu_mv_bb		tabu_mv2_bb	
instances	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
abz	1.96	101.25	**1.93**	61.02	2.02	91.79
ft	0	0.96	0	0.58	0.29	8.41
la01-05	0	0.08	0	0.12	0.15	2.71
la06-10	0	0.03	0	0.03	0	1.00
la11-15	0	0.04	0	0.05	0	1.05
la16-20	0	2.67	0	1.67	0	0.18
la21-25	0.18	22.92	**0.06**	14.80	0.17	0.08
la26-30	0.28	67.26	**0.26**	40.88	0.61	0.03
la31-35	0	0.39	0	0.39	0	0.04
la36-40	0.39	65.02	**0.22**	33.50	0.48	0.05
orb	0.24	8.59	0.09	14.13	0.20	19.19
swv01-05	3.14	44.28	2.93	120.43	**2.74**	136.59
swv06-10	8.94	228.20	9.51	204.27	9.33	184.44
swv11-15	2.00	1045.08	2.03	825.21	1.96	1177.78
swv16-20	0	1.62	0	1.64	0	2.94
yn	7.71	341.02	7.91	73.61	7.42	336.11
ta01-10	0.74	37.78	0.81	67	0.75	124.58
ta11-20	3.66	101.22	3.70	86.60	3.70	307.70
ta21-30	6.57	289.45	6.60	269.97	6.56	479.69
ta31-40	1.69	386.45	**1.60**	258.49	1.92	463.43
ta41-50	6.12	1565.28	**5.88**	559.71	6.48	792.06
# best	42+13		42+13		42+5	
# min	42+21		42+25		42+15	
# only min	7		6		7	
sum RE_{LB}	305.38		303.21		313.97	
sum time	33148.74		**19375.1**		31840.4	

Table C.2 Results by Tabu_VVI: variants tabu_mvinf_bb, tabu_mvinf and tabu_mvinf_ls2, for all groups of instances, in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

sets of instances	tabu_mvinf_bb		tabu_mvinf		tabu_mvinf_ls2	
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
abz	**1.93**	98.82	2.11	63.77	2.38	70.58
ft	0.25	10.61	0	11.72	0	3.47
la01-05	0	0.29	0	0.12	0	0.52
la06-10	0	0.02	0	0.02	0	0.02
la11-15	0	0.05	0	0.04	0	0.04
la16-20	0	2.27	0	1.79	0	4.98
la21-25	0.10	27.16	0.11	23.13	0.30	14.59
la26-30	0.32	19.75	0.29	54.12	0.44	18.21
la31-35	0	0.39	0	0.38	0	0.28
la36-40	0.47	36.55	0.47	22.68	0.37	66.54
orb	0.24	11.63	0.23	7.00	0.22	19.09
swv01-05	2.91	86.32	2.89	88.05	3.30	66.16
swv06-10	9.65	160.14	**8.89**	336.94	9.59	127.20
swv11-15	1.85	1178.17	**1.78**	1734.51	3.38	1222.52
swv16-20	0	1.60	0	1.58	0	1.67
yn	7.81	160.66	7.49	339.33	8.09	168.35
ta01-10	0.97	70.11	0.63	77.72	**0.44**	126.27
ta11-20	3.65	132.14	**3.47**	54.20	3.90	137.72
ta21-30	6.63	281.16	6.51	319.27	6.53	253.49
ta31-40	1.69	288.37	1.79	230.90	2.09	144.51
ta41-50	5.97	542.74	6.04	650.87	6.34	543.96
# best	42+5		42+7		42+11	
# min	42+11		42+20		42+16	
# only min	3		6		2	
sum RE_{LB}	309.66		**299.34**		326.31	
sum time	21993.68		26427.78		20900.69	

Table C.3 Results by Tabu_VVI: variants tabu_mvinf_ls2_bb, tabu_mvhp_bb and tabu_mvinfhp_bb, for all groups of instances, in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

sets of instances	tabu_mvinf_ls2_bb		tabu_mvhp_bb		tabu_mvinfhp_bb	
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
abz	2.05	41.15	1.99	31.38	2.02	45.50
ft	0	15.28	0	16.03	0.22	22.96
la01-05	0	0.14	0	0.65	0	0.81
la06-10	0	0.02	0	0.03	0	0.03
la11-15	0	0.05	0	0.05	0	0.05
la16-20	0	1.89	0	1.19	0	1.37
la21-25	0.30	17.90	0.13	28.93	0.17	38.57
la26-30	0.37	39.72	0.44	29.31	0.38	37.18
la31-35	0	0.28	0	0.28	0	0.28
la36-40	0.47	77.70	0.35	117.26	0.44	82.79
orb	**0.08**	10.09	0.13	9.08	0.12	14.27
swv01-05	3.14	111.25	3.67	58.60	3.63	106.14
swv06-10	9.57	230.26	9.43	272.72	9.61	459.59
swv11-15	3.13	1801.37	3.57	938.57	3.57	1927.01
swv16-20	0	1.69	0	1.65	0	1.67
yn	7.67	175.71	7.70	385.95	7.65	517.77
ta01-10	0.67	58.97	0.86	61.24	0.71	94.45
ta11-20	3.84	113.06	3.80	155.71	3.65	200.22
ta21-30	6.68	311.53	6.58	353.00	6.50	404.12
ta31-40	1.78	438.35	1.94	483.48	1.94	734.99
ta41-50	6.12	678.36	6.22	1040.26	6.34	1657.02
# best	42+10		42+11		42+10	
# min	42+17		42+16		42+15	
# only min	3		2		1	
sum RE_{LB}	317.58		324.05		323.01	
sum time	28469.35		30022.66		46695.71	

Table C.4 Results by Tabu_VVI: variants tabu_mvinfhp, tabu_mv-2infhp_bb and tabu_mv-2infhp, for all groups of instances, in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

sets of instances	tabu_mvinfhp		tabu_mv-2infhp_bb		tabu_mv-2infhp	
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
abz	2.05	234.93	2.31	126.66	1.99	243.72
ft	0	10.20	0	19.33	0	20.27
la01-05	0.21	0.76	0.31	0.65	0.31	0.37
la06-10	0	0.02	0	0.03	0	0.02
la11-15	0	0.05	0	0.05	0	0.05
la16-20	0	1.94	0	3.70	0	4.07
la21-25	0.15	33.13	0.47	11.24	0.32	24.81
la26-30	0.51	59.45	0.51	25.89	0.57	33.67
la31-35	0	0.28	0	0.28	0	0.28
la36-40	0.32	94.22	0.37	115.45	0.34	86.66
orb	0.11	22.88	0.19	13.37	0.38	12.22
swv01-05	3.78	97.06	3.66	87.37	3.88	154.34
swv06-10	9.44	374.73	9.44	325.15	9.69	359.73
swv11-15	3.58	916.70	3.83	852.87	4.67	695.72
swv16-20	0	1.68	0	1.66	0	1.68
yn	7.55	535.12	**7.27**	884.28	7.40	331.62
ta01-10	0.72	66.52	0.53	116.72	1.07	89.92
ta11-20	3.93	175.50	3.91	171.58	3.74	199.32
ta21-30	6.67	331.22	6.75	305.97	6.77	220.15
ta31-40	1.76	346.80	2.50	288.25	2.63	504.83
ta41-50	6.21	770.22	6.44	1052.20	6.45	1395.33
# best	42+9		42+7		42+5	
# min	42+16		42+12		42+8	
# only min	2		1		2	
sum RE_{LB}	324.56		336.69		348.78	
sum time	28377.25		30830.93		33630.7	

Table C.5 Results by Tabu_VVI: variants tabuls_mvinfhp_bb, tabuls_mvinfhp and tabuls_mv_infhp_bb, for all groups of instances, in average percentage of the relative error to the lower bound, and the average time to the best, in seconds.

sets of instances	tabuls_mvinfhp_bb		tabuls_mvinfhp		tabuls_mv_infhp_bb	
	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$	$avg(RE_{LB})$	$avg(time)$
abz	2.14	213.46	2.05	135.99	1.99	168.58
ft	0.39	6.22	0	11.42	0.39	11.81
la01-05	0	0.26	0	0.27	0	0.24
la06-10	0	0.02	0	0.03	0	0.03
la11-15	0	0.04	0	0.05	0	0.05
la16-20	0	1.46	0	1.32	0.10	2.62
la21-25	0.26	36.28	0.15	54.33	0.19	32.25
la26-30	0.43	75.97	0.51	95.08	0.73	31.35
la31-35	0	0.28	0	0.28	0	0.28
la36-40	0.28	124.86	0.47	80.35	0.49	120.56
orb	0.20	16.09	0.21	8.76	0.30	13.71
swv01-05	3.30	74.40	3.22	242.76	3.01	168.02
swv06-10	9.62	274.74	9.29	433.36	9.26	351.11
swv11-15	3.85	1026.71	3.37	1288.44	3.53	2292.86
swv16-20	0	1.66	0	1.65	0	1.69
yn	7.79	546.28	8.29	177.33	7.73	369.16
ta01-10	0.76	112.56	0.74	109.28	0.73	78.47
ta11-20	3.65	240.18	3.70	192.82	3.84	137.21
ta21-30	6.75	279.74	6.56	482.34	**6.49**	211.82
ta31-40	1.86	567.03	1.93	672.53	1.98	138.78
ta41-50	6.11	1487.36	6.17	602.63	6.20	1002.99
# best	42+6		42+6		42+5	
# min	42+9		42+10		42+11	
# only min	1		4		3	
sum RE_{LB}	325.08		321.47		324.08	
sum time	38384.13		33096.74		33190.08	

C.2 Computational Results of Tabu_VVI per Instance

Table C.6 Best Results by Tabu_VVI for Instances abz (Adams, Balas et al. 1988)

instance	size	bk_UB	Tabu_VVI	time	# variants
abz5	10×10	1234	**1234**	1.03	19
abz6	10×10	943	**943**	≅ 0	35
abz7	20×15	656	663	272.41	1
abz8	20×15	*669*	672	3.86	2
abz9	20×15	*679*	685	129.98	3

Table C.7 Best Results by Tabu_VVI for Instances ft (Fisher and Thompson 1963)

instance	size	bk_UB	Tabu_VVI	time	# variants
ft06	6×6	55	**55**	≅ 0	35
ft10	10×10	930	**930**	0.45	21
ft20	20×5	1165	**1165**	≅ 0	32

Table C.8 Best Results by Tabu_VVI for Instances la01-la10 (Lawrence 1984)

instance	size	bk_UB	Tabu_VVI	time	# variants
la01	10×5	666	**666**	≅ 0	35
la02	10×5	655	**655**	≅ 0	30
la03	10×5	597	**597**	0.08	35
la04	10×5	590	**590**	0.06	35
la05	10×5	593	**593**	≅ 0	35
la06	15×5	926	**926**	0.02	35
la07	15×5	890	**890**	0.02	35
la08	15×5	863	**863**	0.03	35
la09	15×5	951	**951**	0.03	35
la10	15×5	958	**958**	0.02	35

Table C.9 Best Results by Tabu_VVI for Instances la11-la20 (Lawrence 1984)

instance	size	bk_UB	Tabu_VVI	time	# variants
la11	20×5	1222	**1222**	0.03	35
la12	20×5	1039	**1039**	0.02	35
la13	20×5	1150	**1150**	0.03	35
la14	20×5	1292	**1292**	0.02	35
la15	20×5	1207	**1207**	0.09	35
la16	10×10	945	**945**	0.75	35
la17	10×10	784	**784**	0.44	35
la18	10×10	848	**848**	≅ 0	35
la19	10×10	842	**842**	0.36	30
la20	10×10	902	**902**	0.66	33

Table C.10 Best Results by Tabu_VVI for Instances la21-la30 (Lawrence 1984)

instance	size	bk_UB	Tabu_VVI	time	# variants
la21	15×10	1046	**1046**	26.13	8
la22	15×10	927	**927**	3.95	23
la23	15×10	1032	**1032**	≅ 0	35
la24	15×10	935	**935**	1.95	1
la25	15×10	977	**977**	7.69	6
la26	20×10	1218	**1218**	≅ 0	35
la27	20×10	1235	**1235**	249.80	1
la28	20×10	1216	**1216**	2.53	34
la29	20×10	1153	1163	164.64	1
la30	20×10	1355	**1355**	≅ 0	35

Table C.11 Best Results by Tabu_VVI for Instances la31-la40 (Lawrence 1984)

instance	size	bk_UB	Tabu_VVI	time	# variants
la31	30×10	1784	**1784**	≅ 0	35
la32	30×10	1850	**1850**	0.78	35
la33	30×10	1719	**1719**	≅ 0	35
la34	30×10	1721	**1721**	≅ 0	35
la35	30×10	1888	**1888**	0.55	35
la36	15×15	1268	**1268**	2.16	15
la37	15×15	1397	**1397**	68.17	1
la38	15×15	1196	**1196**	154.42	1
la39	15×15	1233	**1233**	30.61	12
la40	15×15	1222	1225	30.06	2

Table C.12 Best Results by Tabu_VVI for Instances orb01-orb10 (Applegate and Cook 1991)

instance	size	bk_UB	Tabu_VVI	time	# variants
orb01	10×10	1059	**1059**	2.47	32
orb02	10×10	888	**888**	11.17	4
orb03	10×10	1005	**1005**	≅ 0	16
orb04	10×10	1005	**1005**	8.33	9
orb05	10×10	887	**887**	12.56	8
orb06	10×10	1010	**1010**	2.42	15
orb07	10×10	397	**397**	0.97	35
orb08	10×10	899	**899**	≅ 0	14
orb09	10×10	934	**934**	4.19	28
orb10	10×10	944	**944**	0.88	35

Table C.13 Best Results by Tabu_VVI for Instances swv01-swv10 (Storer, Wu et al. 1992)

instance	size	bk_UB	Tabu_VVI	time	# variants
swv01	20×10	1407	1433	113.89	1
swv02	20×10	1475	1485	143.23	2
swv03	20×10	*1398*	1430	93.89	1
swv04	20×10	*1483*	1492	105.50	1
swv05	20×10	1424	1449	183.03	1
swv06	20×15	*1678*	1700	417.69	1
swv07	20×15	*1620*	1624	423.06	1
swv08	20×15	*1763*	1792	83.95	1
swv09	20×15	*1663*	1675	382.45	1
swv10	20×15	*1767*	**1765**	101.06	1

Table C.14 Best Results by Tabu_VVI for Instances swv11-swv20 (Storer, Wu et al. 1992)

instance	size	bk_UB	Tabu_VVI	time	# variants
swv11	50×10	*2991*	3011	2138.31	1
swv12	50×10	*3003*	3041	2309.63	1
swv13	50×10	3104	3129	2023.23	1
swv14	50×10	2968	2979	774.42	1
swv15	50×10	*2904*	2961	2027.30	1
swv16	50×10	2924	**2924**	1.09	35
swv17	50×10	2794	**2794**	1.13	35
swv18	50×10	2852	**2852**	1.14	35
swv19	50×10	2843	**2843**	2.70	35
swv20	50×10	2823	**2823**	1.86	35

Table C.15 Best Results by Tabu_VVI for Instances yn (Yamada and Nakano 1992)

instance	size	bk_UB	Tabu_VVI	time	# variants
yn1	20×20	*888*	890	121.45	4
yn2	20×20	*909*	911	158.27	1
yn3	20×20	*893*	897	163.81	1
yn4	20×20	*968*	973	212.25	2

Table C.16 Best Results by Tabu_VVI for Instances ta01-ta10 (Taillard 1993)

instance	size	bk_UB	Tabu_VVI	time	# variants
ta01	15×15	1231	1244	119.11	2
ta02	15×15	1244	**1244**	20.99	19
ta03	15×15	1218	**1218**	75.49	2
ta04	15×15	1175	**1175**	34.88	1
ta05	15×15	1224	1228	7.98	3
ta06	15×15	1238	1241	10.05	6
ta07	15×15	1227	1228	10.81	28
ta08	15×15	1217	**1217**	26.11	9
ta09	15×15	1274	1280	70.69	8
ta10	15×15	1241	1244	119.14	2

Table C.17 Best Results by Tabu_VVI for Instances ta11-ta20 (Taillard 1993)

instance	size	bk_UB	Tabu_VVI	time	# variants
ta11	20×15	*1361*	1370	380.42	2
ta12	20×15	*1367*	1374	24.41	2
ta13	20×15	*1342*	1359	33.75	3
ta14	20×15	1345	**1345**	14.81	16
ta15	20×15	*1340*	1356	101.67	1
ta16	20×15	*1360*	1365	103.84	1
ta17	20×15	1462	1477	88.91	2
ta18	20×15	*1396*	1417	28.16	1
ta19	20×15	*1335*	1343	51.75	3
ta20	20×15	*1351*	1358	944.70	1

Table C.18 Best Results by Tabu_VVI for Instances ta21-ta30 (Taillard 1993)

instance	size	bk_UB	Tabu_VVI	time	# variants
ta21	20×20	*1644*	1650	332.92	1
ta22	20×20	*1600*	1609	605.67	2
ta23	20×20	*1557*	1568	579.52	2
ta24	20×20	*1647*	1649	253.78	2
ta25	20×20	*1595*	1599	171.79	1
ta26	20×20	*1645*	1661	570.95	1
ta27	20×20	*1680*	1686	51.91	1
ta28	20×20	*1614*	1619	201.19	3
ta29	20×20	*1625*	1629	151.88	4
ta30	20×20	*1584*	1599	270.59	2

Table C.19 Best Results by Tabu_VVI for Instances ta31-ta40 (Taillard 1993)

instance	size	bk_UB	Tabu_VVI	time	# variants
ta31	30×15	1764	1766	61.22	5
ta32	30×15	*1796*	1818	393.02	1
ta33	30×15	*1793*	1812	271.77	2
ta34	30×15	*1829*	1835	806.25	1
ta35	30×15	2007	**2007**	≅ 0	35
ta36	30×15	1819	1825	71.97	3
ta37	30×15	*1778*	1795	152.06	1
ta38	30×15	1673	1688	107.59	1
ta39	30×15	1795	1806	256.78	1
ta40	30×15	*1674*	1704	85.56	1

Table C.20 Best Results by Tabu_VVI for Instances ta41-ta50 (Taillard 1993)

instance	size	bk_UB	Tabu_VVI	time	# variants
ta41	30×20	*2018*	2035	3020.31	1
ta42	30×20	*1956*	1974	146.45	1
ta43	30×20	*1859*	1890	532.92	2
ta44	30×20	*1984*	2009	1257.19	1
ta45	30×20	*2000*	2014	857.19	1
ta46	30×20	*2021*	2045	316.70	1
ta47	30×20	*1903*	1933	1361.02	2
ta48	30×20	*1952*	1984	310.14	1
ta49	30×20	*1968*	1998	1829.97	1
ta50	30×20	*1928*	1958	530.20	2

References

Adams, J., E. Balas, et al. (1988). "The Shifting Bottleneck Procedure for Job Shop Scheduling." Management Science **34**(3): 391-401.

Adda, J. and R. Cooper (2003). Dynamic Economics, MIT Press.

Aggarwal, C. C., J. B. Orlin, et al. (1997). "An optimized crossover for the maximum independent set." Operations Research **45**: 226-234.

Ahuja, R. K., O. Ergun, et al. (2002). "A survey of very large-scale neighborhood search techniques." Discrete Applied Mathematics **123**(1-3): 75-102.

Ahuja, R. K., J. B. Orlin, et al. (2000). "Very large-scale neighbourhood search." International Transactions in Operational Research: 301-317.

Aiex, R. M., S. Binato, et al. (2003). "Parallel GRASP with Path-relinking for Job Shop Scheduling." Parallel Computing **29**(4): 393-430.

Akker, J. V. d., C. Hurkens, et al. (2000). "Time-indexed formulation for machine scheduling problems: column generation." INFORMS Journal on Computing **12**: 111-124.

Alvim, A. C. F., C. C. Ribeiro, et al. (2003). A hybrid improvement heuristic for the one-dimensional bin packing problem. Rio de Janeiro, Catholic University Department of Computer Science.

Amin, S. (1999). "Simulated Jumping." Annals of Operations Research **86**: 23-28.

Applegate, D., R. Bixby, et al. (1999). Finding Tours in the TSP. Bonn, Germany, Forschungsinstitut für Diskrete Mathematik, University of Bonn.

Applegate, D. and W. Cook (1991). "A Computational Study of the Job-Shop Scheduling Problem." ORSA Journal on Computing **3**(2): 149-156.

Ausiello, G., P. Crescenzi, et al. (1999). Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Berlin Heidelberg, Springer-Verlag.

Balas, E. (1985). "On the Facial Structure of Scheduling Polyhedra." Mathematical Programming Study **24**: 179-218.

Balas, E. (1998). "Disjunctive programming: Properties of the convex hull of feasible points." Discrete Applied Mathematics **89**: 3-44.

Balas, E., S. Ceria, et al. (1996). "Mixed 0-1 programming by lift-and-project in a branch-and-cut framework." Management Science **42**: 1229-1246.

Balas, E. and W. Niehaus (1998). "Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems." Journal of Heuristics **4**(2): 107-122.

Balas, E. and A. Vazacopoulos (1998). "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling." Management Science **44**(2): 262-275.

Barnhart, C., C. Hane, et al. (2000). "Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems." Operations Research **40**: 318-326.

Barnhart, C., E. L. Johnson, et al. (1998). "Branch-and-price: Column generation for solving huge integer programs." Operations Research **46**(3): 316-329.

Battiti, R. and G. Tecchiolli (1994). "The reactive tabu search." ORSA Journal on Computing **6**(2): 126-140.

Bellman, R. (1957). Dynamic Programming. Princeton, N. J., Princeton University Press.

Bennell, J. A. and K. A. Dowsland (2001). "Hybridizing Tabu Search with Optimization Techniques for Irregular Stock Cutting." Management Science **47**(8): 1160-1172.

Bertsekas, D. P. (2000). Dynamic Programming and Optimal Control, Athena Scientific.

Binato, S., W. J. Hery, et al. (2002). A GRASP for Job Shop Scheduling. Essays and Surveys on Metaheuristics C. Ribeiro and P. Hansen, Kluwer Academic Publishers**:** 59-79.

Blum, C. and A. Roli (2003). "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison." ACM Computing Surveys **35**(3): 268-308.

Boyd, E. A. (1994). "Fenchel Cutting Planes for Integer Programs." Operations Research **42**: 53-64.

Büdenbender, K., T. Grünert, et al. (2000). "A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem." Transportation Science **34**(4): 364-380.

Burke, E. K., P. I. Cowling, et al. (2001). Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem. Applications of Evolutionary Computing: EvoWorkshops 2001. E. Boers, Springer. **2037:** 203-212.

Carlier, J. (1982). "The one-machine sequencing problem." European Journal of Operational Research **11**: 42-47.

Carlier, J. and E. Pinson (1989). "An Algorithm for Solving the Job-Shop Problem." Management Science **35**(2): 164-176.

Carlier, J. and E. Pinson (1994). "Adjustments of heads and tails for the job-shop problem." European Journal of Operational Research **78**: 146-161.

Caseau, Y. and F. Laburthe (1995). Disjunctive scheduling with task intervals. Paris, France, Ecole Normale Supérieure Paris.

Cerny, V. (1985). "A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm." Journal of Optimization Theory and Applications **45**: 41-51.

Chen, B., C. N. Potts, et al. (1998). A Review of Machine Scheduling: Complexity, Algorithms and Approximability. Handbook of Combinatorial Optimization. D. Z. Du and P. M. Pardalos, Kluwer Academic Publishers. **3:** 21-169.

Chen, S., S. Talukdar, et al. (1993). Job-shop-scheduling by a team of asynchronous agent. IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control, Chambery, France.

Chu, P. C. and J. E. Beasley (1998). "A Genetic Algorithm for the Multidimensional Knapsack Problem." Journal of Heuristics **4**: 63-86.

Chvátal, V. (1973). "Edmonds polytopes and hierarchy of combinatorial problems." Discrete Mathematics **4**: 305-337.

Clements, D. P., J. M. Crawford, et al. (1997). Heuristic Optimization: A hybrid AI/OR approach. Workshop on Industrial Constraint-Directed Scheduling.

Congram, R. K. (2000). Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation. Faculty of Mathematical Studies. UK, University of Southampton.

Congram, R. K., C. N. Potts, et al. (2002). "An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem." INFORMS Journal on Computing **14**(1): 52-67.

Cook, W. and P. Seymour (2003). "Tour merging via branch-decomposition." INFORMS Journal on Computing **15**(3): 233-248.

Cotta, C. (1998). "A study of hybridisation techniques and their application." IEEE Transactions on Evolutionary Computation **1**(1): 223-224.

Cotta, C., E. G. Talbi, et al. (2005). Parallel hybrid metaheuristics. Parallel Metaheuristics, a New Class of Algorithms. E. Alba, John Wiley**:** 347-370.

Cotta, C. and J. M. Troya (2003). "Embedding Branch and Bound within Evolutionary Algorithms." Applied Intelligence **18**: 137-153.

Cowling, P. I. and R. Keuthen (2005). "Embedded local search approaches for routing optimization." Computers and Operations Research **32**(3): 465-490.

Crowder, H. P., E. L. Johnson, et al. (1983). "Solving large-scale zero-one linear programming problems." Operations Research **31**: 803-834.

Cung, V.-D., T. Mautor, et al. (1997). A Scatter Search Based Approach for the Quadratic Assignment Problem IEEE International Conference on Evolutionary Computation and Evolutionary Programming, Indianapolis, USA.

Dakin, R. J. (1965). "A tree search algorithm for mixed integer programming problems." The Computer Journal **8**: 250-255.

Danna, E., E. Rothberg, et al. (2005). "Exploring relaxation induced neighborhoods to improve MIP solutions." Mathematical Programming, Ser. A **102**: 71-90.

Dantzig, G. B. (1949). "Programming of Interdependent Activities." Econometrics **17**(3): 200-211.

Dantzig, G. B., D. R. Fulkerson, et al. (1954). "Solutions of a large-scale travelling salesman problem." Operations Research **2**: 393-410.

Dantzig, G. B. and P. Wolfe (1960). "Decomposition Principal for Linear Programs." Operations Research **8**: 101-111.

Dell'Amico, M. and M. Trubian (1993). "Applying Tabu-Search to the Job-Shop Scheduling Problem." Annals of Operations Research **41**: 231-252.

Della-Croce, F., M. Ghirardi, et al. (2004). "Recovering Beam Search: Enhancing the Beam Search Approach for Combinatorial Optimization Problems." Journal of Heuristics **10**: 89-104.

Denardo, E. V. (2003). Dynamic Programming: Models and Applications, Dover Publications.

Denzinger, J. and T. Offermann (1999). On cooperation between evolutionary algorithms and other search paradigms. 1999 Congress on Evolutionary Computation (CEC), IEEE Press.

Desrosiers, J., F. Soumis, et al. (1984). "Routing with time windows by column generation." Networks **14**: 545-565.

Dorigo, M., V. Maniezzo, et al. (1996). "The ant system: optimization by a colony of cooperating agents." IEEE Transactions on Systems, Man, and Cybernetics--Part B **26**(2): 29--41.

Dorigo, M. and T. Stützle (2002). The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances Handbook of Metaheuristics. F. Glover and G. Kochenberger. Norwell, MA, Kluwer Academic Publishers. **57:** 251-286.

Dorndorf, U., E. Pesch, et al. (2002). "Constraint propagation and problem decomposition: a pre-processing procedure for the job shop problem." Annals of Operations Research **115**: 125-145.

Dowsland, K. A., E. A. Herbert, et al. (2004). "Using tree bounds to enhance a genetic algorithm approach to two rectangle packing problems." European Journal of Operational Research.

Du, D. Z. and P. M. Pardalos, Eds. (1998). Handbook of Combinatorial Optimization, Springer.

Dumitrescu, I. and T. Stützle (2003). Combinations of local search and exact algorithms. Applications of Evolutionary Computation. G. R. Raidl, Springer. **2611:** 211-223.

El-Abd, M. and M. Kamel (2005). A taxonomy of cooperative search algorithms. Proceedings of the Hybrid Metaheuristics: Second International Workshop. M. J. Blesa, C. Blum, A. Roli and M. Sampels, Springer**:** 32-41.

Feltl, H. and G. R. Raidl (2004). An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem. ACM Symposium on Applied Computing.

Feo, T. and M. Resende (1995). "Greedy Randomized Adaptive Search Procedures." Journal of Global Optimization **6**: 109-133.

Fernandes, S. and H. R. Lourenço (2007). A GRASP and Branch-and-Bound Metaheuristic for the Job-Shop Scheduling. Evolutionary Computation in Combinatorial Optimization. C. Cotta and J. v. Hemert. Berlin / Heidelberg, Springer-Verlag. **LNCS 4446:** 60-71.

Fernandes, S. and H. R. Lourenço (2007b). Optimised Search Heuristics. Meta heurísticas, Algoritmos Evolutivos y Bioinspirados, Tenerife, España.

Fernandes, S. and H. R. Lourenço (2008). Optimised Search Heuristic Combining Valid Inequalities and Tabu Search. Hybrid Metaheuristics. Springer. **LNCS 5296:** 87-101.

Fernandes, S. and H. R. Lourenço (2008b). A Simple Optimised Search Heuristic for the Job Shop Scheduling Problem. Recent Advances in Evolutionary Computation. C. Cotta and J. v. Hemert. Berlin Heidelberg, Springer-Verlag. **SCI 153:** 203-218.

Filho, G. R. and L. A. N. Lorena (2000). Constructive Genetic Algorithm and Column Generation: an Application to Graph Coloring. APORS´2000 - The fifth conference of the association of Asian-pacific Operations Research Societies.

Fischetti, M. and A. Lodi (2003). "Local Branching." Mathematical Programming Series B **98**: 23-47.

Fisher, H. and G. L. Thompson (1963). Probabilistic learning combinations of local job-shop scheduling rules. Industrial Scheduling. J. F. Muth and G. L. Thompson. New Jersey, Prentice Hall, Englewood Cliffs**:** 225-251.

Ford, L. R. and D. R. Fulkerson (1958). "A suggested computation for maximal multicommodity network flows." Management Science **5**: 97-101.

French, A. P., A. C. Robinson, et al. (2001). "Using a Hybrid Genetic-Algorithm/Branch and Bound Approach to Solve Feasibility and Optimization Integer Programming Problems." Journal of Heuristics **7**: 551-564.

French, S. (1982). Sequencing and Scheduling: An introduction to the mathematics of the job shop. New York, Wiley.

Fukasawa, R., J. Lysgaard, et al. (2004). Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. Integer Programming and Combinatorial Optimization. J. R. Correa, A. S. Schulz and A. Sebö, Springer Berlin / Heidelberg. **3064/2004:** 1-15.

Garey, M. R. and D. S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-completeness. San Francisco, Freeman.

Ghirardi, M. and C. N. Potts (2005). "Makespan minimization on unrelated parallel machines: a Recovering Beam Search approach." European Journal of Operational Research, Special issue: Project Management and Scheduling **165**(2): 457-467.

Glover, F. (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence." Computers and Operations Research **13**: 533-549.

Glover, F. (1989). "Tabu Search - Part I." ORSA Journal on Computing **1**(3): 190-206.

Glover, F. (1990). "Tabu Search - Part II." ORSA Journal on Computing **2**(1): 4-32.

Glover, F. (1999). Scatter Search and Path Relinking. New Ideas in Optimization. D. Corne, M. Dorigo and F. Glover, McGraw-Hill.

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

Glover, F., M. Laguna, et al. (2000). "Fundamentals of Scatter Search and Path Relinking." Control and Cybernetics **39**(3): 653-684.

Goldberg, L. A., M. Paterson, et al. (2001). "Better Approximation Guarantees for Job-Shop Scheduling." SIAM Journal of Discrete Mathematics **14**(1): 67-92.

Gomory, R. E. (1958). "Outline of an algorithm for integer solutions to linear programs." Bulletin of the American Mathematical Society **64**: 275-278.

Graham, R. L., E. L. Lawler, et al. (1979). "Optimization and approximation in deterministic sequencing and scheduling: a survey." Annals of Operations Research **5**: 287-326.

Griffer, B. and G. Thompson (1960). "Algorithms for solving production scheduling problems." Operations Research **8**: 487-503.

Guignard, M. (2003). "Lagrangean Relaxation." TOP **11**(2): 151-228.

Hansen, P. and N. Mladenovic (1997). "Variable Neighborhood Search for the p-median." Location Science **5**: 207-226.

Hansen, P. and N. Mladenović (2001). "Variable neighborhood search: principles and applications " European Journal of Operational Research **130**(3): 449-467.

Hedar, A.-R. and M. Fukushima (2004). "Tabu Search directed by direct local methods for nonlinear global optimization." Elsevier Science **preprint**.

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

Held, M. and R. M. Karp (1971). "The traveling salesman problem and minimum spanning trees: part II." Mathematical Programming **1**: 6-25.

Holland, J. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, MI, University of Michigan Press.

Ibaraki, T., M. Kubo, et al. (2001). Effective Local search Algorithms for the Vehicle Routing Problem with General Time Window Constraints. MiC' 2001.

Imahori, S., M. Yagiura, et al. (2003). "Local search algorithms for the rectangle packing problem with general spatial costs." Mathematical Programming Ser. B **97**: 543-569.

Jackson, J. R. (1955). Scheduling a Production Line to Minimize Maximum Tardiness. Los Angeles, University of California.

Jain, A. S. and S. Meeran (1998). An Improved Search Template for Job-Shop Scheduling. INFORMS Spring Meeting, Montreal, Quebec, Canada.

Jain, A. S. and S. Meeran (1999). "Deterministic job shop scheduling: Past, present and future." European Journal of Operational Research **133**: 390-434.

Jain, A. S., B. Rangaswamy, et al. (2000). "New and "Stronger" Job-Shop Neighbourhoods: A Focus on the Method of Nowicki and Smutnicki." Journal of Heuristics **6**: 457-480.

Jünger, M. and S. Thienel (2000). "The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization." Software: Practice and Experience **30**(11): 1325-1352.

Karmarkar, N. (1984). "A New Polynomial Time Algorithm for Linear Programming." Combinatorica **4**(4): 373-395.

Khachiyan, L. G. (1979). "A polynomial algorithm in linear programming." Soviet Mathematics Doklady **20**: 191-194.

Kirkpatrick, S., C. D. Gelatt, et al. (1983). "Optimization by Simulated Annealing." Science **220**(4598): 671-680.

Klau, G. W., I. Ljubíc, et al. (2004). Combining a Memetic Algorithm with Integer Programming to Solve the Prize-Collecting Steiner Tree Problem. Genetic and Evolutionary Computation - GECCO 2004. K. Deb, Springer-Verlag. **3102:** 1304-1315.

Kostikas, K. and C. Fragakis (2004). Genetic Programming for Guiding Branch and Bound Search. Genetic Programming - EuroGP 2004. M. K. e. al, Springer. **3003:** 113-124.

Land, A. H. and A. G. Doig (1960). "An automatic method of solving discrete programming problems." Econometrics **28**: 497-520.

Larrañaga, P. and J. A. Lozano (2002). Estimation of distribution algorithms. A new tool for evolutionary computation. Boston, MA, Kluwer Academic.

Lawler, E. (1976). Combinatorial Optimization, Networks and Matroids, Holt, Rinehart and Winston.

Lawler, E. L., J. K. Lenstra, et al. (1993). Sequencing and scheduling: algorithms and complexity. Handbooks in Operations Research and Management Science. S. Graves, A. H. G. R. Kan and P. Zipkin. North Holland, Amsterdam, Logistics of Production and Inventory. **4:** 445-522.

Lawrence, S. (1984). Resource Constrained Project Scheduling: an Experimental Investigation of Heuristic Scheduling Techniques. Pittsburgh, Pennsylvania, Graduate School of Industrial Administration, Carnegie-Mellon University.

Lee, E. K. and J. E. Mitchell (1997). Computational Experience of an Interior-Point SQP Algorithm in a Parallel Branch-and-Bound Framework. Proceedings of High Performance Optimization Techniques, Springer-Verlag.

Lenstra, J. K., A. H. G. R. Kan, et al. (1977). "Complexity of machine scheduling problems." Annals of Operations Research **1**: 343-362.

Lin, Z.-Z., J. C. Bean, et al. (2004). "A Hybrid Genetic/ Optimization Algorithm for Finite-Horizon, Partially Observed Markov Decision Processes." INFORMS Journal on Computing **16**(1): 27-38.

Lourenço, H. R. (1995). "Job-shop scheduling: Computational study of local search and large-step optimization methods." European Journal of Operational Research **83**: 347-367.

Lourenço, H. R., O. Martin, et al. (2002). Iterated Local Search. Handbook of Metaheuristics. F. Glover and G. Kochenberger. Norwell, MA, Kluwer Academic Publishers. **57:** 321-353.

Lourenço, H. R. and M. Zwijnenburg (1996). Combining large-step optimization with tabu-search: Application to the job-shop scheduling problem. Meta-heuristics: Theory & Applications. I. H. Osman and J. P. Kelly, Kluwer Academic Publishers.

Lübbecke, M. E. and J. Desrosiers (2005). "Selected Topics in Column Generation." Operations Research **53**(6): 1007-1023.

Maniezzo, V. (1999). "Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem." INFORMS Journal on Computing **11**(4): 358-369.

Maniezzo, V. and A. Carbonaro (2000). "An ANTS heuristic for the frequency assignment problem." Future Generation Computer Systems **16**(8): 927-935.

Marino, A., A. Prugel-Bennett, et al. (1999). Improving graph coloring with linear programming and genetic algorithms. EUROGEN 99, Jyvaskyia, Finland.

Mautor, T. (2002). Intensification neighbourhoods for local search methods. Essays and Surveys in Metaheuristics. C. C. Ribeiro and P. Hansen. Norwell, MA, Kluwer Academic Publishers**:** 493-508.

Mautor, T. and P. Michelon (1997). MIMAUSA: A new hybrid method combining exact solution and local search. 2nd International Conference on Metaheuristics, Sophia-Antipolis, France.

Mautor, T. and P. Michelon (2001). MIMAUSA: an application of referent domain optimization. Avignon, France, Laboratoire d'Informatique, Université d'Avignon et des Pays de Vaucluse.

Metropolis, N., A. Rosenbluth, et al. (1956). "Equation of steady-state calculation by fast computing machines." Journal of Chemical Physics **21**: 1087-1092.

Mills, P., E. Tsang, et al. (2004). A survey of AI-based meta-heuristics for dealing with local optima in local search. Colchester, Department of Computer Science University of Essex.

Mitchell, J. E. and E. K. Lee (2001). Branch-and-bound methods for integer programming. Encyclopedia of Optimization. C. A. Floudas and P. M. Pardalos, Kluwer Academic Publishers. **II:** 509-519.

Mladenović, N. and P. Hansen (1997). "Variable neighborhood search." Computers and Operations Research **24**(11): 1097-1100.

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Pasadena, California, USA, California Institute of Technology.

Mühlenbein, H. and G. Paaβ (1996). From Recombination of Genes to the Estimation of Distributions. Parallel Problem Solving from Nature – PPSN IV, Springer, Berlin.

Nagar, A., S. S. Heragu, et al. (1995). "A meta-heuristic algorithm for a bi-criteria scheduling problem." Annals of Operations Research **63**: 397-414.

Nemhauser, G. L. and L. A. Wolsey (1988). Integer and Combinatorial Optimization, John Wiley & Sons.

Nowicki, E. and C. Smutnicki (2002). Some new tools to solve the job shop problem. Wroclaw, Poland, Institute of Engineering Cybernetics, Wroclaw University of Technology.

Nowicki, E. and C. Smutnicki (2005). "An Advanced Tabu Search Algorithm for the Job Shop Problem." Journal of Scheduling **8**: 145-159.

Nowicki, E. and C. Smutniki (1996). "A Fast Taboo Search Algorithm for the Job Shop Problem." Management Science **42**(6): 797-813.

Ozdamar, L. and G. Barbarosoglu (2000). "An integrated Lagrangean relaxation-simulated annealing approach to the multi-level multi-item capacitated lot sizing problem." International Journal of Production Economics **68**: 319-331.

Padberg, M. and G. Rinaldi (1991). "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems." SIAM Review **33**(1): 60-100.

Padberg, M. W. and S. Hong (1980). "On the Symmetric Traveling Salesman Problem: A Computational Study." Mathematical Programming Study **12**: 78-107.

Papadimitriou, C. H. and K. Steiglitz (1982). Combinatorial Optimization - Algorithms and Complexity. NJ, Prentice Hall, Englewood Cliffs.

Pedroso, J. P. (2004). Hybrid Enumeration Strategies for Mixed Integer Programming.

Pedroso, J. P. (2004b). Tabu Search for Mixed Integer programming.

Pelikan, M., D. E. Goldberg, et al. (1999). A Survey of Optimization by Building and Using Probabilistic Model. Illinois, IlliGAL, University of Illinois.

Péridy, L. and D. Rivreau (2005). "Local adjustments: A general algorithm." European Journal of Operational Research **164**: 24-38.

Pesant, G. and M. Gendreau (1996). A view of local search in constraint programming. Proceedings of Constraint Programming 1996. E. Freuder. Berlin, Germany, Springer Verlag. **1118:** 353-366.

Pesant, G. and M. Gendreau (1999). "A constraint programming framework for local search methods." Journal of Heuristics **5**: 255-279.

Pezzella, F. and E. Merelli (2000). "A tabu search method guided by shifting bottleneck for the job shop scheduling problem." European Journal of Operational Research **120**: 297-310.

Pigatti, A., M. P. d. Aragão, et al. (2005). "Stabilized Branch-and-cut-and-price for the Generalized Assignment Problem." Electronic Notes in Discrete Mathematics **19**.

Plateau, A., D. Tachat, et al. (2002). "A hybrid search combining interior point method and metaheuristics for 0-1 programming." International Transactions in Operational Research **9**: 731-746.

Puchinger, J. and G. R. Raidl (2004). An Evolutionary Algorithm for Column Generation in Integer Programming: an Effective Approach for 2D Bin Packing. Parallel Problem Solving from Nature - PPSN VIII. X. Y. e. al, Springer. **3242:** 642-651.

Puchinger, J. and G. R. Raidl (2004b). Models and algorithms for three-stage two dimensional bin packing. Vienna, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

Puchinger, J. and G. R. Raidl (2005). "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification." Lecture Notes in Computer Science **3562**.

Puchinger, J., G. R. Raidl, et al. (2004c). Solving a Real-World Glass Cutting Problem. Evolutionary Computation in Combinatorial Optimization - EvoCOP 2004. J. G. a. G. R. Raidl, Springer. **3004:** 162-173.

Raidl, G. R. (1998). An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. 1998 IEEE International Conference on Evolutionary Computation, IEEE Press.

Raidl, G. R. (2006). A Unified View on Hybrid metaheuristics. Vienna Austria, Institute of Computer Graphics and Algorithms University of Technology.

Resende, M. and C. Ribeiro (2003). Greedy Randomized Adaptive Search Procedure. Handbook of Metaheuristics. F. Glover and G. Kochenberger, Kluwer Academic**:** 219-249.

Rosing, K. E. (2000). "Heuristic concentration: a study of stage one." ENVIRON PLANN B **27**(1): 137-150.

Rosing, K. E. and C. S. ReVelle (1997). "Heuristic concentration: Two stage solution construction." European Journal of Operational Research: 955-961.

Rosing, K. E. and C. S. ReVelle (1998). "Heuristic concentration and tabu search: A head to head comparison." European Journal of Operational Research **117**(3): 522-532.

Roy, B. and B. Sussman (1964). Les problèmes d'ordonnancement avec contraintes disjonctives. Paris, SEMA, Paris.

Schaal, A., A. Fadil, et al. (1999). Meta heuristics diversification of generalized job shop scheduling based upon mathematical programming techniques. Cp-ai-or'99.

Schiavinotto, T. and T. Stützle (2004). "The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms." Journal of Mathematical Modelling and Algorithms **3**: 367-402.

Schrage, L. (1970). "Solving resource-constrained network problems by implicit enumeration: Non pre-emptive case." Operations Research **18**: 263-278.

Schrijver, A. (1986). Theory of Linear and Integer Programming, John Wiley and Sons.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. Gentilly, France, ILOG S.A.

Smyth, K., H. H. Hoos, et al. (2003). Iterated Robust Tabu Search for MAX-SAT Advances in Artificial Intelligence, 16th Conference of the Canadian Society for Computational Studies of Intelligence. Y. Xiang and B. Chaib-draa, Springer Verlag, Berlin, Germany**:** 129-144.

Staggemeier, A. T., A. R. Clark, et al. (2002). A hybrid genetic algorithm to solve a lot-sizing and scheduling problem. Conference of the International Federation of Operational Research Societies, Edinburgh, U.K.

Storer, R. H., S. D. Wu, et al. (1992). "New search spaces for sequencing problems with application to job shop scheduling." Management Science **38**(10): 1495-1509.

Stützle, T. (1999). Iterated Local Search for the Quadratic Assignment Problem. Darmstadt, Intellektik, TU.

Taillard, E. (1991). "Robust Tabu Search for the Quadratic Assignment Problem " Parallel Computing **17**: 443-455.

Taillard, E. D. (1993). "Benchmarks for Basic Scheduling Problems." European Journal of Operational Research **64**(2): 278-285.

Taillard, É. D. (1994). "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem." ORSA Journal on Computing **6**(2): 108-117.

Tamura, H., A. Hirahara, et al. (1994). "An approximate solution method for combinatorial optimisation." Transactions of the Society of Instrument and Control Engineers **130**: 329-336.

Thompson, P. and J. Orlin (1989). The theory of cycle transfers. Boston, MIT Operations Research Center.

Thompson, P. and H. Psaraftis (1993). "Cycle transfer algorithm for multivehicle routing and scheduling problems." Operations Research **41**: 935-946.

Umetani, S., M. Yagiura, et al. (2003). "One Dimensional Cutting Stock Problem to Minimize the Number of Different Patterns." European Journal of Operational Research **146**: 146.

Vaessens, R. J. M., E. H. L. Aarts, et al. (1996). "Job Shop Scheduling by Local Search." INFORMS Journal of Computing **8**: 302-317.

Van-Laarhoven, P. J. M., E. H. L. Aarts, et al. (1992). "Job Shop Scheduling by Simulated Annealing." Operations Research **40**: 113-125.

Vanderbeck, F. (1998). "Lot-sizing with start-up times." Management Science **44**: 1409-1425.

Vanderbeck, F. and L. A. Wolsey (1996). "An exact algorithm for IP column generation." Operations Research Letters **19**: 151-159.

Vasquez, M. and J.-K. Hao (2001). A Hybrid Approach for the 0-1 Multidimensional Knapsack problem. International Joint Conference on Artificial Intelligence 2001.

Voudouris, C. (1997). Guided Local Search for Combinatorial Optimisation Problems. Department of Computer Science. Colchester, UK, University of Essex.

Voudouris, C. and E. Tsang (1999). "Guided local search." European Journal of Operational Research **113**(2): 469-499.

Woodruff, D. L. (1999). A chunking based selection strategy for integrating meta-heuristics with branch and bound. Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization. S. V. e. al, Kluwer Academic Publishers**:** 499-511.

Yagiura, M. and T. Ibaraki (1996). "The use of dynamic programming in genetic algorithms for permutation problems." European Journal of Operational Research **92**: 387-401.

Yamada, T. and R. Nakano (1992). A genetic algorithm applicable to large-scale job-shop problems. Parallel Problem Solving from Nature 2. R. Manner and B. Manderick. Brussels Belgium, Elsevier Science**:** 281-290.

Yamada, T. and R. Nakano (1995). Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search Metaheuristics International Conference, Hilton, Breckenridge, Colorado, USA.